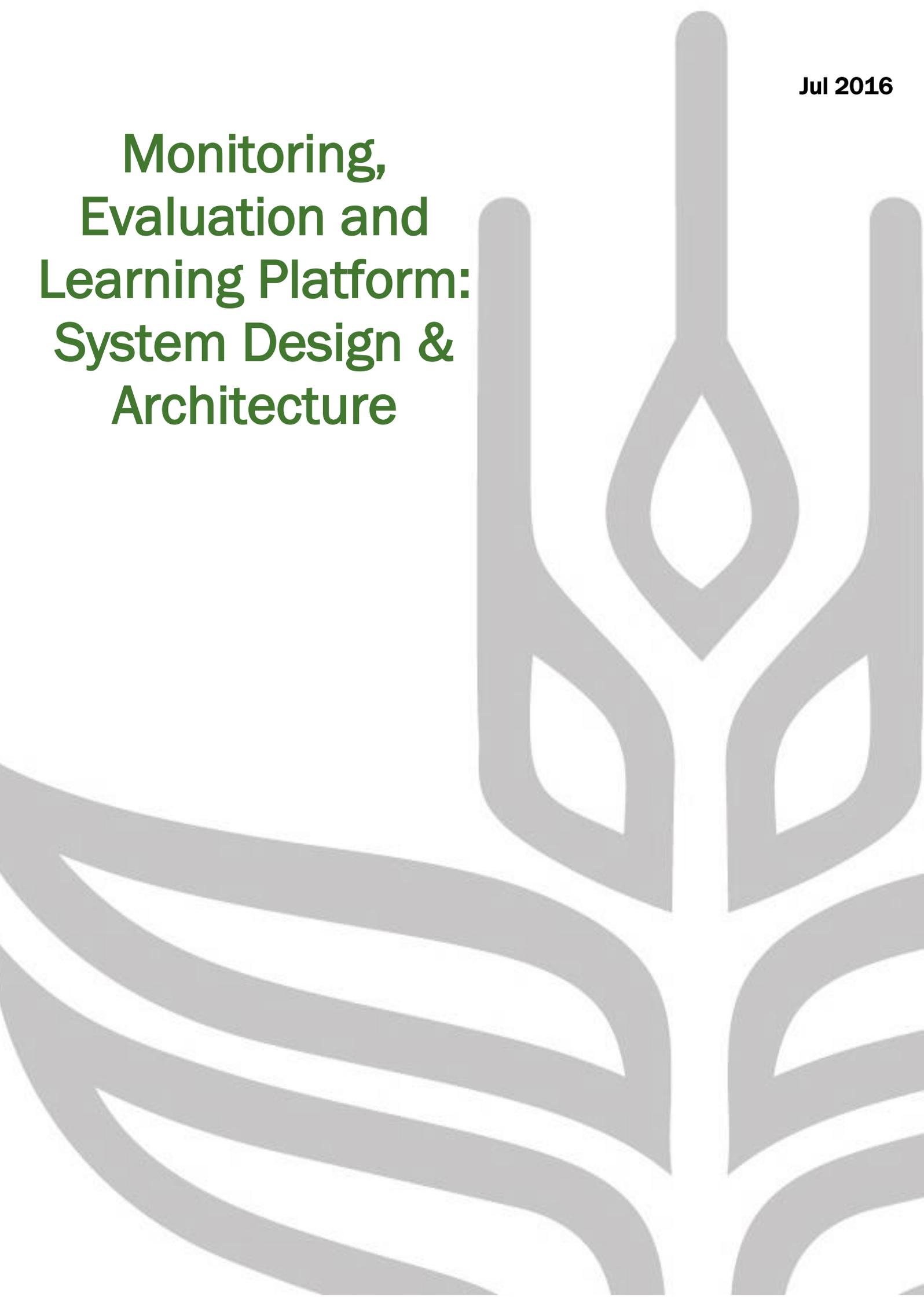


Jul 2016

Monitoring, Evaluation and Learning Platform: System Design & Architecture





RESEARCH
PROGRAM ON
Dryland Systems



RESEARCH
PROGRAM ON
Dryland Cereals



RESEARCH
PROGRAM ON
Grain Legumes



RESEARCH
PROGRAM ON
Roots, Tubers
and Bananas



ICARDA
Science for Better Livelihoods in Dry Areas



Code**Obia**



powered by
amazon
web services

AUTHORS

Aya Mousa, Bashar Ayyash, Mehtab Khan, Victor Kimathi (1);

CONTRIBUTORS

Jalal Eddin Omary, Chandrashekhara Biradar (2); Patricia Bravo, Claudio Proietti (3), Percy Cabello (4); Moayad Al-Najdawi (5); Belal Mazlom, Enrico Bonaiuti, Valerio Graziano (6);

SUGGESTED CITATION

CRP DS, CRP RTB, CRP DC, CRP GL 2016. Monitoring, Evaluation and Learning Platform: System Design & Architecture.

DISCLAIMER

The views expressed in this document do not necessarily reflect the views of the CGIAR System Organization.



This document is licensed for use under the Creative Commons Attribution 3.0 Unported Licence. To view this licence, visit <http://creativecommons.org/licenses/by-nc-sa/3.0/>

Unless otherwise noted, you are free to copy, duplicate, or reproduce and distribute, display, or transmit any part of this publication or portions thereof without permission, and to make translations, adaptations, or other derivative works under the following conditions:



ATTRIBUTION. The work must be attributed, but not in any way that suggests endorsement by the publisher or the author(s).

¹iMAP

²International Center for Agricultural Research in Dry Areas (ICARDA)

³CGIAR Research Program on Roots, Tubers and Bananas (RTB)

⁴ International Potato Center (CIP)

⁵ CodeObia

⁶ CGIAR Research Program on Dryland Systems (DS)



Revision History

Table 1. Revision Summary

Version	Date	Author(s)	Description
0.1	01-Mar-16	Mehtab Khan	Baseline Version
0.2	15-Mar-16	Enrico Bonaiuti	Business Logic
1.0	04-Apr-16	Aya Mousa	Alpha Compilation
1.0	05-Apr-16	Victor Kimathi	Alpha Finalization
2.0	25-Apr-16	Aya Mousa	Beta Compilation
2.5	10-May-16	Bashar Ayyash Aya Mousa	Final Compilation
2.6	16-Jul-16	Enrico Bonaiuti	CGIAR Branding
2.7	9-Aug-16	Valerio Graziano	CGIAR System Organization Branding



Table of Contents

List of Tables.....	7
List of Figures.....	7
1. Introduction	10
1.1 Document Overview	10
1.2 Scope.....	12
1.3 Audience	12
1.4 Related Documentation.....	13
1.5 Document Conventions	14
2. General Overview and Approach.....	17
2.1 General Overview	17
2.1.1 Overview of the System.....	17
2.1.2 System Requirements.....	18
2.2 Business Processes.....	20
2.3 Assumptions / Constraints / Risks.....	20
2.3.1 Assumptions.....	20
2.3.2 Constraints.....	20
2.3.3 Risks	23
3. Design Considerations	25
3.1 Goals and Guidelines.....	25
3.2 Development Methods & Contingencies.....	27
3.3 Architectural Strategies	27
4. System Architecture	29
4.1 Hardware Architecture	31
4.1.1 Performance Hardware Architecture	31
4.2 Software Architecture.....	31
4.2.1 Performance Software Architecture.....	33
4.3 Information Architecture	34
4.4 Internal Communications Architecture	35
4.5 System Architecture Diagram	41
5. System Design	43
5.1 Business Requirements.....	43
5.2 Database Design.....	46
5.2.1 Data Objects and Resultant Data Structures	49
5.3 File and Database Structures	50



5.3.1 Database Management System Files	52
5.3.2 Non-Database Management System Files	61
5.4 Database Information	62
5.5 Data Conversion	62
5.6 User Interface Design	62
6. Operational Scenarios.....	64
6.1 Operational Modes	64
6.2 Operational Scenarios	65
6.3 Data Flow Diagrams	76
7. Detailed Design	82
7.1 Quick Installation Guide	82
7.2 Conceptual Infrastructure Design	83
7.3 Hardware Detailed Design	83
7.4 Application Users	84
7.4.1 Inputs	85
7.4.2 Outputs	85
7.5 Software Detailed Design.....	86
7.5.1 Authentication (Zend_Auth)	86
7.5.2 Authorization (Zend_Acl)	90
7.5.3 System Layout (Zend_Layout).....	104
7.5.4 CRUD (Create, Retrieve, Update, Delete)	143
7.6 Software Functions	158
7.7 Security Detailed Design	158
7.8 Performance Detailed Design	160
7.9 Software Components Off the Shelf (COTS)	161
7.10 Achievement of functional requirements.....	161
8. System Integrity Controls.....	165
9. External Interfaces.....	167
9.1 Interface Architecture.....	167
9.2 Interface Detailed Design.....	169
Appendix A: Acronyms	171
Appendix B: Glossary	172
Appendix C: Approvals	175
Appendix D: Security Architecture	176
D.1 Security Policy.....	177
D.1.1 Security Development	177
D.1.1.1 Security Analysis	177
D.1.1.2 Cryptography	178



Monitoring, Evaluation and Learning Platform: System Design & Architecture

D.1.1.3 Risk Assessment.....	178
D.1.2 Security Infrastructure.....	179
D.2 Security Analysis.....	179
Appendix E: Performance.....	181
E.1 Performance	181
E.1.1 Static Content.....	182
E.1.2 Business Logic	184
E.1.3 Storage	184
E.2 Scalability	184



List of Tables

Table 1. Revision Summary	3
Table 2. MEL System Related Documentation.....	13
Table 3. MEL System Requirements Mapping	19
Table 4. Application Context Mapping.....	30
Table 5. Business Process 0: Project Definition	43
Table 6. Business Process 1: Project Planning and Monitoring	43
Table 7. Business Process 2: Financial Accounting	44
Table 8. Business Process 3: Human Resources Allocation	44
Table 9. Business Process 4: Surveys	45
Table 10. Business Process 5: Impact Pathway.....	45
Table 11. Business Process 6: Knowledge Sharing.....	45
Table 12. Database Naming Convention.....	49
Table 13. Code Naming Convention	49
Table 14. Application Locations	83
Table 15. Role and Resource Mapping	84
Table 16. MEL Access Controls.....	92
Table 17. headMeta() types	106
Table 18. url function parameters	110
Table 19. Feature #1 Project Management	158
Table 20. Workflow / Sequence 0: Project Definition	161
Table 21. Workflow / Sequence 1: Project Planning and Monitoring	162
Table 22. Workflow / Sequence 2: Financial Accounting	162
Table 23. Workflow / Sequence 3: Human Resources Allocation	163
Table 24. Workflow / Sequence 4: Surveys	163
Table 25. Workflow / Sequence 5: Impact Pathway.....	163
Table 26. Workflow / Sequence 6: Knowledge Sharing.....	164
Table 27. Acronyms.....	171
Table 28. Glossary	172
Table 29. MEL System Analysis	179
Table 30. Web Application Levels, Core Technologies, and Performance and Scalability Techniques	182

List of Figures

Figure 1. Semantic Versioning Numbers.....	12
Figure 2. The MEL System Used Flowcharts Symbols and Meanings	14
Figure 3. Context Diagram Symbols and Meanings	15
Figure 4. The MEL System Used Data Flow Diagram Notations	15
Figure 5. UML Diagram	16
Figure 6. MEL System High-level Context Diagram.....	17
Figure 7. MEL System Focus.....	18
Figure 8. Zend MVC Design Pattern Structure	26
Figure 9. MEL Application Context Mapping	29
Figure 10. File caching inside PlanningController class	34
Figure 11. MEL System Information Architecture General-to-Specific Approach	35
Figure 12. DSpace Internal Communication	37
Figure 13. Controller to Action Internal Communication	38
Figure 14. Controller to View Internal Communication.....	38



Figure 15. System Architecture.....	41
Figure 16. Sample Extracted Database Model.....	46
Figure 17. Columns in tbl_activity.....	47
Figure 18. Server validation rule for activity_id field in tbl_activity.....	47
Figure 19. Model level diagram for MainDB.....	48
Figure 20. Directory Structure (Application).....	50
Figure 21. Directory Structure (Assets).....	50
Figure 22. Directory Structure (Documents).....	51
Figure 23. Directory Structure (KML).....	51
Figure 24. Directory Structure (Library).....	51
Figure 25. Directory Structure (Uploads).....	52
Figure 26. Directory Structure (Root).....	52
Figure 27. Database Tables.....	53
Figure 28. Action site Tables Relationships.....	53
Figure 29. Country Tables Relationships.....	54
Figure 30. Discussion Tables Relationship.....	54
Figure 31. Field site Tables Relationship.....	55
Figure 32. Flagship Tables Relationships.....	55
Figure 33. Flagship Activity Tables Relationships.....	55
Figure 34. Flagship Activity Details Tables Relationship.....	57
Figure 35. IDO Tables Relationship.....	59
Figure 36. Partner Tables Relationship.....	59
Figure 37. Project Tables Relationship.....	60
Figure 38. Reports Tables Relationships.....	60
Figure 39. User Tables Relationship.....	61
Figure 40. User Interface.....	63
Figure 41. MEL System Operational Scenarios Sorted Based on Roles.....	64
Figure 42. MEL System Operational Scenarios Sorted Based on Actions.....	65
Figure 43. Creation of Partners and Contacts Operation.....	65
Figure 44. Creation of Users Operation.....	66
Figure 45. Creation of ALS Operation.....	66
Figure 46. Creation of IDO Operation.....	67
Figure 47. Creation of Flagships Operation.....	67
Figure 48. Creation of Action Sites/Cluster of Activities Operation.....	68
Figure 49. Creation of Projects Agreements Operation.....	68
Figure 50. Creation of Projects Operation.....	69
Figure 51. Creation of Activities Operation.....	69
Figure 52. Editing Projects Information Operation.....	70
Figure 53. Editing Activities Information Operation.....	70
Figure 54. Planning within Projects and Activities Operation.....	71
Figure 55. Reporting Operation.....	71
Figure 56. Self-Assess a project Operation.....	72
Figure 57. Launching and Reviewing a Survey Operation.....	72
Figure 58. Consulting Overview Operation.....	73
Figure 59. Consulting Open Facts Operation.....	73
Figure 60. Exporting Data Operation.....	74
Figure 61. FP and CoA Leaders Approval Operation.....	74
Figure 62. Partners and Contacts Approval Operation.....	75



Figure 63. Open Access Approval Operation	75
Figure 64. Discussion Forum Operation	76
Figure 65. Technical Assistance Request Operation	76
Figure 66. Activity/Project Creation Data Flow Diagram	77
Figure 67. Pre-planning, Exporting, Open Facts Consulting Data Flow Diagram for Administrator, Activity/Project Leader, Cluster of Activity/Action Site Leader, and Flagship Leader	78
Figure 68. Reporting Data Flow Diagram for Administrator, Activity/Project Leader, Cluster of Activity/Action Site Leader, and Flagship Leader	79
Figure 69. Open Access Approval data flow diagram for Administrator, Activity/Project Leader, Cluster of Activity/Action Site Leader, and Flagship Leader	80
Figure 70. Editing Projects/Activities Data Flow Diagram for Administrator, Activity/Project Leader, Cluster of Activity/Action Site Leader, and Flagship Leader	81
Figure 71. MEL System Default Layout	105
Figure 72. Horizontal Menu	114
Figure 73. Search Box.....	117
Figure 74. User Menu Top Navigation Menu.....	118
Figure 75. Discussion Live Box	120
Figure 76. HTML Footer	122
Figure 77. Route Convention	123
Figure 78. Request Cycle through Zend Application.....	123
Figure 79. Multiple templates are used to build up complete page	124
Figure 80. View folder structure	125
Figure 81. Overview main page (http://mel.cgiar.org/overview)	126
Figure 82. User Index Page (http://mel.cgiar.org/user)	131
Figure 83. Login View Page (http://mel.cgiar.org/user/login).....	132
Figure 84. View When the System Is Down (http://mel.cgiar.org/user/message)	134
Figure 85. Login view (http://mel.cgiar.org/user/pdgmlogin)	134
Figure 86. Reset Password View (http://mel.cgiar.org/user/reset)	135
Figure 87. Input fields that will be validated before the form will be submitted to the action	142
Figure 88. Data Mapper Representative Flow (Flower, 2002).....	143
Figure 89. Table Data Gateway Representative Flow	144
Figure 90. Security Detailed Diagram	159
Figure 91. Performance Detailed Design	160
Figure 92. Process of Getting an Open Access Approval for a Document.....	166
Figure 93. DSpace System Architecture (Bass, Stuve, & Tansley, 2002).....	168
Figure 94. DSpace Conceptual Model (http://cs.calstatela.edu).....	169
Figure 95. Page Components	181



1. Introduction

Instructions: Provide identifying information for the existing system (e.g., the full names and acronyms for the existing system), and expected evolution of the document. Also describe any security or privacy considerations associated with use of this document.

The web-based Monitoring, Evaluation and Learning (MEL) platform enables better result-based management including planning, reporting, coordination, risk management, performance evaluation, management of legal mechanisms in place among partners, as well as knowledge sharing and learning amongst different groups of stakeholders (donors, partners, project/program implementers, managers and collaborators) within and across Projects and Programs. The MEL platform was initially developed by the in-house team of the CGIAR Research Program on Dryland Systems and launched at the end of 2014.

The system shall undergo further development and expansion over the coming period in order to adapt to partners and donors needs in the framework of the Development Assistance Committee (DAC) of the Organization for Economic Co-operation and Development (OECD) Official Development Assistance (ODA).

This document is a living document and template; it shall be updated over the lifetime of iMMAP's engagement with ICARDA as the system evolves. The document is used by all Partners⁷ customizing and developing the system in order to support broad understanding and report funds allocated for more efficient and effective Monitoring and Evaluation. It will serve as a guide to document the system and its evolution.

The present document is designed to be open access and accessible to all stakeholders in order to increase customization and adaptation of different institutions partners of this initiative. As per CGIAR Policy⁸ the information products including codes and software should be accessible and use granted free of cost using appropriate licence (<https://creativecommons.org>).

1.1 Document Overview

Instructions: Describe the document.

This dual-use document has been developed by iMMAP Inc. for the Dryland Systems Program led by the International Centre for Agricultural Research in Dry Areas (ICARDA) and its partners for their Monitoring and Evaluation System. This document was developed from review of components of the system by iMMAP staff (Aya Mousa, Bashar Ayyash and Victor Kimahi) and the original system developer; former ICARDA staff (Jalal Omary Eddin), and is intended to satisfy the customers' requirements, objectives and expectations as:

- A Technical Documentation of the system, and
- A guidance for future development of the system

Although this document will go into some detail about system specifications and technical details, developers are encouraged to use their best judgment when making implementation decisions. Remaining sections of this document are organized as follows:

⁷2016: MEL System development partners: CGIAR Research Program on Roots, Tubers and Bananas (RTB) - <http://www.rtb.cgiar.org>; International Potato Center (CIP) - <http://cipotato.org>; International Center for Agricultural Research in Dry Areas (ICARDA) - <http://icarada.org>; IMMAP - <http://www.immap.org>; CODEOBIA - <http://codeobia.com>, CGIAR Dryland Cereals (DC) <http://drylandcereals.cgiar.org>; CGIAR Grain Legumes (GL); <http://grainlegumes.cgiar.org>.

⁸ <https://library.cgiar.org/bitstream/handle/10947/2875/CGIAR%200A%20Policy%20-%20October%202%202013%20-%20Approved%20by%20Consortium%20Board.pdf?sequence=4>



- **Section 2:** General Overview and Approach: Provides an overview of the system and software architectures and the design goals.
- **Section 3:** Design Considerations: Describes issues which were addressed or resolved when designing the system, along with the development methods and, architectural strategies.
- **Section 4:** System Architecture: Describes how the functionality and responsibilities of the system were partitioned and then assigned to subsystems or components, and how the application interacts with other applications.
- **Section 5:** System Design: Describes the business requirements, database design and management system.
- **Section 6:** Operational Scenarios: Describes the general functionality of the system from the users' perspectives.
- **Section 7:** Detailed Design: Provides the information needed for a system development team to actually build and integrate the hardware components, code and integrate the software components, and interconnect the hardware and software segments into a functional product.
- **Section 8:** System Integrity Controls: Provides design specifications for certain levels of control.
- **Section 9:** External Interfaces: Describes any interfaces that exist with external systems that are not within the scope of the system.
- **Appendix A:** Acronyms: Lists acronyms and associated literal translations used in this document.
- **Appendix B:** Glossary: Defines terms used in this document.
- **Appendix C:** Approvals.
- **Appendix D:** Security Architecture.
- **Appendix E:** Performance.



1.2 Scope

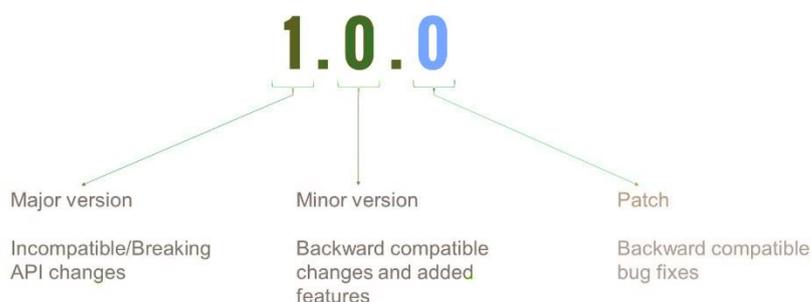
Instructions: Provide a summary of models and versions of software to which this document relates.

Release management are not yet applied to the system, though we encourage ICARDA to start applying GIT for this. The current MEL system release is V1.0.0, upcoming releases will follow semantic versioning as shown in (Fig. 1).

Figure 1. Semantic Versioning Numbers

MEL

Version Syntax



Given a version number MAJOR. MINOR. PATCH, increment the:

1. MAJOR version when the new API is not back compatible with and the previously shipped API. It does not matter how different it is, if the API acts differently, change the MAJOR version.
2. MINOR version when the API is changed, but it is completely back compatible with previous versions of this MAJOR release. Use this when ADDING features to API, too.
3. PATCH version when made bug fixes do not affect the API.

To keep iMMAP's, CODEOBIA's, DS/ICARDA's, RTB/CIP's, DC's, and GL's development teams' commits organized, and meaningful, Vincent branching model will be followed in both changelog and releases.

1.3 Audience

Instructions: State the skills required and assumptions.

This document is developed for two primary audiences:

- **Business Owners:**
Those with intimate knowledge of the functional requirements of the users of the system, i.e. Project Managers, Research for Development Organisations involved in ODA interventions,



Monitoring, Evaluation and Learning Platform: System Design & Architecture

Donors, Academia and Advance Research Institutions involved in development project/program implementation.

- **System Developers:**

Those charged with developing the system further. It is assumed they are conversant with the following:

- MVC software architecture (Zend Framework).
- Standard system design documentation.
- PHP, JavaScript, AJAX, JQuery, and MySQL RDBMS.

1.4 Related Documentation

Instructions: List related documents including supplier documentation, test plans and results as appropriate for this document; List any naming standard of common business process documents to guide. List any supporting Interface Control Documents, and indicate how to obtain these.

MEL System documentation includes the following:

Table 2. MEL System Related Documentation

Document Title	Related To	Source
Metronic User Guide	Global system template	http://metronicwordpress.com/documentation
PHP Manual	Programming language	http://php.net/manual/en/index.php
Zend Framework Manual	Framework	http://framework.zend.com/manual/1.12/en/manual.html
jQuery API Documentation	Other Libraries	https://api.jquery.com
DSpace	Digital resources open access package	https://wiki.duraspace.org
Pest	RESTful web services	https://github.com/educoder/pest
SimpleHtmlDom	A HTML DOM parser	http://simplehtmldom.sourceforge.net/manual.htm
PHPExcel	Spreadsheet engine	https://phpexcel.codeplex.com
ImageLib	Image manipulation	http://wideimage.sourceforge.net/documentation
Modules	Supplier documentation	ICARDA
Database design physical report	Database	The Full Physical Report.html in the root of this documentation delivery
Vincent Branching Model	System Semantic Versioning	http://nvie.com/posts/a-successful-git-branching-model
Theme Support	User Interface Theme	http://themeforest.net/item/metronic-responsive-admin-dashboard-template/4021469?ref=keenthes
CG Core Metadata Schema (V3)	Information Architecture	http://mel.cgiar.org/xmlui/handle/20.500.11766/4764

Document Title	Related To	Source
Reference for Information Architecture	Information Architecture	http://web.mit.edu/dspace/live/implementation/design_documents/architecture.pdf
Security Architecture	Security Architecture	Appendix D
Performance	Performance	Appendix E

1.5 Document Conventions

Instructions: Describe what diagrammatic notation has been used in this document to represent architectural views.

The following standards are applied to the document:

- **Tables and Figures (Sequential Numbering):**

Tables and Diagrams are numbered sequentially e.g. Table 1, Figure 2 etc.

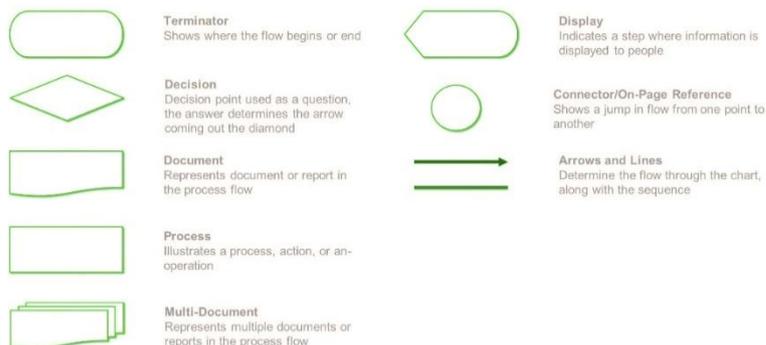
- **Flowcharts (Standard Notation Flow Chart):**

A flowchart is a visual representation of the sequence of steps and decisions needed to perform a process. Each step in the sequence is noted within a diagram shape. Steps are linked by connecting lines and directional arrows. This allows anyone to view the flowchart and logically follow the process from beginning to end. The MEL System used flowcharts symbols and meanings can be found in (Fig. 2).

Figure 2. The MEL System Used Flowcharts Symbols and Meanings

MEL

Used Flowcharts symbols and meanings



- **Context Diagram:**

A high-level view of the overall business or system boundaries. It identifies the external entities along with major data interfaces that interact with the target single high-level process. This process centric diagram symbols and meanings are shown in (Fig. 3).

Figure 3. Context Diagram Symbols and Meanings

MEL

Context diagrams symbols and meanings



Circle:
Represents the high-level single process that transforms input into output.



Rectangle:
Represents external entity which is people, places, or things provide data to the system or receive data from the system.



Arrows:
Represents data flow between the process and external entities. This should be named to identify the piece of data.

- **Data Flow Diagrams:**

A data flow diagram (DFD) illustrates how data is processed by a system in terms of inputs and outputs. As its name indicates, its focus is on the flow of information, where data comes from, where it goes and how it gets stored.

MEL is using Yourdon & Coad's (YC) Object Oriented Analysis and Design (OOA/OOD). The currently used YC notations are shown in (Fig. 4).

Figure 4. The MEL System Used Data Flow Diagram Notations

MEL

Used data flow diagram notations (Yourdon & Coad)



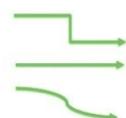
Process
Transforms incoming data flow into outgoing data flow.



External Entity
A source or destination of a system inputs or outputs.



Data Store
Repositories of data in the system, referred to as files.



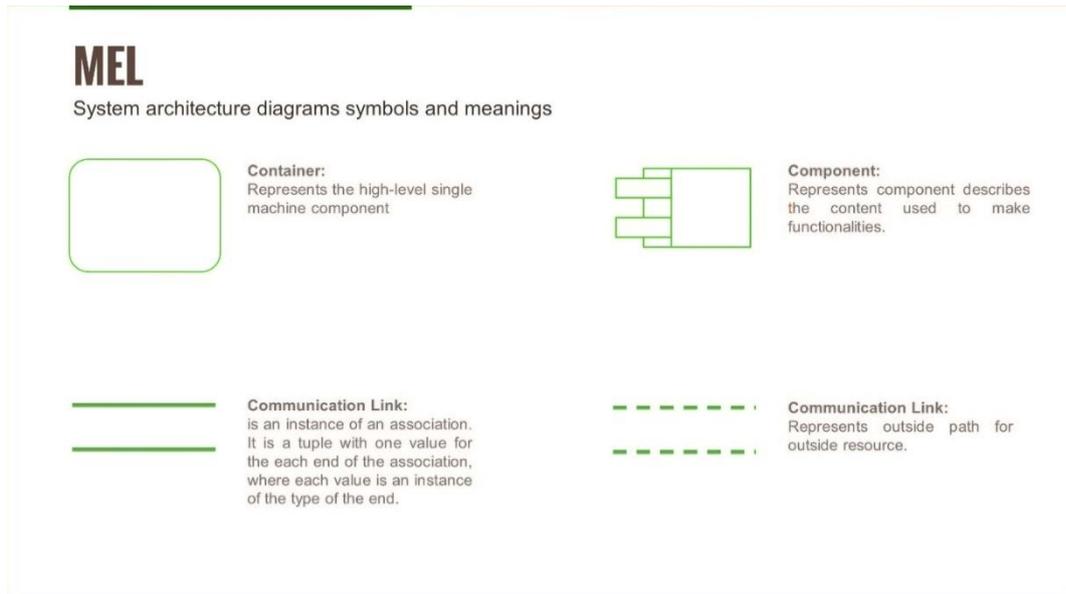
Data Flow
Pipelines through which packets of information flow. Usually labelled with the name of the data that moves through it.

- **UML Diagram:**

Represents elements of systems and organize them into related groups to minimize dependencies between system components.

This diagram is used to describe the architecture of MEL system and the interaction between its component.

Figure 5. UML Diagram



2. General Overview and Approach

2.1 General Overview

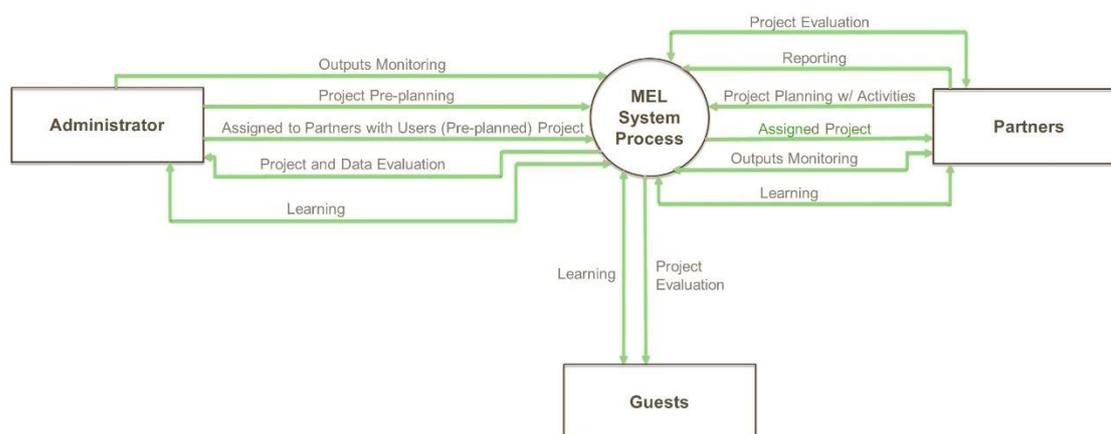
Instructions: Briefly introduce the system context and organisation. Provide a brief overview of the system and software architectures and the design goals. Include the high-level context diagram(s) for the system and subsystems.

Over the years, Dryland Systems has needed a Monitoring and Evaluation system to provide a central location for projects and scientists from all over the world to record their activities and to manage funds for results in efficient and effective manners in order to maximize taxpayers' funds invested in ODA. To this end, the MEL system was developed in-house by the Lead Centre, ICARDA, which has an advance unit for Geo-Informatics Unit (<http://geoagro.icarda.org>) with experienced staff to integrate programming skills, scientific software knowledge and practical experience in research for development projects and programs.

Figure 6. MEL System High-level Context Diagram

MEL

System context diagram



2.1.1 Overview of the System

Instructions: A brief functional description with key concepts. Provide a top-level description of the system and its major external interfaces to aid the reader in understanding what the software is to accomplish. Reference appropriate graphics, illustrations, tables etc. to show functions.

- In what environment it works
- Who the users are
- What it is for
- The main functions
- The main interfaces, inputs and outputs

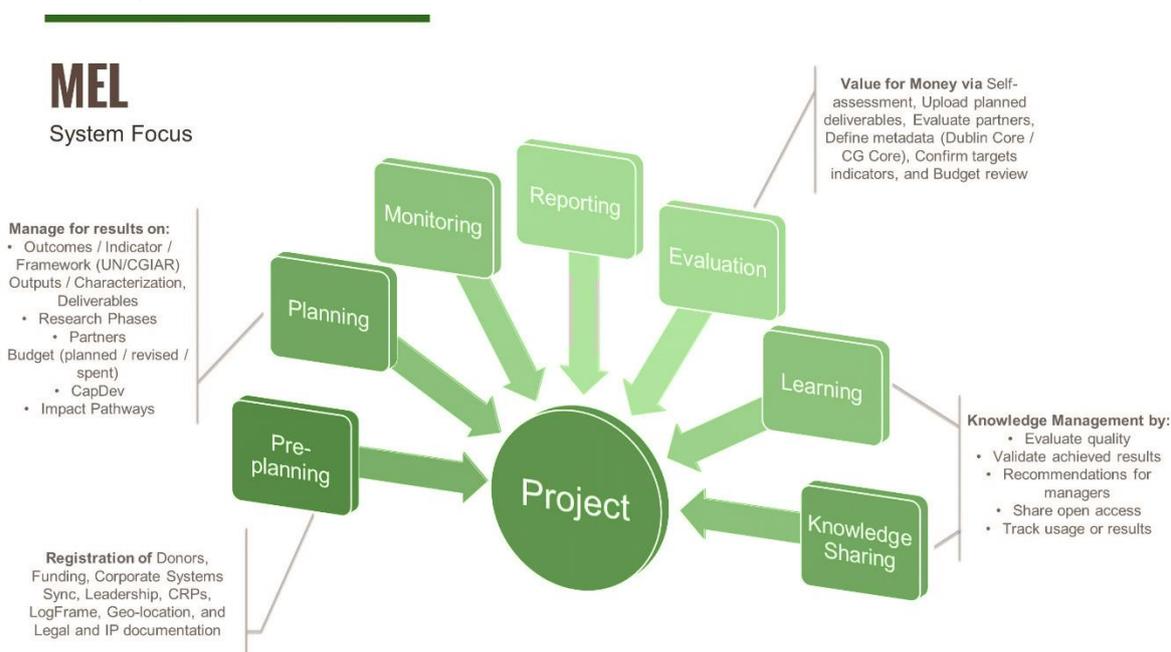
The system is designed for managers and staff in projects and programs. It reduces users reporting time while aggregating information across well-known indicators in order to manage resources

effectively and efficiently. It allows manager to identify resources, partnerships and plan for results while predicting a shortfall of planned deliverables in order to re-allocate projects/programs elements. It generates knowledge sharing through discussion forum and keywords analysis.

The system consists of several modules, each module completing a specific task. The system monitors project deliverables, participation, and feeds back to the overall goals of the project allowing for monitoring of the impact of the projects within a Program.

There are several modules on Project information: Staff, consultants allocation and productivity, partners engagement and evaluation, research output phases from discovery to scaling up and out, outputs and deliverables planned, un-expected and completed, outcomes risks and assumptions, budget allocation and burn rate, outcomes-based budget, capacity development, gender, intellectual assets monitoring, reporting, interoperable repository with Dublin core metadata, discussion forum, partners mapping, projects interaction and clustering, survey tool, open facts and dynamic impact pathways.

Figure 7. MEL System Focus



2.1.2 System Requirements

Instructions: Include a description of each major requirement in narrative and include a table of the mapping of requirements to modules and current status: These should include significant functional workload and functional performance, operational, technical, security and any other special requirements.

Requirement 0: Project Definition

The system is expected to register and archive all project information and display them for comprehensive knowledge.

Requirement 1: Project Monitoring

The system is expected to allow manager to plan for results and monitor the products and deliverables from projects and at programs' level. The products may take different forms, the



system focuses on electronic ones, such as data. Electronic Deliverables such as reports are also stored on the system allowing for reference.

Requirement 2: Financial Accounting

The System monitors budget allocation and burn rate in projects at country level, outcome, output and key sectors (i.e. Gender/Capacity Development) levels. This information is entered in tables and processed by the system in order to produce informative reports to managers to take strategic decision on resources allocation.

Requirement 3: Human Resources Allocation

The System monitors scientists and consultants time allocated in projects. This information is selected by Project Managers and processed by the system in order to produce informative reports to managers to take strategic decision on resources allocation and staff (user) performance evaluation.

Requirement 4: Surveys

The System sends surveys to selected users and contacts to assess their level of knowledge, Skills and Attitude (KSA) in order to plan future interventions or assess the impact of ongoing activities. This information is organized by users and launched by the system via email in order to produce informative reports to managers to take strategic decision.

Requirement 5: Impact Pathway

The System generates dynamic impact pathways aggregating information from Projects, sites, countries and regions. This information is entered by users and the system produce visual pathways to link outputs, outcomes, risks, assumptions, indicators and strategic result framework contribution. It is used to analyse effects of integrated interventions to identify synergies and redundancies.

Requirement 6: Knowledge Sharing

The System has a build in discussion forum linked with research outputs and deliverables to share opinion and improve quality. The reporting module provide a standard metadata schema (Dublin core) and it is linked to an external open access repository (D-Space) to share knowledge with existing partners and broadly with stakeholders.

Table 3. MEL System Requirements Mapping

Requirement	Details	Modules
Project Monitoring	Electronic Product Warehousing Electronic Deliverable Warehousing	Project Manage: Results (Outputs) Reporting Capacity Development
Financial Accounting	Electronic data warehousing	Project Manage: Budget Project Info
Human Resources Allocation	Electronic data warehousing	User Project Manage: Scientists and consultants
Surveys	Electronic data warehousing	Survey

Requirement	Details	Modules
Impact Pathway	Electronic data warehousing	Project Manage: Outputs Project Manage: Outcomes Project Manage: Budget Project Manage: SRF mapping Project Manage: Outcome Indicators Project Manage: Related Outputs Project Manage: Related Outcomes Project Manage: Risks and Assumptions
Knowledge Sharing	Electronic Deliverable Warehousing	Project Reporting D-Space

2.2 Business Processes

Instructions: Briefly describe the various business processes at a system overview level; descriptions of the specific business process will be tackled later in this document.

The overall business process of the system is monitoring and evaluation of projects and programs. This includes monitoring of deliverables, mapping of impacts, financial accounting and reporting to various stakeholders.

2.3 Assumptions / Constraints / Risks

2.3.1 Assumptions

Instructions: Describe any assumptions or dependencies regarding the system and its use. These may concern such issues as related software or hardware, operating systems, end-user characteristics, and possible and/or probable changes in functionality.

The usual assumptions on user, system administrators and developers' prerequisite knowledge are taken for granted. The system is known to require module addition and evolution in the coming years.

2.3.2 Constraints

Instructions: Describe any limitations or constraints that have a significant impact on the system, software and/or communications, and describe the associated impact. Such constraints may be imposed by any of the following (the list is not exhaustive):

- a) Hardware or software environment
- b) End-user environment
- c) Availability or volatility of resources
- d) Standards compliance
- e) Interoperability requirements
- f) Interface/protocol requirements
- g) Licensing requirements
- h) Data repository and distribution requirements
- i) Security requirements (or other such regulations)
- j) Memory or other capacity limitations
- k) Performance requirements
- l) Network communications



Monitoring, Evaluation and Learning Platform: System Design & Architecture

- m) *Verification and validation requirements (testing)*
- n) *Other means of addressing quality goals*
- o) *Other requirements described in the Requirements Document*

Zend Framework requires a PHP 5 interpreter with a web server configured to handle PHP scripts correctly. Zend recommends the most current release of PHP for critical security and performance enhancements, and currently supports PHP 5.2.11 or later.

a) **Hardware or software environment**

MEL web server should have the mod_rewrite module installed and enabled. To prevent files manipulation processes from getting the service down, a max_execution_time should be increased to 360 ms. Other PHP configurations are:

memory_limit = 128 MB

upload_max_filesize = 200 MB

b) **End-user environment**

A basic compatible browser with JavaScript is needed, IE is not recommended, as it is not fully compatible with MEL.

c) **Availability or volatility of resources**

No limitations, nor constraints are found.

d) **Standards compliance**

No limitations, nor constraints are found.

e) **Interoperability requirements**

No limitations, nor constraints are found.

f) **Interface/protocol requirements**

No limitations, nor constraints are found.

g) **Licensing requirements**

MEL is using a paid Theme called Metronic, license details are reported below:

LICENSE CERTIFICATE : Envato Market Item

=====

This document certifies the purchase of:
ONE REGULAR LICENSE
as defined in the standard terms and conditions on Envato Market.

Licensor's Author Username: keenthemes
Licensee: ICARDA

For the item:
Metronic - Responsive Admin Dashboard Template

<http://themeforest.net/item/metronic-responsive-admin-dashboard-template/4021469>
Item ID: 4021469

Item Purchase Code: a93d6c45-9530-4ffb-8ca6-2637e280583a

Purchase Date: 2014-06-18 07:26:04 UTC

For any queries related to this document or license please contact Help Team via <https://help.market.envato.com>

Envato Pty. Ltd. (ABN 11 119 159 741)
PO Box 16122, Collins Street West, VIC 8007, Australia

==== THIS IS NOT A TAX RECEIPT OR INVOICE ====

Also, MEL is using Zend Framework, an open source framework for developing web applications and services using PHP, and is under the following BSD license:

Copyright (c) 2005-2016, Zend Technologies USA, Inc. All rights reserved.

- Redistribution and use in source and binary forms, with or without modification.
- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.



- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of Zend Technologies USA, Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

h) Data repository and distribution requirements

No limitations, nor constraints are found.

i) Security requirements (or other such regulations)

No limitations, nor constraints are found.

j) Memory or other capacity limitations

No limitations, nor constraints are found.

k) Performance requirements

No limitations, nor constraints are found.

l) Network communications

No limitations, nor constraints are found.

m) Verification and validation requirements (testing)

No limitations, nor constraints are found.

n) Other means of addressing quality goals

MEL is trying to follow Zend Framework condign standards, to help ensure that the code is high quality, has fewer bugs, and can be easily maintained. The following has a brief explanation of what coding standards Zend Framework requires:

• **PHP File Formatting**

- Indentation

Indentation should consist of four spaces. Tabs are not allowed.

- Maximum Line Length

The target line length is 80 characters. That is to say, Zend Framework developers should strive keep each line of their code under 80 characters where possible and practical. However, longer lines are acceptable in some circumstances. The maximum length of any line of PHP code is 120 characters.

- Line Termination

Line termination follows the UNIX text file convention. Lines must end with a single linefeed (LF) character. Linefeed characters are represented as ordinal 10, or hexadecimal 0x0A.

• **Naming Conventions**

- Classes

Zend Framework standardizes on a class naming convention whereby the names of the classes directly map to the directories in which they are stored. The root level directory of Zend Framework's standard library is the "Zend/" directory, whereas the root level directory of Zend Framework's extras library is the "ZendX/" directory. All Zend Framework classes are stored hierarchically under these root directories. Class names may only contain alphanumeric characters. Numbers are permitted in class names but are discouraged in most cases. Underscores are only permitted in place of the path separator; the filename "Zend/Db/Table.php" must map to the class name "Zend_Db_Table".

- Abstract Classes

In general, abstract classes follow the same conventions as classes, with one additional rule: abstract class names must end in the term, "Abstract", and that term must not be preceded by an underscore. As an example, Zend_Controller_Plugin_Abstract is considered an invalid name, but Zend_Controller_PluginAbstract or Zend_Controller_Plugin_PluginAbstract would be valid names.

- Interfaces

In general, interfaces follow the same conventions as classes, with one additional rule: interface names may optionally end in the term, "Interface", but that term must not be preceded by an underscore. As an example, Zend_Controller_Plugin_Interface is considered an invalid name, but Zend_Controller_PluginInterface or Zend_Controller_Plugin_PluginInterface would be valid names.

- Filenames

For all other files, only alphanumeric characters, underscores, and the dash character ("-") are permitted. Spaces are strictly prohibited.

- Functions and Methods

Function names may only contain alphanumeric characters. Underscores are not permitted. Numbers are permitted in function names but are discouraged in most cases. Function names must always start with a lowercase letter. When a function name consists of more than one word, the first letter of each new word must be capitalized. This is commonly called "camelCase" formatting.

- Variables

Variable names may only contain alphanumeric characters. Underscores are not permitted. Numbers are permitted in variable names but are discouraged in most cases. For instance, variables that are declared with the "private" or "protected" modifier, the first character of the variable name must be a single underscore. This is the only acceptable application of an underscore in a variable name. Member variables declared "public" should never start with an underscore.

- Constants

Constants may contain both alphanumeric characters and underscores. Numbers are permitted in constant names.

All letters used in a constant name must be capitalized, while all words in a constant name must be separated by underscore characters. For example, EMBED_SUPPRESS_EMBED_EXCEPTION is permitted but EMBED_SUPPRESSEMBEDEXCEPTION is not. For more information about Zend Framework coding style, have a look at:

<http://framework.zend.com/manual/1.12/en/coding-standard.coding-style.html>

o) Other requirements described in the Requirements Document

2.3.3 Risks

Instructions: Describe any risks associated with the system design and proposed mitigation strategies.



- **MVC pattern is not on the same level**

MEL is applying the MVC in a different way than Zend suggests, The M (Model/) exists directly inside the application/ folder while the VC exist inside the default module/.

iMMAP believes that this approach has no impact on the system flow, but it has an impact on the MVC design pattern of Zend as it doesn't completely follow it. A suggested movement for the shared models (User Model) to the default module, with being cautious on not moving the shared models and keeping a track on the classes' names -If changed-.

- **Missing Data Mapper pattern component**

MEL is merging the Data Source Model with the Data Mapper Model.

iMMAP believes that this has an impact on the number of database hits, and the execution time of mapper models before hitting the database (It might be negligible for now, but worths mentioning).

- **JSON Parsing**

Parsing JSON on MySQL version 5.1 has an impact on the performance, MEL solved this by creating a JSON mapper that takes JSON data and convert to object.



3. Design Considerations

Instructions: Describe issues, which were addressed or resolved when designing the system.

No issues were solved nor addressed, the biggest risk on MEL was the frequently changing requirements (Fields, relations, features) since the CGIAR Research Program (CRP) Planning and Reporting requirements have been modified during the period 2013-2016.

As mentioned by Jalal Omary (MEL former developer), design decisions were not based on the best practices because of the highly changeable requirements.

An example of a decision made based on the frequent requested changes is:

- “We used JSON in some tables because they were changed many times; we used one field for JSON inside those tables. Performance issue from this arises because of JSON parsing on MySQL Version 5.1 which doesn't support JSON, this was solved by creating a JSON mapper that takes JSON data and convert it to object”. Jalal Eddin Omary (MEL System Developer).

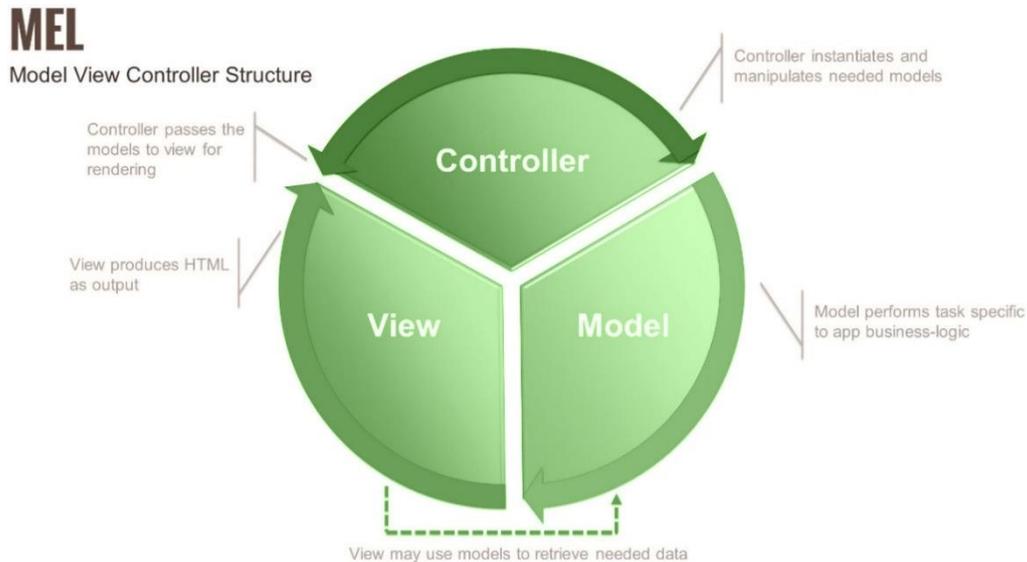
3.1 Goals and Guidelines

Instructions: Describe any goals, guidelines, principles, or priorities, which dominate or embody the design of the system and its software. Examples of such goals might be an emphasis on speed versus memory use; or working, looking, or “feeling” like an existing product. Guidelines include coding guidelines and conventions. For each such goal or guideline, describe the reason for its desirability unless it is implicitly obvious.

MEL system design guidelines focuses on maintainability, and extensibility. The MVC architectural pattern was the choice to help improve these guidelines and reusability, splitting the application into three distinct components with certain responsibilities.

The MVC pattern breaks a project into three manageable modules to satisfy the separate functions in a development process and achieve the loosely coupled design. The result is that the presentation code can be consolidated in one part with the business logic in another and data access code in another module. This well-defined separation indispensable for keeping code organized, especially when more than one developer is working on the same application. A breakdown of the MVC pattern into individual pieces is shown in (Fig. 8).

Figure 8. Zend MVC Design Pattern Structure



- **Model**

Contains domain-specific instructions. This is the part of the application that defines its basic functionality behind a set of abstractions. Data access routines and some business logic can be defined in the model.

- **View**

Views define user interface of the controller, and what is exactly presented to the user. Usually controllers pass data to each view to render in some format. Views will often collect data from the user, as well. This is where HTML mark-up can be found in MVC applications.

- **Controller**

Controllers bind the whole pattern together, containing code to handle all the allowed user events and coordinates the view and models. They manipulate models, decide which view to display based on the user's request and other factors, pass along the data that each view will need, or hand off control to another controller entirely. Most MVC experts recommend keeping controllers as skinny as possible.

The "use-at-will" design assures that each framework component is designed with few dependencies on other components, and improves the loosely coupled architecture, which allows developers to use components individually.

With the Object-oriented Goodness and its tested code, Zend Framework 1 (ZF1) was the framework of the choice.

In addition to MVC, one of the main MEL goals was to create a user-friendly interface; Metronic has a clean and intuitive metro style design. Metronic comes with a huge collection of plugins and UI components that enables developer to extend, and enhance feel and look of the system.

A good performance was also one of the MEL System goals. For this, MEL is using Ajax, DataTables, and MVC to provide performance enhancement guidelines.



3.2 Development Methods & Contingencies

Instructions: Briefly describe the method or approach used for the system and software design (e.g., structured, object-oriented, prototyping, J2EE, UML, XML, etc.). If one or more formal/published methods were adopted or adapted, then include a reference to a more detailed description of these methods. Describe any contingencies that might arise in the design of the system and software that may change the development direction. Possibilities include lack of interface agreements with outside agencies or unstable architectures at the time. Address any possible workarounds or alternative plans.

The MEL system is developed on an Object Oriented Approach based on the Zend MVC framework. Interface agreements with the Research program on Roots, Tubers, and Bananas (RTB) and the International Potato Center (CIP) have required forking the current development path to include their requirements.

ICARDA team has installed and applied some changes on the MEL system to adhere RTB technical requirements, changes include:

- Overview section sidebar addition to include more information about: Enhanced genetic resources, Productive varieties and quality seed, resilient crops, Nutritious food and added value, improving livelihoods at scale, and CIP Strategy and Corporate Plan (SCP).
- Sections inactivating: Survey and Open Facts sections are not included in the RTB system.
- Sub-sections changes: RTB planning section has no Activities, and its Pre-planning section has no IDO, nor ALS. Action sites are referred to as Clusters in RTB.

3.3 Architectural Strategies

Instructions: Describe any design decisions and/or strategies that affect the overall organization of the system and its higher-level structures. These strategies should provide insight into the key abstractions and mechanisms used in the system architecture. Describe the reasoning employed for each decision and/or strategy (possibly referring to stated design goals and principles) and how any design goals or priorities were balanced or traded-off.

Examples of design decisions might concern (but are not limited to) things like the following:

- a) Use of a particular type of product (programming language, database, library, commercial off-the-shelf (COTS) product, etc.)
- b) Reuse of existing software components to implement various parts/features of the system
- c) Future plans for extending or enhancing the software
- d) User interface paradigms (or system input and output models)
- e) Hardware and/or software interface paradigms
- f) Error detection and recovery
- g) Memory management policies
- h) External databases and/or data storage management and persistence
- i) Distributed data or control over a network
- j) Generalized approaches to control
- k) Concurrency and synchronization
- l) Communication mechanisms
- m) Management of other resources

a) Use of a particular type of product (programming language, database, library)

As MEL is built on top of Zend Framework 1, the MVC architectural pattern should be used to follow the framework guidelines. PHP is the main programming language for Zend Framework,

other used libraries are: jQuery, Bootstrap, SimpleHtmlDom, PHPEXcel, ImageLib, Excanvas, Respond, DataTables, FancyBox. Please refer to Metronic documentation for the full frontend libraries. MEL is using MySQL, the free, lightweight, and open-source relational database management system (RDBMS).

b) Reuse of existing software components to implement various parts/features of the system
Metronic, a multi-purpose Bootstrap HTML5 global system theme for MEL and is used for its user-friendly interface, responsive, and extensibility.

c) Future plans for extending or enhancing the software

No design decisions and/nor strategies taken affecting the overall organization of the system.

d) User interface paradigms (or system input and output models)

No design decisions and/nor strategies taken affecting the overall organization of the system.

e) Hardware and/or software interface paradigms

No design decisions and/nor strategies taken affecting the overall organization of the system.

f) Error detection and recovery

No design decisions and/nor strategies taken affecting the overall organization of the system.

g) Memory management policies

MEL has no applied memory management policy, the requested memory requested by the developers were provided from ICARDA's side.

h) External databases and/or data storage management and persistence

No design decisions and/nor strategies taken affecting the overall organization of the system.

i) Distributed data or control over a network

No design decisions and/nor strategies taken affecting the overall organization of the system.

j) Generalized approaches to control

No design decisions and/nor strategies taken affecting the overall organization of the system.

k) Concurrency and synchronization

No design decisions and/nor strategies taken affecting the overall organization of the system.

l) Communication mechanisms

No design decisions and/nor strategies taken affecting the overall organization of the system.

m) Management of other resources

No design decisions and/nor strategies taken affecting the overall organization of the system.

4. System Architecture

Instructions: Describe the system architecture, how the application interacts with other applications. Not necessarily how the application itself works but, how the appropriate data is correctly passed between applications. Provide an overview of how the functionality and responsibilities of the system were partitioned and then assigned to subsystems or components. Don't go into too much detail about the individual components themselves in this section. A subsequent section of the System Design Document (SDD) will provide the detailed component descriptions. The main purpose here is to gain a general understanding of how and why the system was decomposed, and how the individual parts work together to provide the desired functionality. At the top-most level, describe the major responsibilities that the software must undertake and the various roles that the system (or portion of the system) must play. Describe how the system was broken down into its components/subsystems (identifying each top-level component/subsystem and the roles/responsibilities assigned to it). Describe how the higher-level components collaborate with each other in order to achieve the required results. Provide some sort of rationale for choosing this particular decomposition of the system. Make use of design patterns whenever possible, either in describing parts of the architecture (in pattern format), or for referring to elements of the architecture that employ them. Provide rationale for choosing a particular algorithm or programming idiom (or design pattern) to implement portions of the system's functionality.

Figure 9. MEL Application Context Mapping

MEL

Application Context Mapping

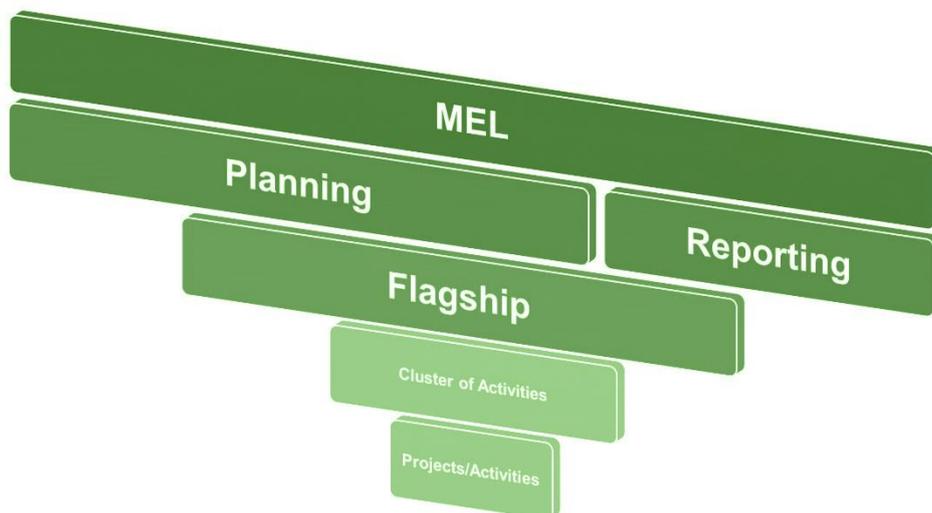


Table 4. Application Context Mapping

Component	Description	Interface Name	Interface System
FP	Aggregation of CoA with an oversight leader	Flagship Project	Flagship Project
CoA	Aggregation of Projects/Activities with an oversight leader	Cluster of Activities	Cluster of Activities
Project/Activities	Planning and reporting environment for a project manager	Project Activity	Project Activity
ALS	Cross-cutting aggregation domain	ALS	ALS

The high-level application design identifies the major components of the system and the relationships of the major application components to each other and the surrounding applications. The major components of the application are at the subsystem or top-level service area. Core architecture tenets include:

- Utilization of Commercial Off-the-Shelf (COTS) for visualization and development framework: MEL is built on top of the COTS capabilities of PHP on the Zend Framework, Metronic and JQuery. These functions will be maximized as COTS functions to ensure maximum benefit from the defined architecture. This aligns with the design principles of ICARDA enterprise initiatives.
- Loose coupling across components: for all COTS and custom components, the principles of loose coupling have been utilized to maximize the possibility of further enhancements. To meet this design direction, MEL maximizes the use of standards-based interactions and limits the use of proprietary data interchange.
- Object-relational mapping (ORM) works by providing the developer with an object-oriented solution for interacting with the database, each database table is mapped to a corresponding class. This class is not only able to communicate with the table, performing tasks such as selecting, inserting, updating, and deleting data, but can also be extended by the developer to include other behaviour, such as data validation and custom queries.
- The Data Mapper design pattern is a layer of software that separates the in-memory objects from the database. Its responsibility is to transfer data between the two and to isolate them from each other. With Data Mapper, the in-memory objects need not know even that there is a database present; they need no SQL interface code, and certainly no knowledge of the database schema.
- The Zend_Auth class implements the Singleton pattern - only one instance of the class is available - through its static getInstance() method.



4.1 Hardware Architecture

Instructions: Describe the overall system hardware and organization, indicating whether the processing system is distributed or centralized. Identify the type, number, and location of all hardware components including the presentation, application, development and data servers and any peripheral devices (e.g., load balancers, SSL accelerator, CDN with a brief description of each item and diagrams showing the connectivity between the components along with required firewalls, ports. Include resource estimates for processor capacity, memory, on-line storage, and auxiliary storage.

MEL system is hosted on a server with the following specification:

- **Hosting provider**
 - Netblock owner: CGNET Services International, Inc. 1170 Hamilton Ct. Menlo Park CA US 94025.
 - IP address: 192.156.137.193.
 - OS: Windows Server 2012.
 - Web server: Apache/2.4.17 Win32 OpenSSL/1.0.2d PHP/5.6.14.
 - Nameserver: ns-752.awsdns-30.net.
- **OpenSSL**
 - A collaborative effort project to develop a robust, commercial-grade, full-featured, and Open Source toolkit implementing the Secure Sockets Layer (SSL v2/v3) and Transport Layer Security (TLS v1) protocols as well as a full-strength general purpose cryptography library.
 - OpenSSL: OpenSSL/1.0.2d.

4.1.1 Performance Hardware Architecture

Instructions: Describe the hardware components and configuration supporting the performance and reliability of the system. Identify single points of failure and, if relevant, describe high availability design (e.g., clustering). The selection process of the cloud host and description from the hosting providers should suffice. The proposed use of a Content Delivery Network (CDN) should be described here too.

MEL has no dedicated hardware to enhance the system performance, nothing of the following were implemented: load balancer, CDN, clusters.

4.2 Software Architecture

Instructions: Describe all software that is needed to support the system, and specify the physical location of all software systems. List such things as logical components (e.g., CSS in presentation layer, Controllers in application layer, MySQL connectors in data layer), database platforms, computer languages, utilities, operating systems, communications software, commercial off-the-shelf (COTS) software, open source frameworks, etc., with a brief description of the function of each item and any identifying information such as manufacturer, version number, number and types of licenses needed, etc., as appropriate. Identify all Computer Software Configuration Items (CSCIs), Computer Software Components (CSCs) and Application Programming Interfaces (APIs) to include name, type, purpose and function for each; the interfaces, messaging, and protocols for those elements; and rationale for the software architectural design. Include software modules that are functions, subroutines, or classes. Use functional hierarchy diagrams, structured organization diagrams (e.g., structure charts), or object-oriented diagrams that show the various segmentation levels down to the lowest level. All features on the diagrams should have reference numbers and names.

Include a narrative that expands on and enhances the understanding of the functional breakdown. If necessary, describe how a component was further divided into subcomponents, and the



relationships and interactions between the subcomponents. Proceed into as many levels/subsections of discussion as needed in order to provide a high-level understanding of the entire system or subsystem, leaving the details for inclusion in a later section of the SDD. Include data flow diagrams that conform to appropriate standards (e.g., Yourdon-Demarco conventions) and provide the physical process and data flow related to the logical process and data flow decomposed to the primitive process level (describing how each input is processed/transformed into the resulting output).

- **Frameworks**

Zend, an open source, object-oriented web application framework implemented in PHP 5 and licensed under the New BSD License. PHP is a widely-used general-purpose scripting language that is especially suited for Web development and can be embedded into HTML.

- **JavaScript Libraries**

- **JQuery 1.11.0**

JQuery is a fast, concise, JavaScript Library that simplifies how you traverse HTML documents, handle events, perform animations, and add Ajax interactions to your web pages.

- **JQuery UI**

JQuery UI provides abstractions for low-level interaction and animation, advanced effects and high-level, themeable widgets, built on top of the jQuery JavaScript Library, that you can use to build highly interactive web applications.

- **JQuery BlockUI**

JQuery BlockUI Plugin lets you simulate synchronous behaviour when using AJAX, without locking the browser.

- **Uniform**

Uniform masks your standard form controls with custom themed controls.

- **JQuery Form**

JQuery Form Plugin allows you to easily and unobtrusively upgrade HTML forms to use AJAX.

- **JQuery Validate**

JQuery Form Validation Plugin.

- **Backstretch**

A simple jQuery plugin that allows you to add a dynamically resized, slideshow-capable background image to any page or element.

- **Widgets**

- **Google Font API**

The Google Font API helps you add web fonts to any web page.

- **Font Awesome**

Iconic font and CSS toolkit.

- **Document Information**

- **HTML5 DocType**

The DOCTYPE is a required preamble for HTML5 websites.

- **Conditional Comments**

The website uses conditional comments that are supported by Microsoft Internet Explorer. They allow web developers to show or hide HTML code based on the version of the viewer's browser.

- **X-UA-Compatible**

Allows a website to define how a page is rendered in Internet Explorer 8, allowing a website to decide to use IE7 style rendering over IE8 rendering.

- **Cascading Style Sheets**

Cascading Style Sheets (CSS) is a stylesheet language used to describe the presentation of a document written in a mark-up language. Its most common application is to style web pages written in HTML.

- **Twitter Bootstrap**

Bootstrap is a toolkit from Twitter designed to kick-start development of webapps and sites.

- **JavaScript**

JavaScript is a scripting language most often used for client-side web development. Its proper name is ECMAScript, though "JavaScript" is much more commonly used. The website uses JavaScript.

- **Encoding**

- **UTF-8**

UTF-8 (8-bit UCS/Unicode Transformation Format) is a variable-length character encoding for Unicode. It is the preferred encoding for web pages.

4.2.1 Performance Software Architecture

Instructions: Describe the software components and configuration supporting the performance and reliability of the system. Identify single points of failure and, if relevant, describe high availability design (e.g., clustering).

MEL system is using AJAX: ajaxForm and ajaxSubmit, to gather information from the form element to determine how to manage the submit process.

In addition, DataTables the table plug-in for jQuery is used to enhance instant search, multi-column ordering, hidden columns, dynamic creation of tables, and Ajax auto loading of data.

MEL Application is using the following two techniques to increase performance inside classes:

- Classes are written in a way to improve data communication between classes by using absolute paths instead of include path and reduce the numbers of include paths example:

```
defined('APPLICATION_PATH')
    || define('APPLICATION_PATH', realpath(dirname(__FILE__) . '/application'));

defined('UPLOAD_PATH')
    || define('UPLOAD_PATH', realpath(dirname(__FILE__) . '/uploads'));

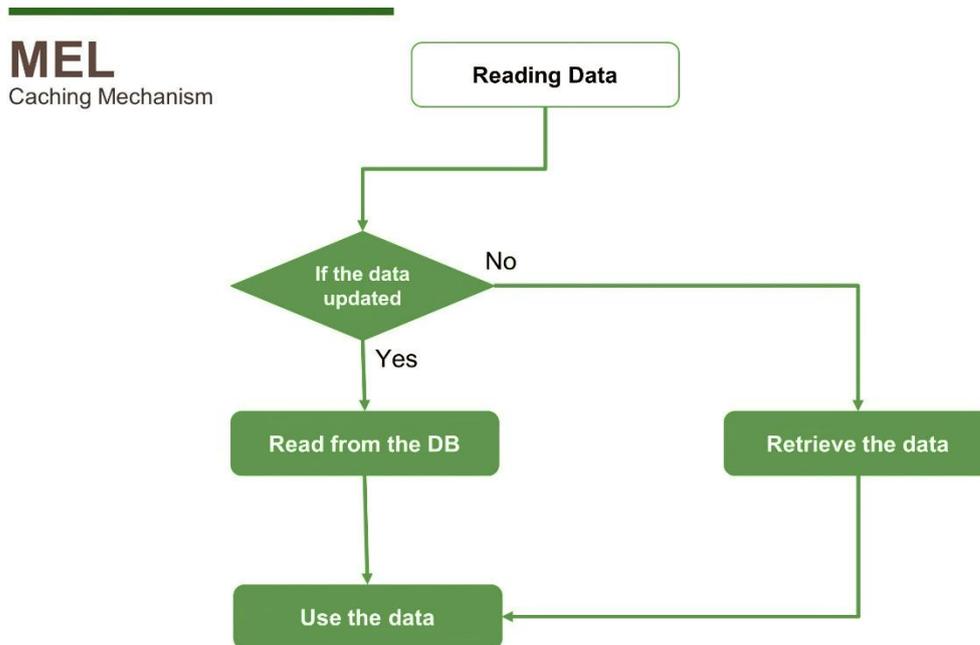
defined('LIBRARY_PATH')
    || define('LIBRARY_PATH', realpath(dirname(__FILE__) . '/library'));
```

- In the following code it shows how to using caching mechanism:

```
protected function cacheActivities()
{
    $_flagshipActivityMapper = new Model_Mapper_FlagshipActivity ();
    $_flagshipActivityCollection = $_flagshipActivityMapper->fetchMany(null, array(
        'parent_activity_id ASC',
        'code ASC'
    ));
    $this->view->flagshipActivityCollection = $_flagshipActivityCollection;
    $_cachedActivities = $this->view->render('planning/flagshipactivities.phtml');
    file_put_contents(APPLICATION_PATH . '/cache/activities.dat', $_cachedActivities);
}
```

The function gets FlagshipActivities from Model_Mapper_FlagshipActivity and file_put_contents render flagshipactivities.phtml page and store the output in cache folder to load the activities from file instead of the database, as shown in the following figure:

Figure 10. File caching inside PlanningController class



4.3 Information Architecture

Instructions: Describe the information that will be stored in the system (e.g. Project information, research data, etc.).

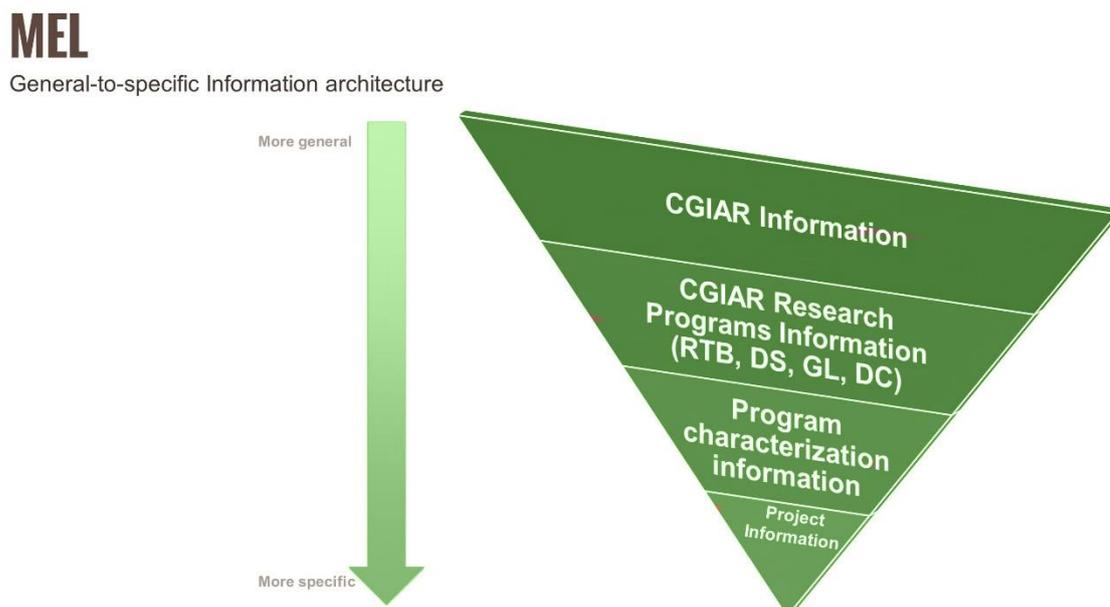
MEL users store multiple types of information:

- CGIAR Information:**
 Strategy and Results Framework (SRF) (System-Level Outcome (SLO), Intermediate Development Outcomes (IDO), Sub-IDO), Financial Details (One Corporate System (OCS) and L-Series format), type of funding (Window 1 / Window 2, Window 3, Bilateral), Output Type, Deliverables Type, Deliverables Metadata Schema (incl. Country List ISO 3166).
- CGIAR Research Programs Information (Roots, Tubers and Banana (RTB), Dryland Systems (DS), Grain Legumes (GL), Development Countries (DC)):** Partners Type, Nested Partner Convention, Indicators, CapDev Elements.
- Program characterization information:** which includes static information such as, Flagship Projects, Cluster of Activities, Indicators.
- Project information:** which includes: Project name, status, reference code to activate web services with other organizational systems (e.g. OCS), Donors, Leadership, Implementation period, CRP mapping, Objectives, Goals, Outputs, deliverables, Outcomes, indicators, relations Output-Outcome (Impact Pathway dynamic generation), Research Phase, Scientists and Partners, Budget (planned, revised and current expenditures disaggregated

in different categories, Capacity Development Intervention (as a special sub-set of deliverables), Project files as part of the reporting process to meet planned deliverables. It is also allowed to report un-planned deliverables, online survey data.

A general to specific view of the stored Information architecture us shown in (Fig. 11).

Figure 11. MEL System Information Architecture General-to-Specific Approach



Additionally, the following three types of information will be added in 2016:

- UN SDG, Indicators and Regions.
- USAID Feed the Future Indicators.
- IFAD Result Framework.

For UN SDG, and USAID Feed the Future Indicators, a detailed reference is handed over in this documentation delivery as: “CG Core Metadata Schema and Application Profile.pdf”, and “Reference for Information Architecture.xlsx”.

4.4 Internal Communications Architecture

Instructions: Provide a diagram depicting the communications flow between the system and subsystem components.

MEL System components interacts with each other in systematic defined ways, definition depends on either Zend Framework logic itself, or a customized way built for a custom feature in the system i.e. Open access repository. The following will cover these internal interactions with code snippets.

- **Open Access Repository**

As MEL is using the DSpace digital repository to grant open access to deliverables and outcomes, internal communication between MEL and this repository located at <http://mel.cgiar.org/repo> is customized by a PHP client library for RESTful web services (PEST).

Unlike Zend_Rest_Client, which is not really a "REST" client at all (more like RPC-over-HTTP), Pest supports the four REST verbs (GET/POST/PUT/DELETE) and pays attention to HTTP response status codes.

Pest's get(), post(), put(), and delete() return the raw response body as a string. **PestJSON** – what MEL is using- is a JSON-centric version of Pest, specifically targeted at REST services that return JSON data. Rather than returning the raw response body as a string, PestJSON will try to parse the service's response into a JSON object. The following getCollection method located at (library/Dspace/Dspace.php) illustrates the idea:

```

public function getCollection($id = 0)
{
    if (!empty($id) && $id > 0) {
        try {
            $response = $this->pestRequest->get('/collections/' . $id);
            return array('status' => 1, 'result' => json_decode($response));
        } catch (Exception $e) {
            return array('status' => 0, 'message' => $e->getMessage(), 'code' => $e->getCode());
        }
    } else {
        return array('status' => 0, 'message' => 'ID not provided', 'code' => -1);
    }
}

```

Although DSpace class located at (library/Dspace/Dspace.php) requires both PestJSON.php, and PestXML.php, but its Constructor method is defaulting the response type to JSON as follows:

```

require_once LIBRARY_PATH . '/Pest/PestJSON.php';
require_once LIBRARY_PATH . '/Pest/PestXML.php';

class Dspace
{
    private $base_url;
    private $domain;
    private $pestRequest = null;
    public $authToken = null;
    public $authEmail = null;
    public $authPassword = null;

    public function __construct($domain, $typeResponse = 'JSON')
    {
        $this->domain = $domain;
        $this->base_url = $domain . "/rest";

        if (!empty($this->base_url)) {
            if ($typeResponse == "JSON") {
                $this->pestRequest = new PestJSON($this->base_url);
            }
        }
    }
}

```

```

    } else {
      $this->pestRequest = new PestXML($this->base_url);
    }
  }
}
}

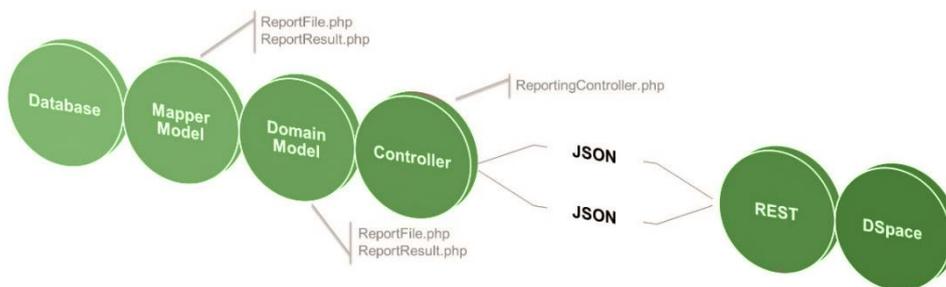
```

The internal communications of the DSpace open repository is shown in (Fig. 12), with its corresponding Controllers, Domain and Mapper Models.

Figure 12. DSpace Internal Communication

MEL

DSpace internal communication



- **Controller to Database**

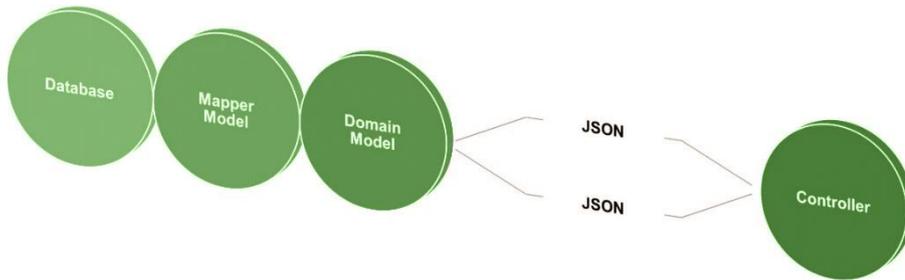
MEL is implementing its Custom Data Access object (DAO) pattern, by the missing Data Source element (More information on this at Section 7: Software Modules/CRUD).

A resultant Database to Controller internal communication can be summarized in (Fig. 13).

Figure 13. Controller to Action Internal Communication

MEL

Controller to Database internal communication



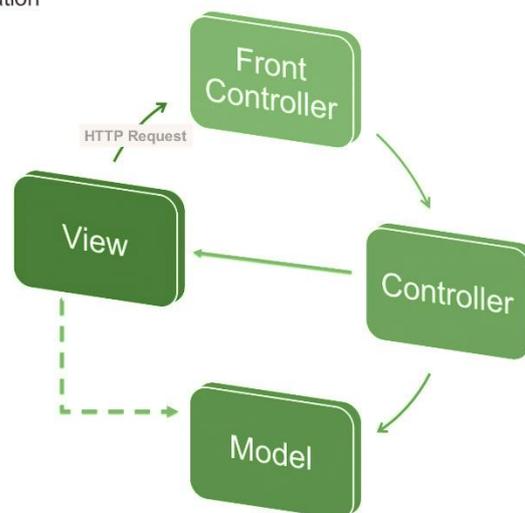
- **Controller to View**

MEL is following Zend Framework logic by passing data from Controller to View via the front controller that enables the ViewRenderer action helper. This helper takes care of injecting the view object into the controller, as well as automatically rendering views. (Fig. 14) Shows this Front Controller pattern.

Figure 14. Controller to View Internal Communication

MEL

Controller to view internal communication



A direct example of this can be found at [\(application/modules/default/controllers/OverviewController.php\)](#) as follows:

```
class OverviewController extends Zend_Controller_Action
{
    $this->view->mainTitle = $_flagshipEntity->name;
    $this->view->subTitle = 'Overview';
}
```

- **JavaScript to Controller**

This is where MEL is initiating URL Controller at (application/views/layouts/default.phtml) as follows:

```
<script>
base_url = "<?=$this->baseUri()?>";
var discussionsHomeLink = '<?=$this-
>url(array('module'=>'default','controller'=>'discussion','action'=>'index'),null,
,true)?>';
var userDataLink = '<?=$this-
>url(array('module'=>'default','controller'=>'user','action'=>'getallusers'),null,
,true)?>';
var discussionCountLink = '<?=$this-
>url(array('module'=>'default','controller'=>'discussion','action'=>'discussions
count'),null,true)?>';
</script>
```

- **View to Controller/Scripts**

Because our JS files are isolated in the assets/ directory, we are defining global variables with the URLs names as values, and are being passed to those JS scripts. Those Ajax calls URLs are Zend URLs so they are compatible with the default route.

An example can give a better understanding of this view to controller direction, JS variables at (application/modules/default/views/scripts/planning/activities.phtml) are defined as follows:

```
<?php $this->placeholder('styleVars')->captureEnd() ?>

<?php $this->placeholder('jsvars')->captureStart();?>
var tableDataLink = '<?=$this-
>url(array('module'=>'default','controller'=>'planning','action'=>'flagshipactiv
ities'),null,true)?>';
var tableDataCountLink = '<?=$this-
>url(array('module'=>'default','controller'=>'planning','action'=>'flagshipactiv
itiescount'),null,true)?>';
var getDataLink = '<?=$this-
>url(array('module'=>'default','controller'=>'planning','action'=>'getflagshipac
tivity'),null,true)?>';
var deleteDataLink = '<?=$this-
>url(array('module'=>'default','controller'=>'planning','action'=>'delflagshipac
tivity'),null,true)?>';
```

```
var alsDataLink = '<?=$this-  
>url(array('module'=>'default','controller'=>'preplanning','action'=>'getallals'),  
null,true)?>';  
var actionsiteDataLink = '<?=$this-  
>url(array('module'=>'default','controller'=>'preplanning','action'=>'getallacti  
onsites'),null,true)?>';  
var flagshipDataLink = '<?=$this-  
>url(array('module'=>'default','controller'=>'preplanning','action'=>'getallflag  
ships'),null,true)?>';  
var actionsiteDataLink = '<?=$this-  
>url(array('module'=>'default','controller'=>'preplanning','action'=>'getallacti  
onsites'),null,true)?>';  
var userDataLink = '<?=$this-  
>url(array('module'=>'default','controller'=>'user','action'=>'getallusers'),null,  
true)?>';  
<?php $this->placeholder('jsvars')->captureEnd() ?>
```

- **View to View**

According to the previous snippet of code; MEL views talk to each other via the Placeholders. MEL views do not write inside the same view, they write inside a place holder, or write a new placeholder. This placeholder has place inside the layout.

4.5 System Architecture Diagram

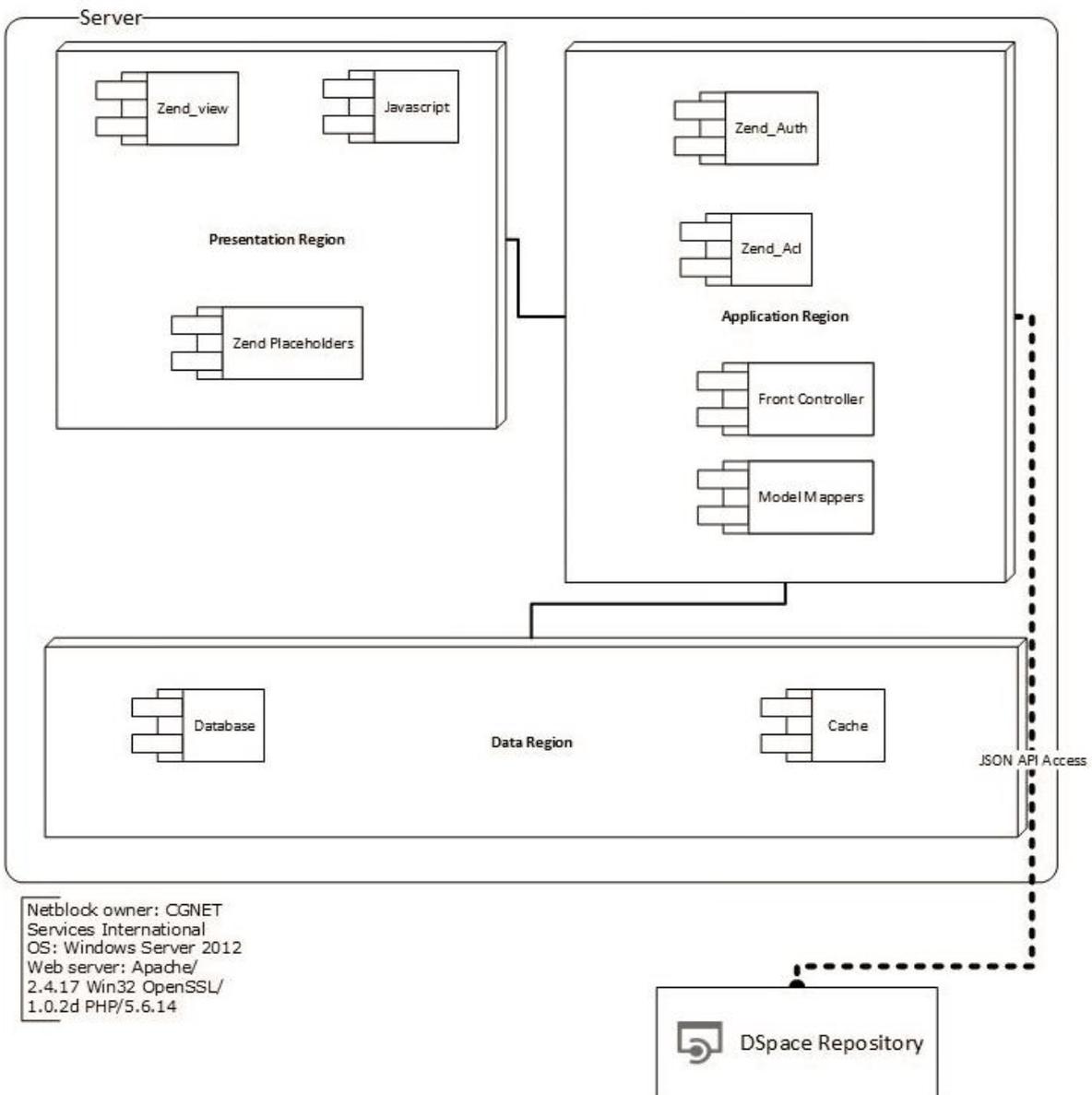
Instructions: Using the hardware, software, communications, and information designs described above, depict the overall, integrated structure of the system in terms of presentation, application, and data regions.

Diagram(s) must reflect the complete system’s context, i.e., more detailed software components, internal/external interfaces, and their underlying infrastructure, etc. Include a table of Objects that are in the diagram.

Figure 15. System Architecture

MEL

System Architecture





Monitoring, Evaluation and Learning Platform: System Design & Architecture

This diagram illustrates system architecture and its internal components and their relations. The system inside hardware container (CGNET Server) and it includes 3 main nodes each node represents the functionality of the specified node components. Presentation region consists of system components that are responsible for displaying data from application stage as following:

- Zend_View component transfer data between the view script files and the Front Controller.
- Zend_Placeholder component pass data between views.
- Javascript component initiate controller URLs to transfer data from and to the controller.

In the application region, it contains the components that are responsible for controlling the data and pass them between data region, outside DSpace repository and view region. The data region contains the components responsible for preserving the data either in database or cache.

5. System Design

5.1 Business Requirements

Instructions: Describe each top level business process including actors and process flows. Insert any related project business requirements or provide a reference to where they are stored.

The following tables (Table. 5 – Table. 11) represent MEL System business processes with a detailed description on Actors, Process Flow, and Requirements.

Table 5. Business Process 0: Project Definition

Actors	<ul style="list-style-type: none"> i. Admin User. ii. Project Manager (User). iii. Financial Administrator (User-Focal Point). iv. Guest.
Process Flow	<ul style="list-style-type: none"> i. Admin User: Collects basic project information and enters on the MEL Interface to create a project. ii. Project Manager (User): Add detail information and enters on the MEL interface to complete the project. iii. Financial Administrator (User-Focal Point): Add budget details for the project. iv. Guest: View information about project.
Requirements	<ul style="list-style-type: none"> i. Detailed project information. ii. Project Budget.

Table 6. Business Process 1: Project Planning and Monitoring

Actors	<ul style="list-style-type: none"> i. Project Manager (User). ii. Reviewers (User-Center Focal Point; CapDev Focal Point; Gender Focal Point)). iii. Guest.
Process Flow	<ul style="list-style-type: none"> i. Project Manager (User): Add Management Information in terms of Inputs/Outputs and enters on the MEL interface to complete the project. ii. Reviewers (User-Center Focal Point; CapDev Focal Point; Gender Focal Point)): <ul style="list-style-type: none"> a. Approve the entered inputs, outputs to confirm project start. b. Monitor deliverables and burn rate in dedicated dashboard. iii. Financial Administrator (User-Focal Point): Add budget details for the project. iv. Guest: View planned results of the projects and check resources involved.
Requirements	<ul style="list-style-type: none"> i. Project management information (Resources, inputs, and outputs).

Table 7. Business Process 2: Financial Accounting

Actors	<ul style="list-style-type: none"> i. Financial Administrator (User-Focal Point). ii. Project Manager (User). iii. Reviewers (User-Center Focal Point; CapDev Focal Point; Gender Focal Point). iv. System. v. Guest.
Process Flow	<ul style="list-style-type: none"> i. Financial Administrator (User-Focal Point): Add Financial information at project start and quarterly for project expenditures. ii. Project Manager (User): Add required resources and modify along project implementation requesting approval for modifications. iii. Reviewers (User-Center Focal Point; CapDev Focal Point; Gender Focal Point): Approve plan of work and budget. iv. System: <ul style="list-style-type: none"> a. Calculate is planned resources are covered by available budget. b. Notify any shortage or un-planned budget and request explanation. c. Display budget by different levels. v. Guest: View information from Flagship, Cluster, Project and Open Facts Section.
Requirements	<ul style="list-style-type: none"> i. Project financial information. ii. Project time-line.

Table 8. Business Process 3: Human Resources Allocation

Actors	<ul style="list-style-type: none"> i. Project Manager (User). ii. Reviewers (User-Center Focal Point). iii. Guests and Users selected in the Projects.
Process Flow	<ul style="list-style-type: none"> i. Project Manager (User): Add required resources and modify along project implementation requesting approval for modifications. ii. Reviewers (User-Center Focal Point): Approve required resources. iii. Guests and Users selected in the Projects: <ul style="list-style-type: none"> a. Staff (user) involved receive notification. b. Dashboard displays allocation of time across projects available to Guests and Users.
Requirements	<ul style="list-style-type: none"> i. Project allocated resources (people).

Table 9. Business Process 4: Surveys

Actors	<ul style="list-style-type: none"> i. Guests and Users (incl. Project Manager and Focal Points). ii. Admin User.
Process Flow	<ul style="list-style-type: none"> i. Guests and Users (incl. Project Manager and Focal Points): User request Admin to activate Survey Dashboard providing Survey title, scope and intended audience. ii. Admin User: <ul style="list-style-type: none"> a. Admin Approve Survey. b. Create title and activate Launch Survey Module and Report Survey Module.
Requirements	<ul style="list-style-type: none"> i. Survey access approval. ii. Survey scope, type, and audience.

Table 10. Business Process 5: Impact Pathway

Actors	<ul style="list-style-type: none"> i. Project Manager (User). ii. System.
Process Flow	<ul style="list-style-type: none"> i. Project Manager (User): Project Manager specifies Impact Pathway (IP) elements and activate links. ii. System: System displays interactive, collapsible, impact pathway.
Requirements	<ul style="list-style-type: none"> iii. Impact Pathway (IP) elements.

Table 11. Business Process 6: Knowledge Sharing

Actors	<ul style="list-style-type: none"> i. Guests and Users. ii. Reviewers (User-Center Focal Point; CapDev Focal Point; Gender Focal Point).
Process Flow	<ul style="list-style-type: none"> i. Guests and Users: Guests and Users can initiate a multi-stakeholder discussion with any Contact in the system. ii. Reviewers (User-Center Focal Point; CapDev Focal Point; Gender Focal Point): Reviewers can initiate a discussion during the review/approval process.
Requirements	<ul style="list-style-type: none"> iv. Once technical/Scientific assistance is needed.

5.2 Database Design

Instructions: Describe the design of all database management system (DBMS) files and non-DBMS files associated with the system. Provide a comprehensive data dictionary showing data element name, type, length, source, validation rules, maintenance (create, read, update, delete (CRUD) capability), data stores, outputs, aliases, and description. The Data Design information can be included as an appendix or recorded in a separate Database Design Document (DDD), as appropriate, which would be referenced here.

At MEL system the used data storage for the application is MySQL as RDBMS, the best very fast, very reliable and very feature-rich open-source RDBMS.

A physical data model (PDM) is provided in html format file name (Full Physical Report.html) which helps you to analyze the tables, views, and other objects in a database, including multidimensional objects necessary for data warehousing. A PDM is more concrete than a conceptual (CDM) or logical (LDM) data model. You can model, reverse-engineer, and generate for all the most popular DBMSs.

- A physical data diagram provides a graphical view of your database structure, and helps to analyze its tables (including their columns, indexes, and triggers), views, and procedures, and the references between them.
- A multidimensional data diagram provides a graphical view of the DataMart or data warehouse database, and helps you identify its facts, cubes and dimensions.

Here are some parts of the Table tbl_activity modeling:

Figure 16. Sample Extracted Database Model



Table tbl_activity

Card of table tbl_activity

Name	tbl_activity
Code	tbl_activity
DBMS	MySQL 5.0

Check constraint name of the table tbl_activity

TK_tbl_activity

Server validation rule of the table tbl_activity

%RULES%

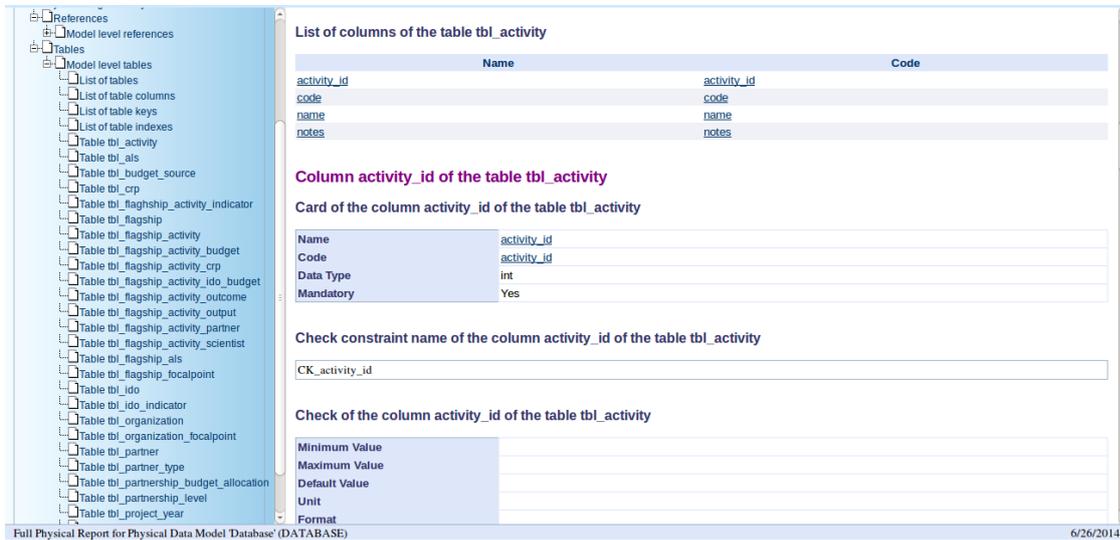
Code preview of the table tbl_activity

```
drop table if exists tbl_activity;

/*=====*/
/* Table: tbl_activity */
/*=====*/
create table tbl_activity
(
  activity_id int not null auto_increment,
  code varchar(5) not null,
  name varchar(255) not null,
  notes text,
  primary key (activity_id)
);
```

Full Physical Report for Physical Data Model 'Database' (DATABASE) 6/26/2014

Figure 17. Columns in tbl_activity



List of columns of the table tbl_activity

Name	Code
activity_id	activity_id
code	code
name	name
notes	notes

Column activity_id of the table tbl_activity

Card of the column activity_id of the table tbl_activity

Name	activity_id
Code	activity_id
Data Type	int
Mandatory	Yes

Check constraint name of the column activity_id of the table tbl_activity

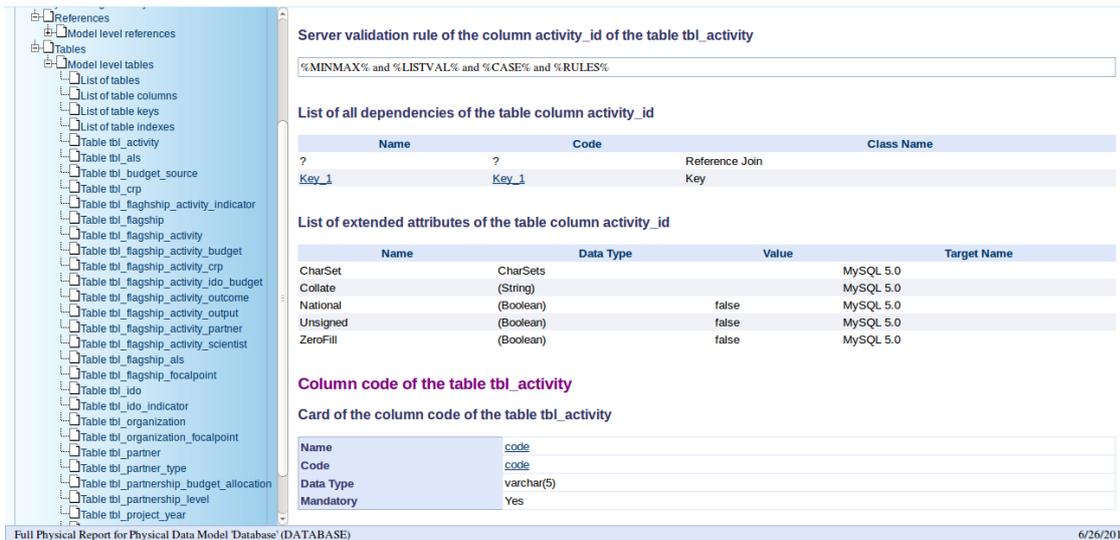
CK_activity_id

Check of the column activity_id of the table tbl_activity

Minimum Value	
Maximum Value	
Default Value	
Unit	
Format	

Full Physical Report for Physical Data Model 'Database' (DATABASE) 6/26/2014

Figure 18. Server validation rule for activity_id field in tbl_activity



Server validation rule of the column activity_id of the table tbl_activity

%MINMAX% and %LISTVAL% and %CASE% and %RULES%

List of all dependencies of the table column activity_id

Name	Code	Class Name
?	?	Reference Join
Key_1	Key_1	Key

List of extended attributes of the table column activity_id

Name	Data Type	Value	Target Name
CharSet	CharSets		MySQL 5.0
Collate	(String)		MySQL 5.0
National	(Boolean)	false	MySQL 5.0
Unsigned	(Boolean)	false	MySQL 5.0
ZeroFill	(Boolean)	false	MySQL 5.0

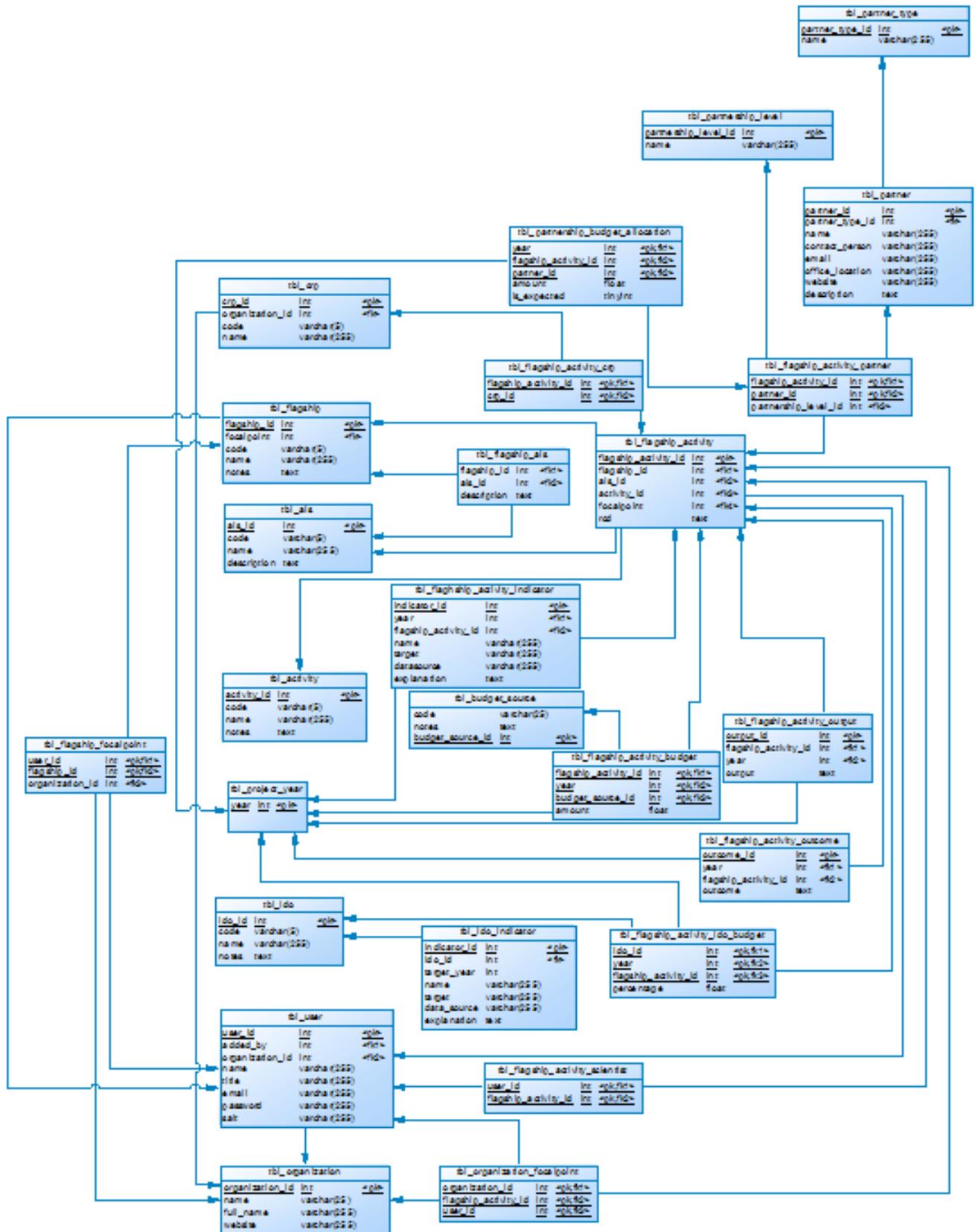
Column code of the table tbl_activity

Card of the column code of the table tbl_activity

Name	code
Code	code
Data Type	varchar(5)
Mandatory	Yes

Full Physical Report for Physical Data Model 'Database' (DATABASE) 6/26/2014

Figure 19. Model level diagram for MainDB





A full Physical Data Model diagram is available here: [Full Physical Report.html](#).

5.2.1 Data Objects and Resultant Data Structures

Instructions: For each functional data object, specify the data structure(s) which will be used to store and process the data. Describe any data structures that are a major part of the system, including major data structures that are passed between components. List all functions and function parameters. For functions, give function input and output names in the description. Refer as appropriate to the decomposition diagrams.

- **Database Naming Conventions**

In the database, there are mainly tables. Delimiter-separated words are used for naming all the objects. Every table name contains the module name to which the table belongs as a prefix followed by underscore and then table name.

Table 12. Database Naming Convention

Database Object	Naming Convention	Example
Table	Prefix_tablename	tbl_partner
Table	Prefix_tablename	Tbl_partner_type
View	Prefix_viewname	View_users
View	Prefix_viewname	View_activity_budget

- **Code Naming Conventions**

The following table shows the naming conventions for different objects used in the code for the platform.

Table 13. Code Naming Convention

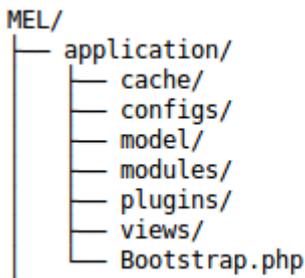
Type/Control	Naming Convention	Example
Namespace	There is one app namespace "Application"	
Classes	Pascal Casing	DataAnalysisController
Method	Camel Casing	indexAction
Dropdown list	Delimiter-separated words	partner_type_id
Text box	Delimiter-separated words	
Label	Delimiter-separated words	
List box	Delimiter-separated words	
Button	Delimiter-separated words	

5.3 File and Database Structures

Instructions: Using the Logical Data Model (LDM), create a physical data model that describes data storage and manipulation in the systems architectural setting. Describe file structures and their locations. Explain how data may be structured in the selected DBMS, if applicable. Refer to a separate DDD, as appropriate.

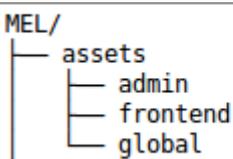
The following describes the use cases for each directory:

Figure 20. Directory Structure (Application)



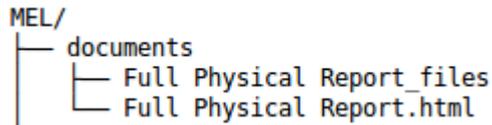
- **application/:** This directory contains your application. It will house the MVC system, as well as configurations, services used, and your bootstrap file.
- **configs/:** The application-wide configuration directory.
- **model/, and views/:** These directories serve as the default model or view directories. Having these two directories inside the application directory provides the best layout for starting a simple project as well as starting a modular project that has global models/views.
- **views/helpers/:** These directories will contain view helpers.
- **views/layouts/:** This layout directory is for MVC-based layouts. Since Zend_Layout is capable of MVC- and non-MVC-based layouts, the location of this directory reflects that layouts are not on a 1-to-1 relationship with controllers and are independent of templates within views/.
- **plugins/:** This directory provides a place to store user code to be called when certain events occur in the controller process lifetime, these event methods are defined in the abstract class Zend_Controller_Plugin_Abstract.
- **modules/:** Modules allow a developer to group a set of related controllers into a logically organized group. The structure under the modules directory would resemble the structure under the application directory.
- **cache/:** This directory provides a place to store application cache files that is volatile and possibly temporary.
- **Bootstrap.php:** This file is the entry point for your application, and should implement Zend_Application_Bootstrap_Bootstrapper. The purpose for this file is to bootstrap the application and make components available to the application by initializing them.

Figure 21. Directory Structure (Assets)



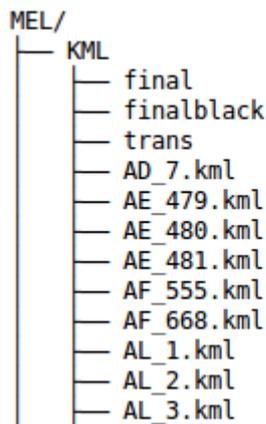
- **assets/:** This directory provides a place to store Metronic admin, frontend, and global files.
- **admin/pages/scripts/:** This directory stores MEL custom admin JS files.

Figure 22. Directory Structure (Documents)



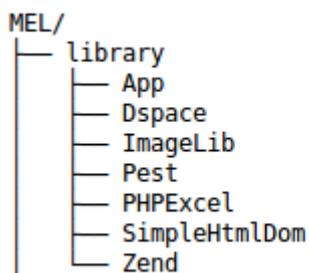
- **documents/**: This directory contains documentation, either generated or directly authored.

Figure 23. Directory Structure (KML)



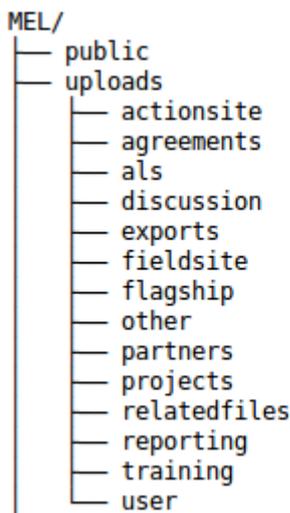
- **KML/**: This directory provides a place to store an XML based file format used to display geographic data in an Earth browser such as Google Earth, Google Maps, and Google Maps for mobile. With KML, you can display pretty much everything on a map.

Figure 24. Directory Structure (Library)



- **library/**: This directory is for common libraries on which the application depends, and should be on the PHP include_path.

Figure 25. Directory Structure (Uploads)



- **public/:** This directory contains all public files for your application. index.php sets up and invokes Zend_Application, which in turn invokes the application/Bootstrap.php file, resulting in dispatching the front controller. The web root of your web server would typically be set to this directory. MEL has this directory **empty**, we are placing index.php file and .htaccess in the root directory with some compatibility changes made on the .htaccess file.
- **uploads/:** This directory provides a place to store files uploaded from discussions, reports, and related uploads.

Figure 26. Directory Structure (Root)



- **.htaccess:** At the root of your web server, you should have an .htaccess file that redirects all non-file requests to your ZF application. Anything that exists as an actual file (like CSS, Javascript, images, etc.) will be served like normal, but all others will be routed to your ZF application.
- **index.php:** This file is where it all begins. All of your .htaccess files should point here. It sets up your environment, the include path, and creates the Zend_Application based on your configuration file. The very last actions it takes is bootstrapping the application and running it.

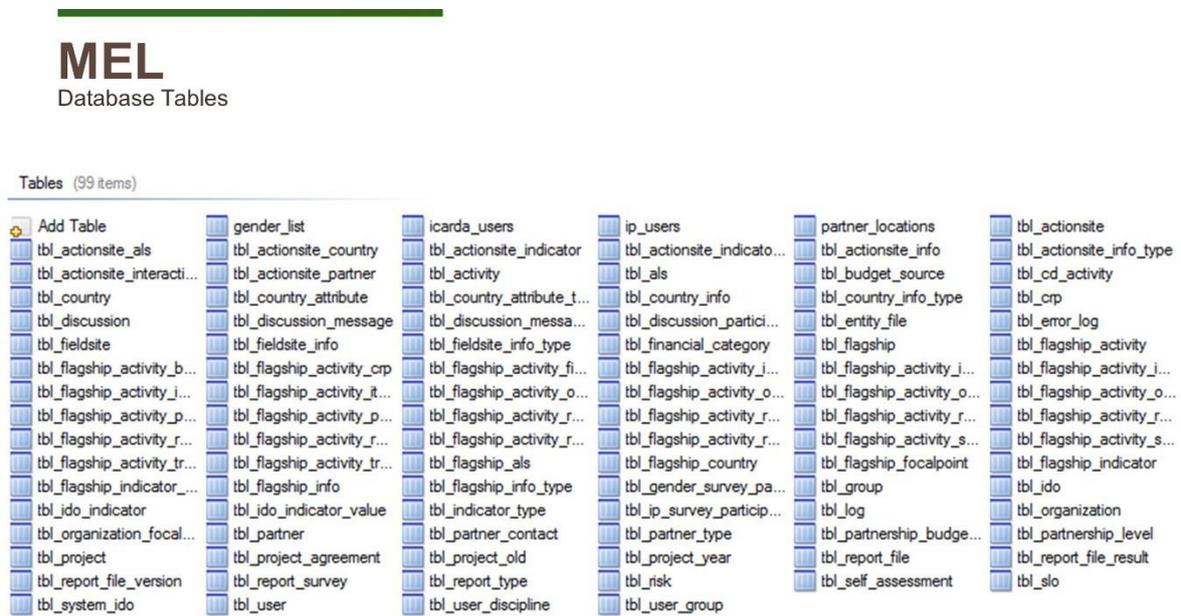
5.3.1 Database Management System Files

Instructions: Provide the detailed design of the DBMS files. Generally, this information should be documented in a separate DDD that should be referenced within this section.

MEL system database consisting of independent and related tables to store and split system data, DDD can be found here.

The system contains 99 tables as follows:

Figure 27. Database Tables



Figures from (Fig. 28 to Fig. 39) shows detailed design for relations between tables:

Figure 28. Action site Tables Relationships

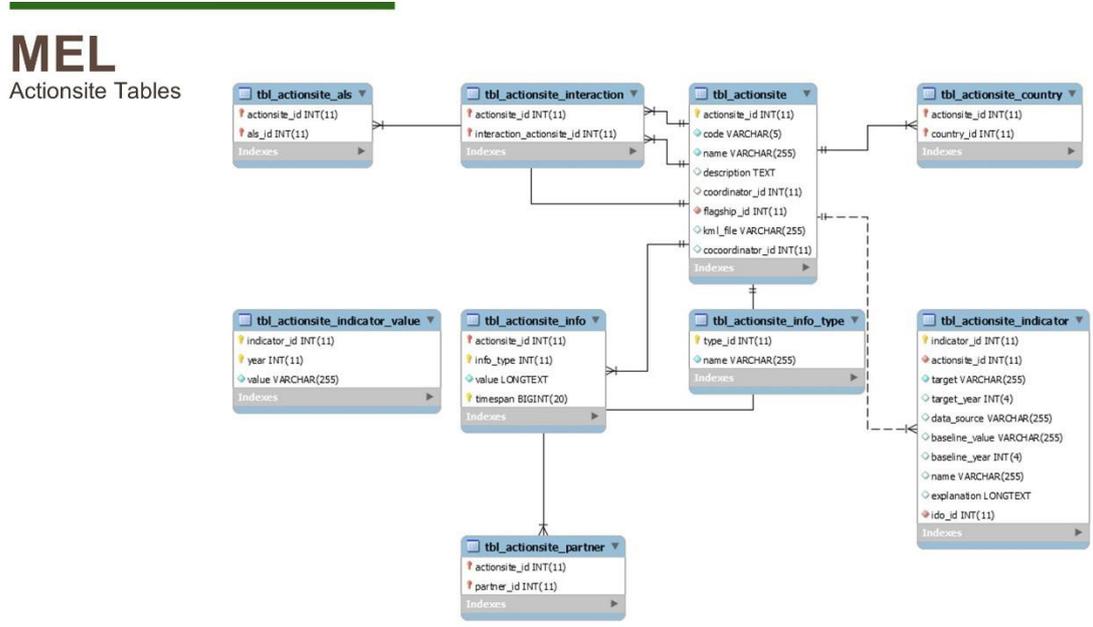


Figure 29. Country Tables Relationships

MEL

Actionsite Tables

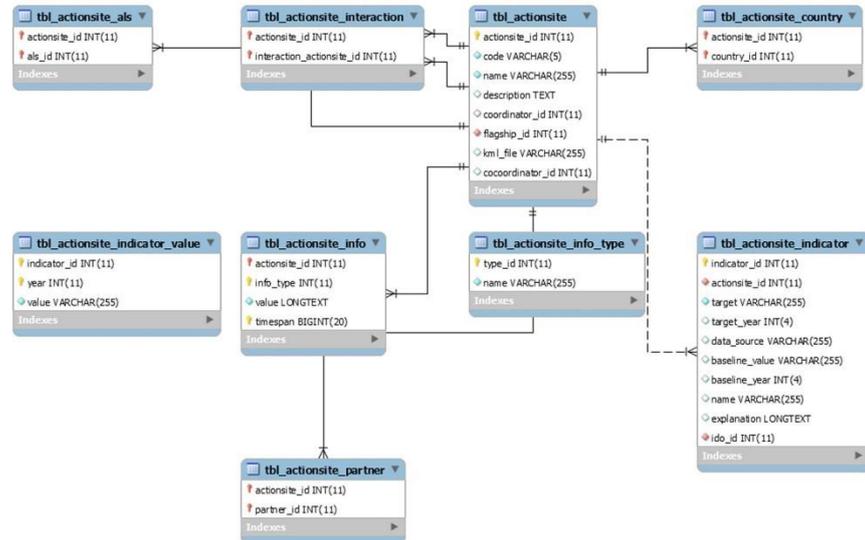


Figure 30. Discussion Tables Relationship

MEL

Discussion Tables

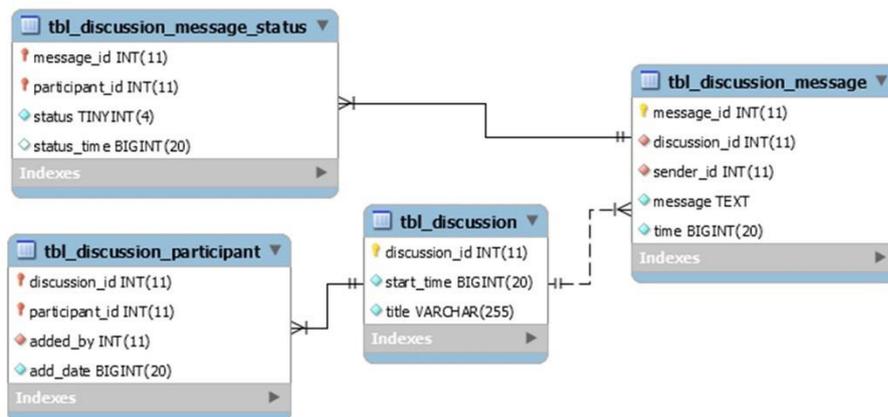


Figure 31. Field site Tables Relationship

MEL
Fieldsite Tables

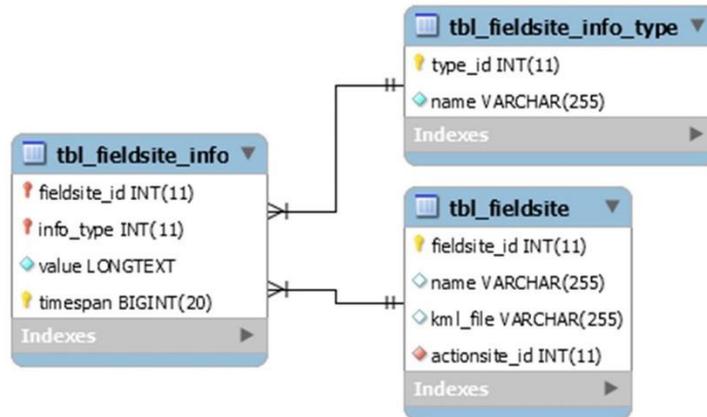


Figure 32. Flagship Tables Relationships

MEL
Flagship Tables

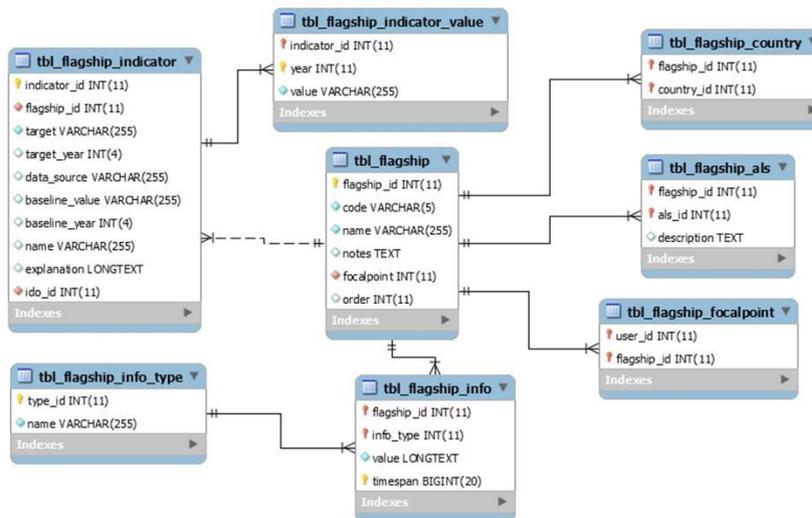


Figure 33. Flagship Activity Tables Relationships

MEL

Flagship Tables

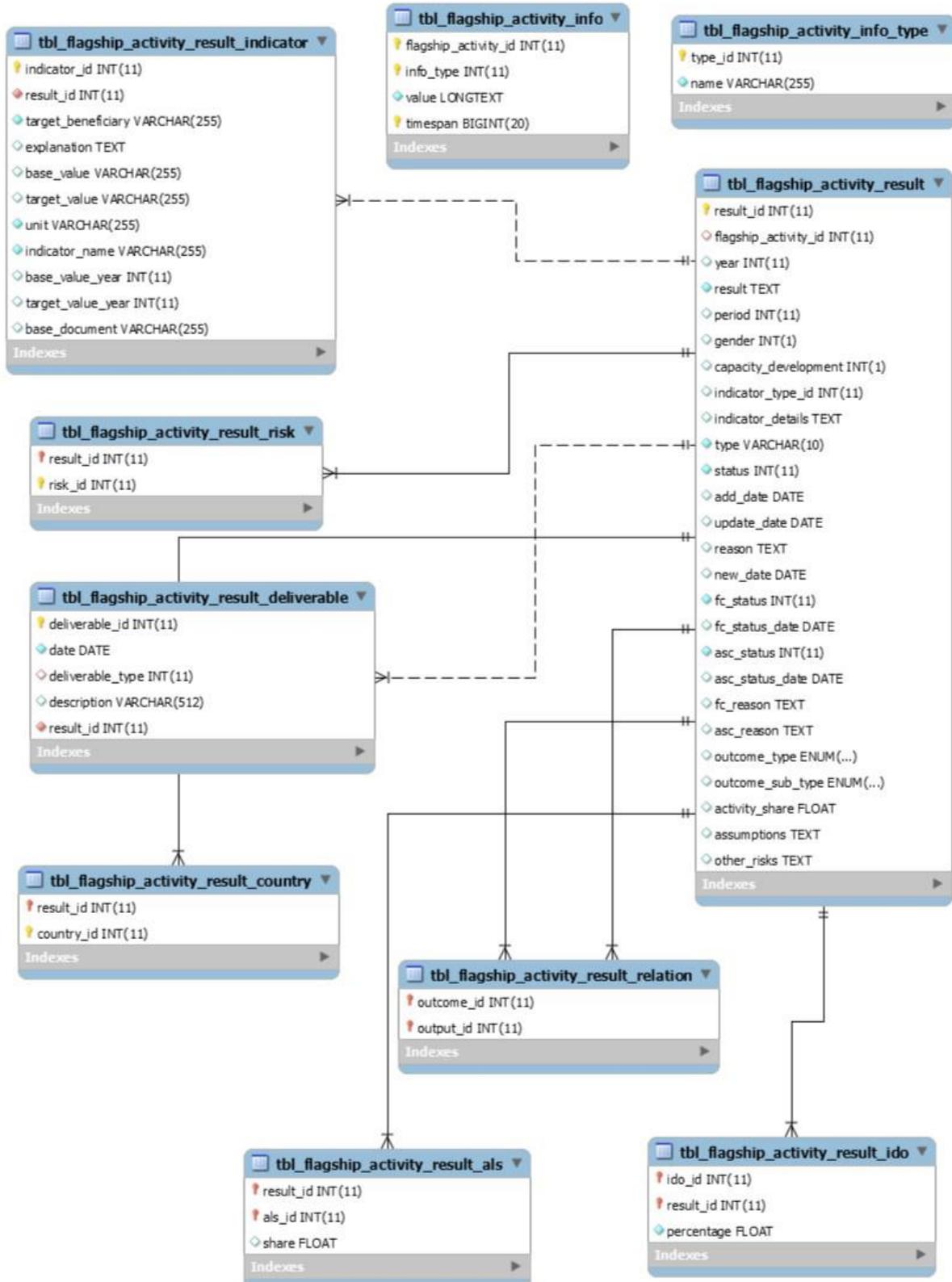




Figure 34. Flagship Activity Details Tables Relationship

MEL

Flagship Tables

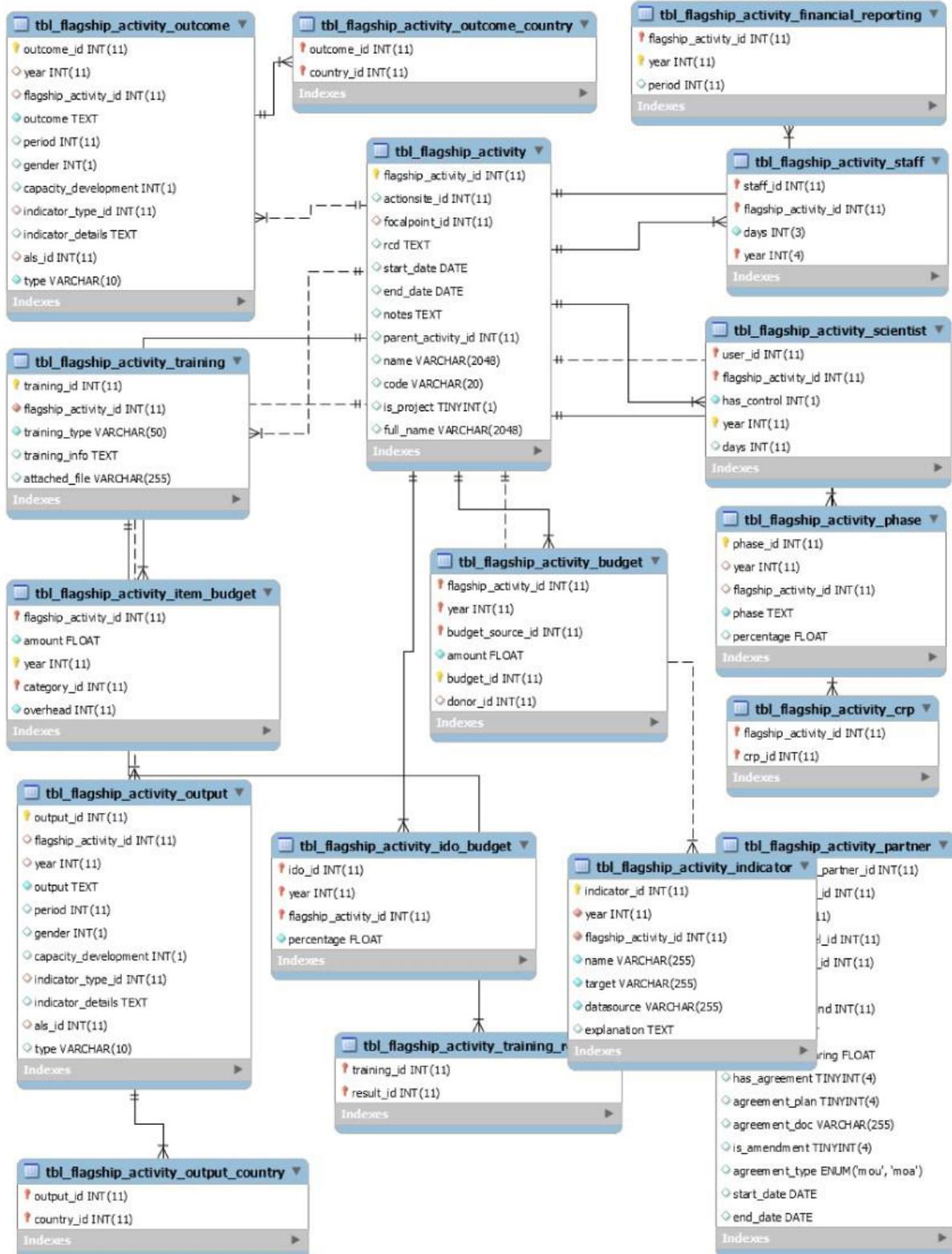


Figure 35. IDO Tables Relationship

MEL
IDO Tables

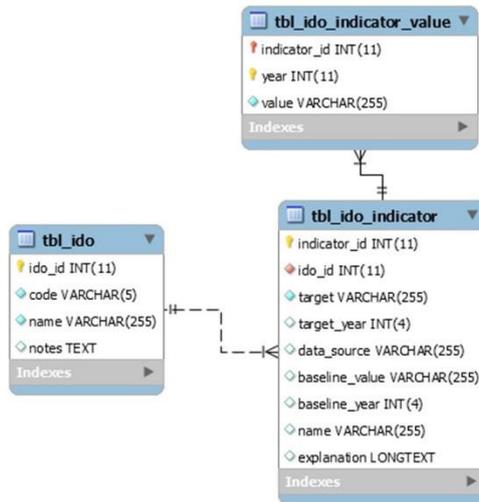


Figure 36. Partner Tables Relationship

MEL
Partner Tables

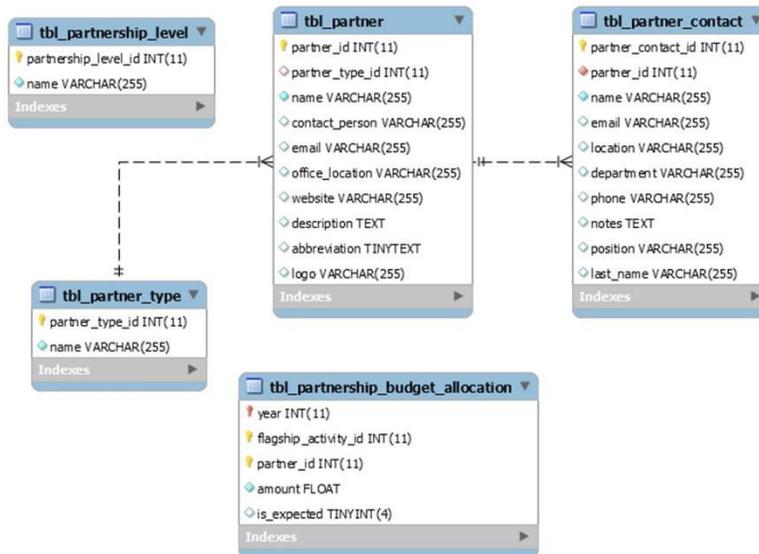


Figure 37. Project Tables Relationship

MEL
Project Tables

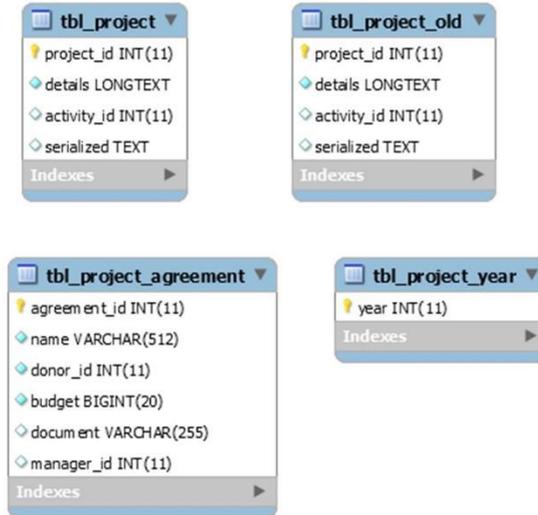


Figure 38. Reports Tables Relationships

MEL
Reports Tables

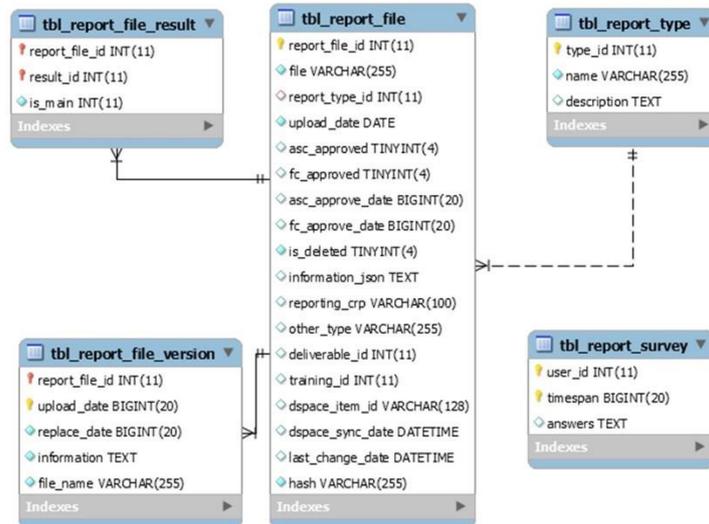
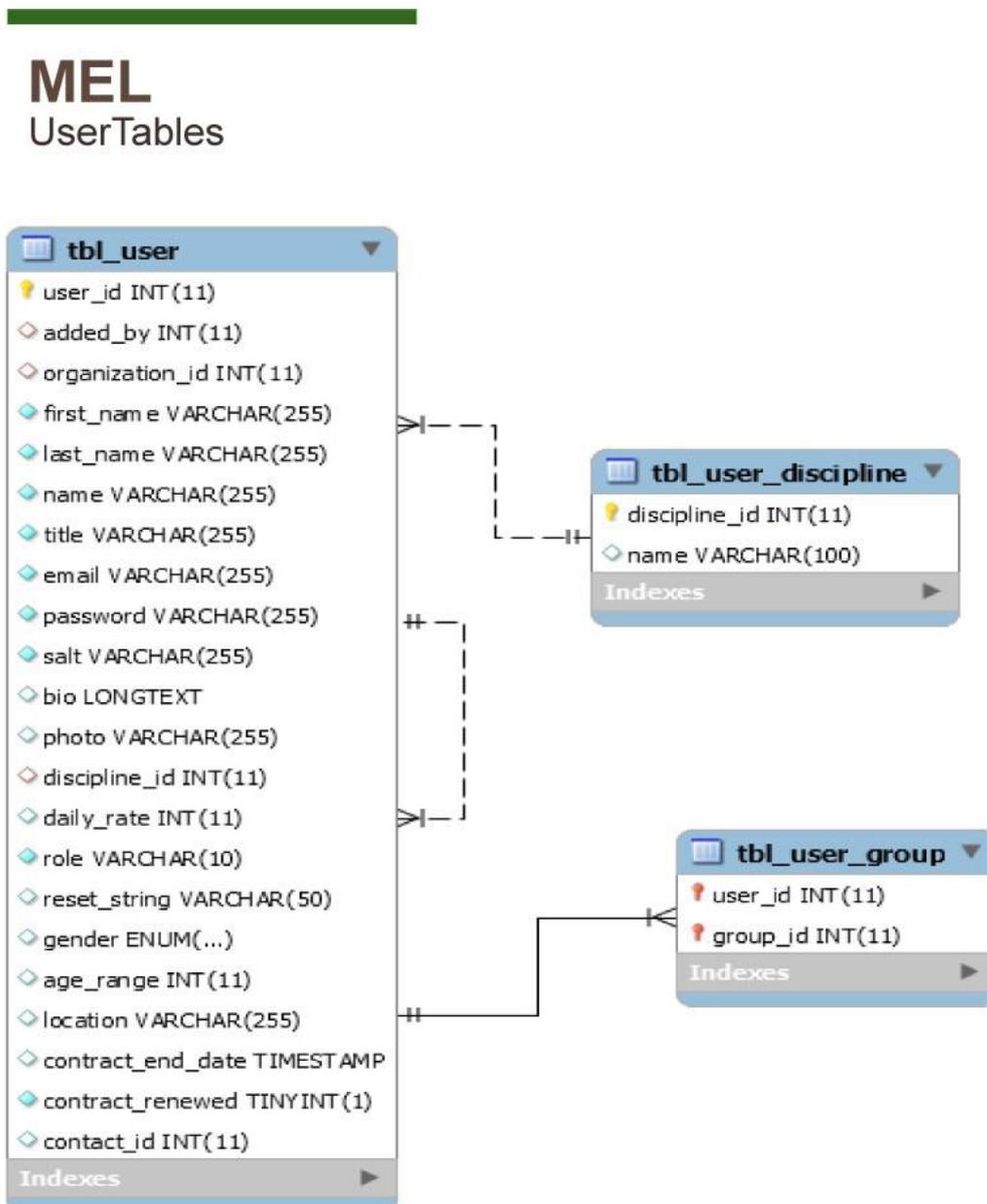


Figure 39. User Tables Relationship



5.3.2 Non-Database Management System Files

Instructions: Provide the detailed description of all non-DBMS files and include a narrative description of the usage of each file that identifies if the file is used for input, output, or both, and if the file is a temporary file. Also provide an indication of which modules read and write the file and include file structures (refer to the data dictionary). As appropriate, the file structure information should include the following:

- Record structures, record keys or indexes, and data elements referenced within the records
- Record length (fixed or maximum variable length) and blocking factors
- Access method (e.g., index sequential, virtual sequential, random access, etc.)



Monitoring, Evaluation and Learning Platform: System Design & Architecture

- d) Estimate of the file size or volume of data within the file, including overhead resulting from file access methods
- e) Definition of the update frequency of the file (If the file is part of an online transaction-based system, provide the estimated number of transactions per unit of time, and the statistical mean, mode, and distribution of those transactions.)
- f) Backup and recovery specifications

f) Backup and recovery specifications

MEL has two backups tasks performed, and are done as follows:

- A server level backup policy (Daily).
- A full local database backup (Per 30 Minutes) via SQLyog on Windows scheduler.

5.4 Database Information

MEL has a single database hosted on CGNET server, with the following specifications:

- Database name: crpdump.
- Database adapter: PDO_MYSQL.
- Database host: localhost.
- Database user name: root.
- Current database size: 85.6 MB.

5.5 Data Conversion

Instructions: Insert any documents describing any necessary data conversions or provide a reference to where they are stored.

Zend_Json provides convenience methods for serializing native PHP to JSON and decoding JSON to native PHP.

JSON, JavaScript Object Notation, can be used for data interchange between JavaScript and other languages. Since JSON can be directly evaluated by JavaScript, it is a more efficient and lightweight format than XML for exchanging data with JavaScript clients.

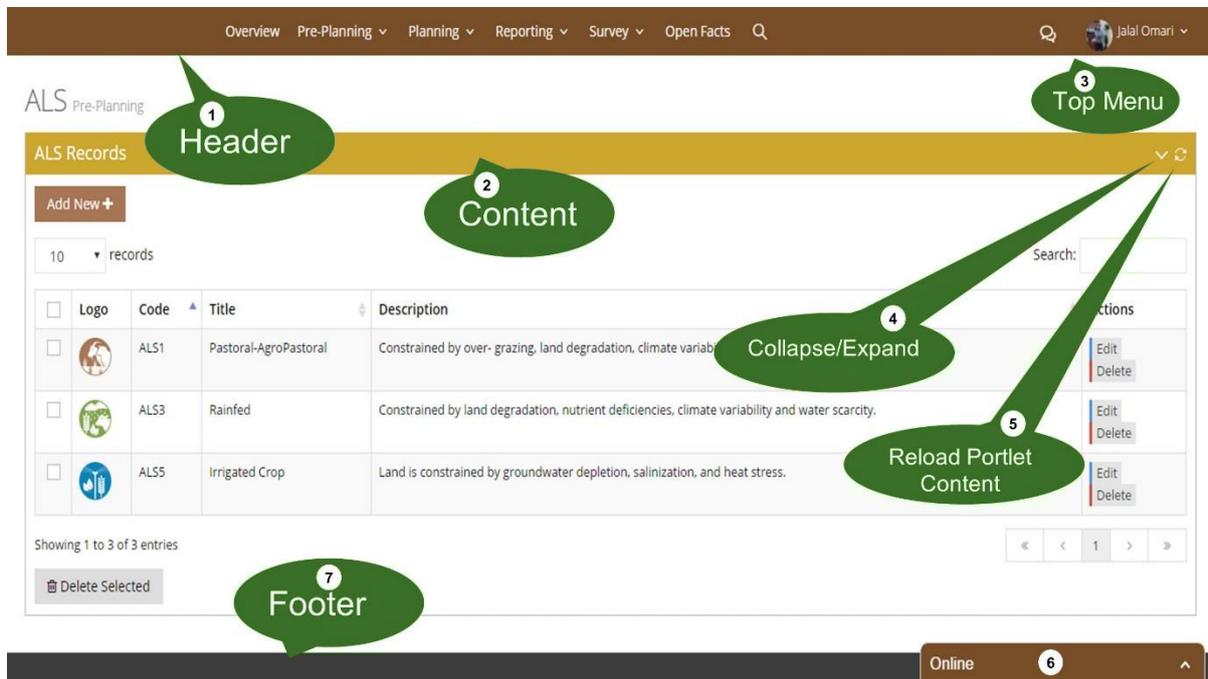
In addition, Zend_Json provides a useful way to convert any arbitrary XML formatted string into a JSON formatted string. This built-in feature will enable PHP developers to transform the enterprise data encoded in XML format into JSON format before sending it to browser-based Ajax client applications. It provides an easy way to do dynamic data conversion on the server-side code thereby avoiding unnecessary XML parsing in the browser-side applications. It offers a nice utility function that results in easier application-specific data processing techniques.

5.6 User Interface Design

Instructions: Insert any user interface design documents or provide a reference to where they are stored.

All template files have fixed structure consisting of header, top menu, content and footer as shown below:

Figure 40. User Interface



1. **Header:** Header contains of logo and top menu bar and it used in all pages.
2. **Content:** Content consists of page title, breadcrumbs and page's main body.
3. **Top Menu:** Top menu enables an easy access to most frequently accessed information and pages.
4. **Collapse/Expand:** metronic component collapse and expand panels.
5. **handlePortletTools:** Initializes & handles portlet tools as described in above image.

A detailed template structure and code snippets can be found in the Metronic documentation.

6. Operational Scenarios

Instructions: Describe the general functionality of the system from the users' perspectives and provide an execution or operational flow of the system via operational scenarios that provide step-by-step descriptions of how the system should operate and interact with its users and its external interfaces under a given set of circumstances. The scenarios tie together all parts of the system, the users, and other entities by describing how they interact, and may also be used to describe what the system should not do.

Operational scenarios should be described for all operational modes, transactions, and all classes of users identified for the proposed system. For each transaction, provide an estimate of the size (use maximum, if variable) and frequency (e.g., average number per session). Identify if there any transactional peak periods and include an estimate of frequency during those periods. Each scenario should include events, actions, stimuli, information, and interactions as appropriate to provide a comprehensive understanding of the operational aspects of the proposed system.

The scenarios can be presented in several different ways: 1) for each major processing function of the proposed system, or 2) thread-based, where each scenario follows one type of transaction type through the proposed system, or 3) following the information flow through the system for each user capability, following the control flows, or focusing on the objects and events in the system. The number of scenarios and level of detail specified will be proportional to the perceived risk and the criticality of the project.

6.1 Operational Modes

MEL System enables user to perform operations based on his/her role (Administrator, User, or Guest) plus actions based on both roles, and project phase. A role based view of operational scenarios is shown in (Fig. 41). Additionally, an action based view of operational scenarios is shown in (Fig. 42).

Figure 41. MEL System Operational Scenarios Sorted Based on Roles

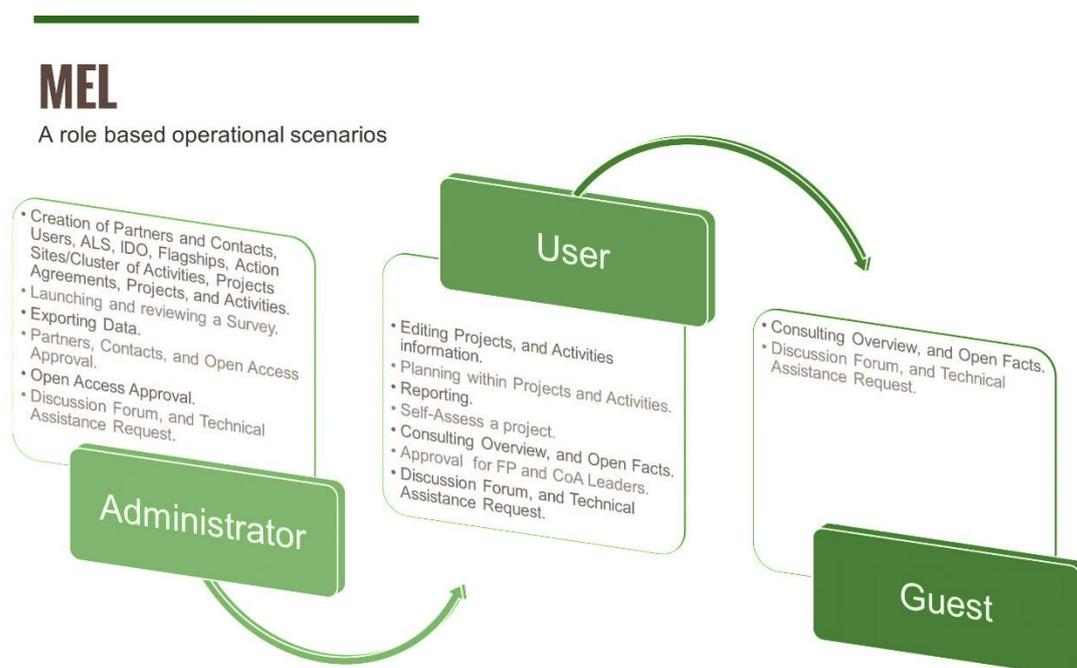
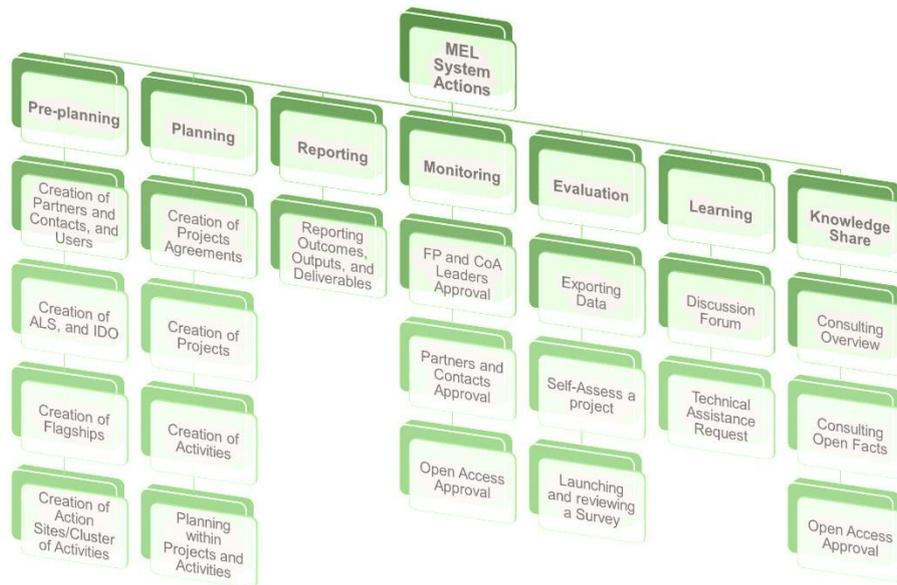


Figure 42. MEL System Operational Scenarios Sorted Based on Actions

MEL An action based operational scenarios



6.2 Operational Scenarios

Once MEL Roles are granted to the different users types, a couple of actions/scenarios can be achieved on the system, (Fig. 43 – Fig. 65) show how to operate on MEL.

Figure 43. Creation of Partners and Contacts Operation

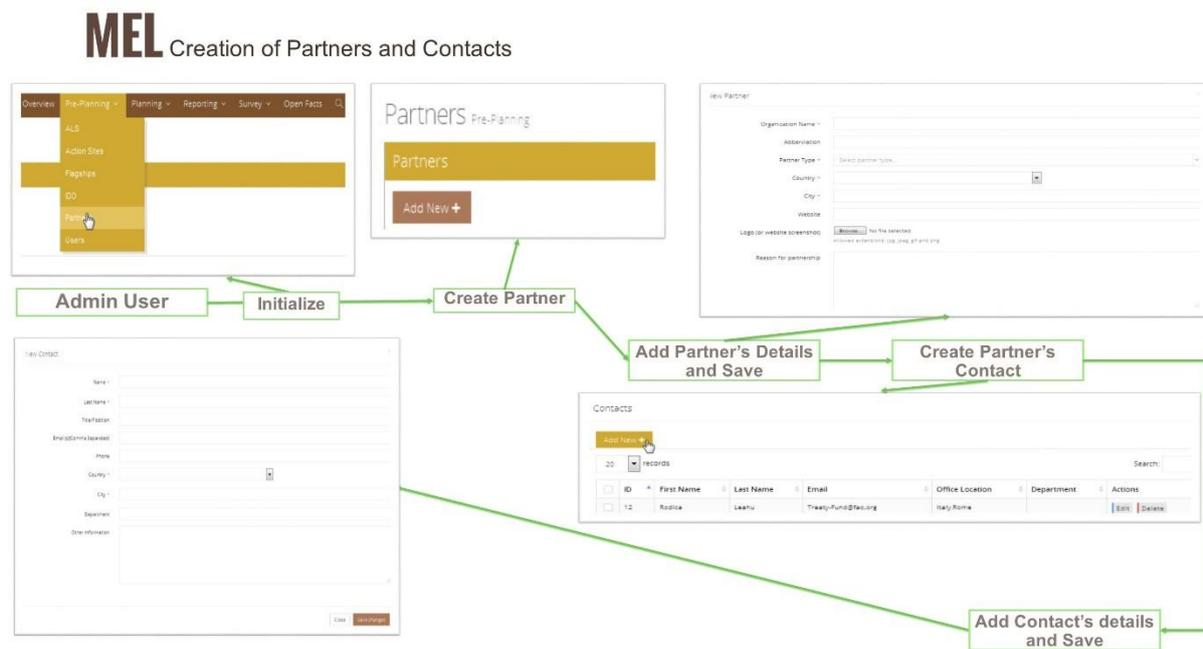


Figure 44. Creation of Users Operation

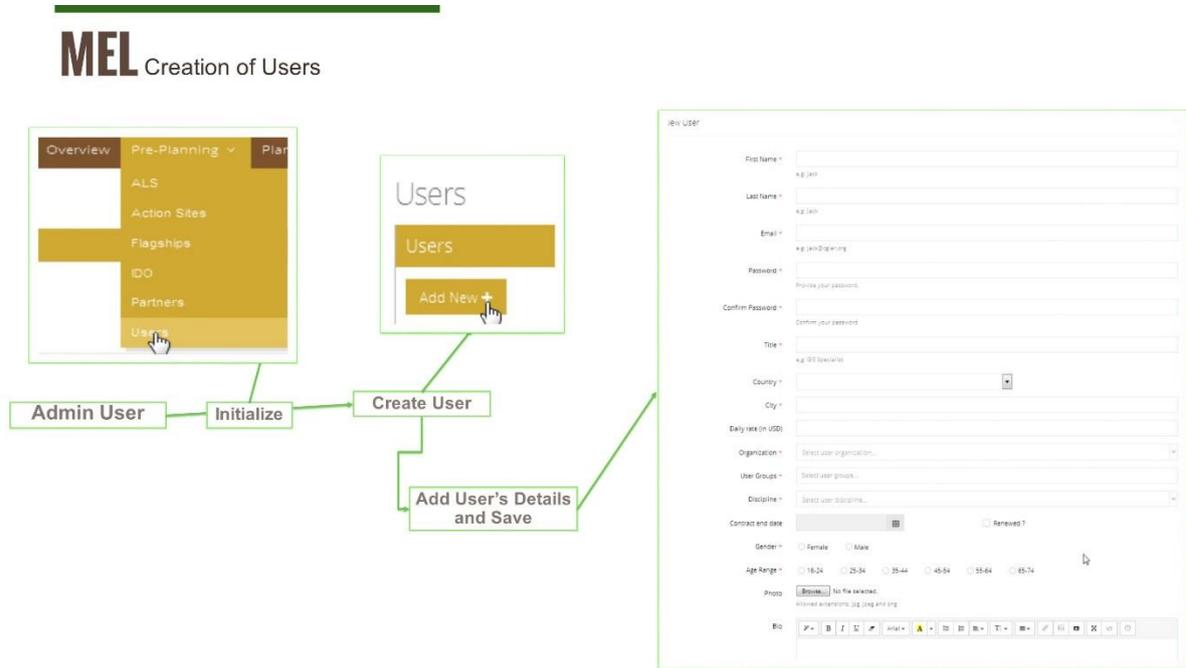


Figure 45. Creation of ALS Operation

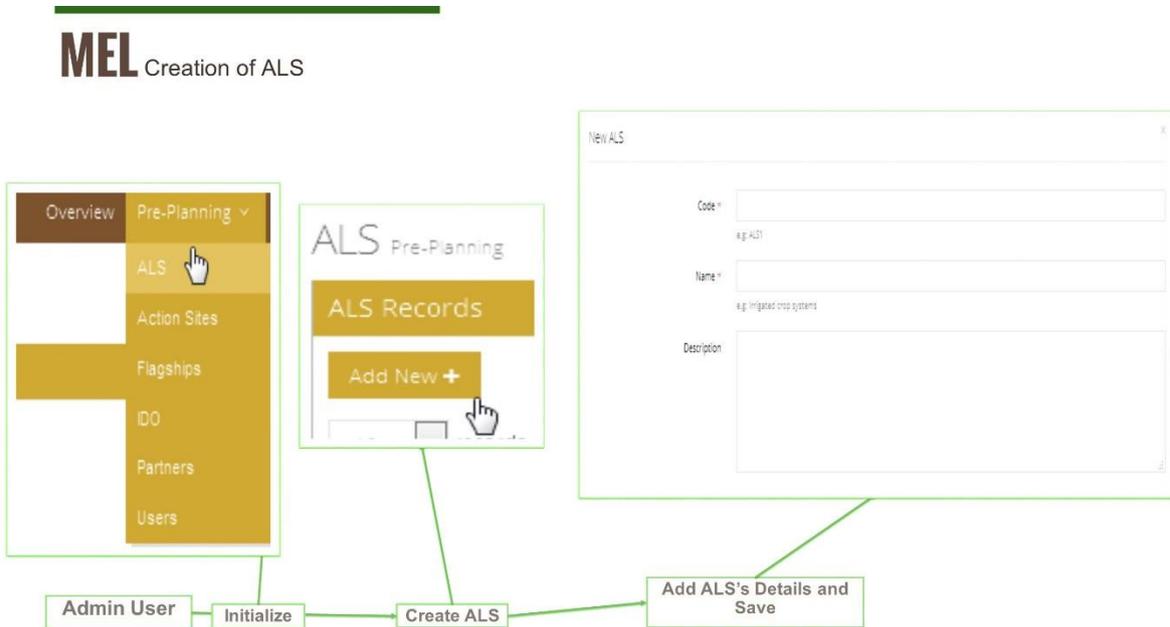


Figure 46. Creation of IDO Operation



Figure 47. Creation of Flagships Operation

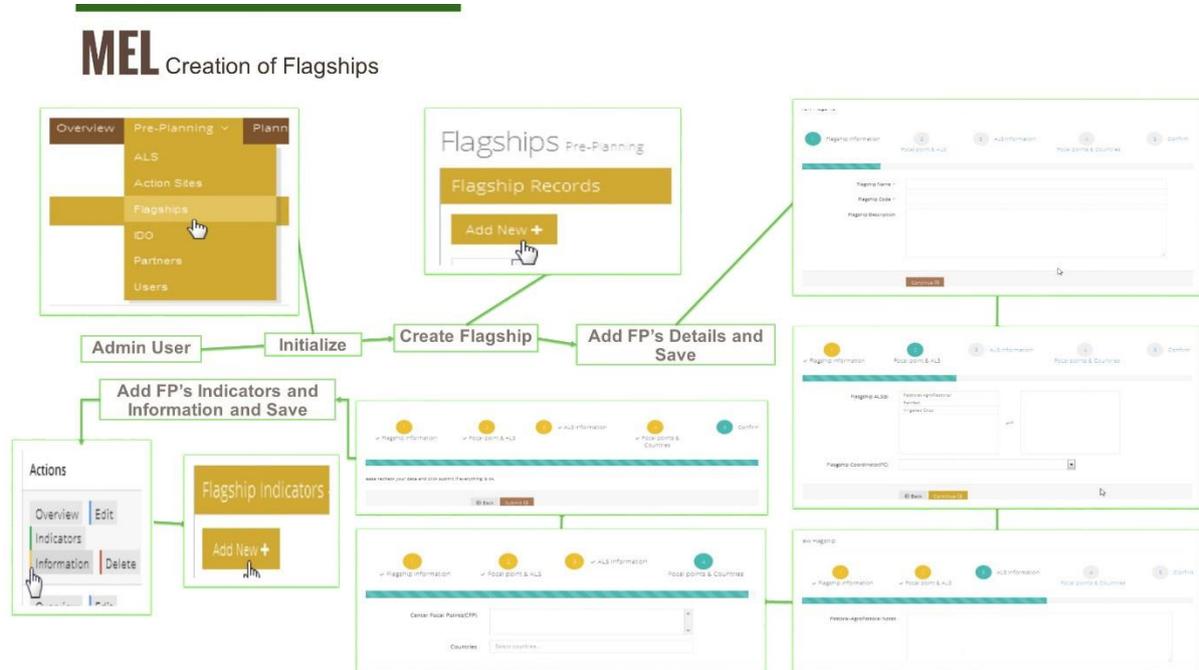


Figure 48. Creation of Action Sites/Cluster of Activities Operation

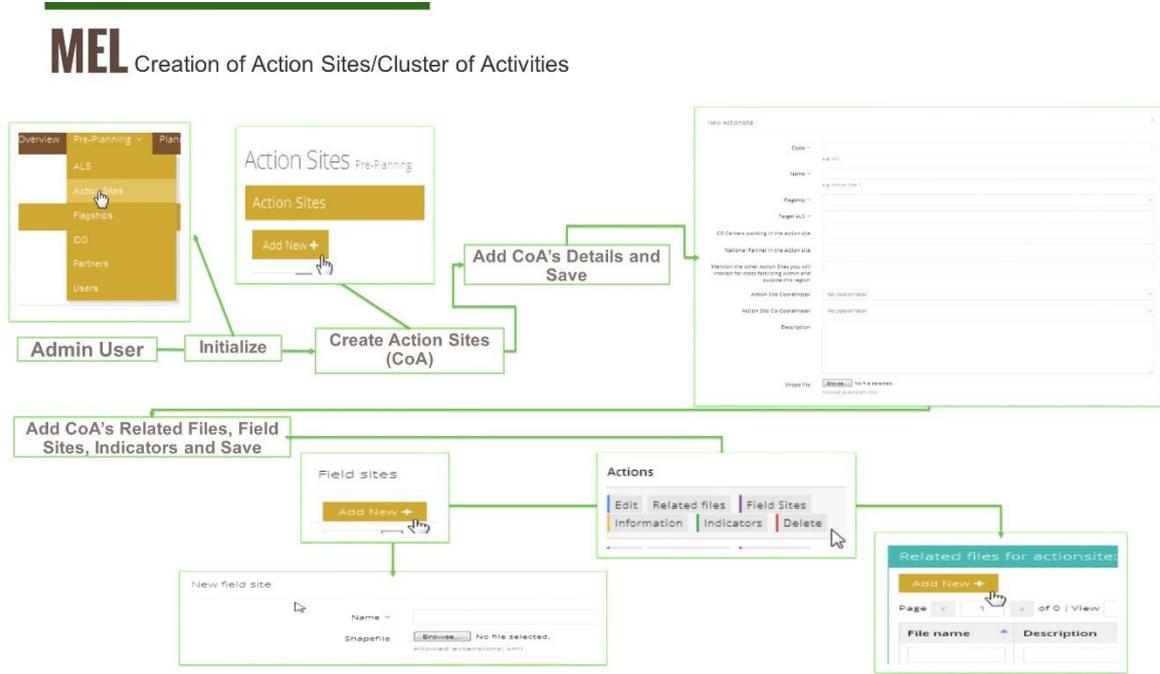


Figure 49. Creation of Projects Agreements Operation

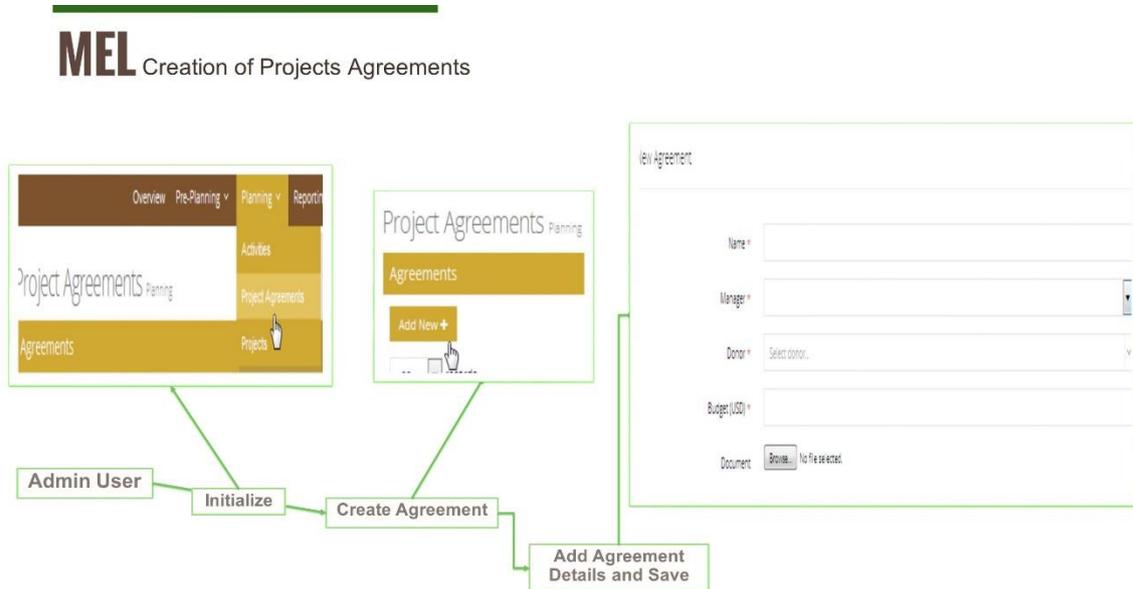


Figure 50. Creation of Projects Operation

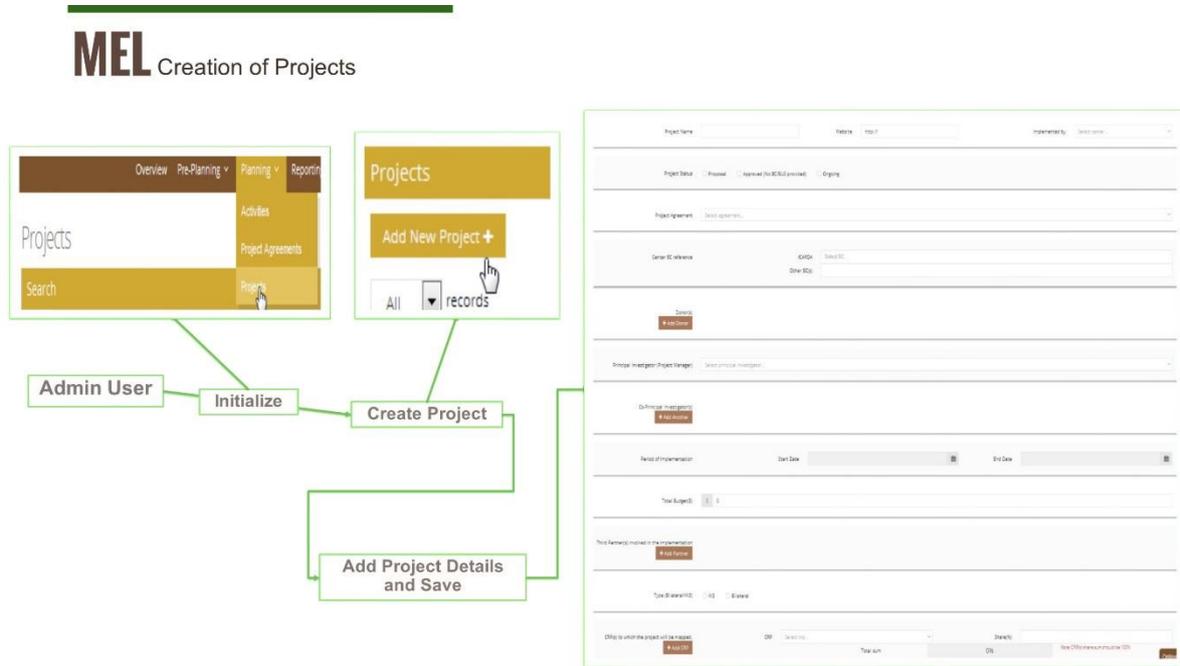


Figure 51. Creation of Activities Operation

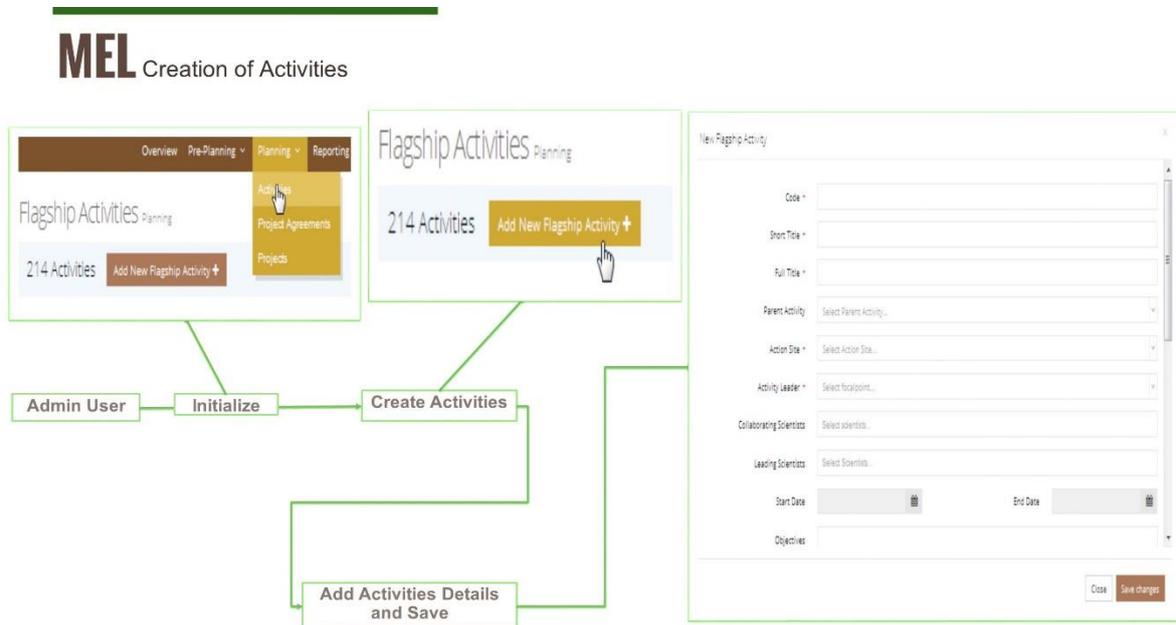


Figure 52. Editing Projects Information Operation

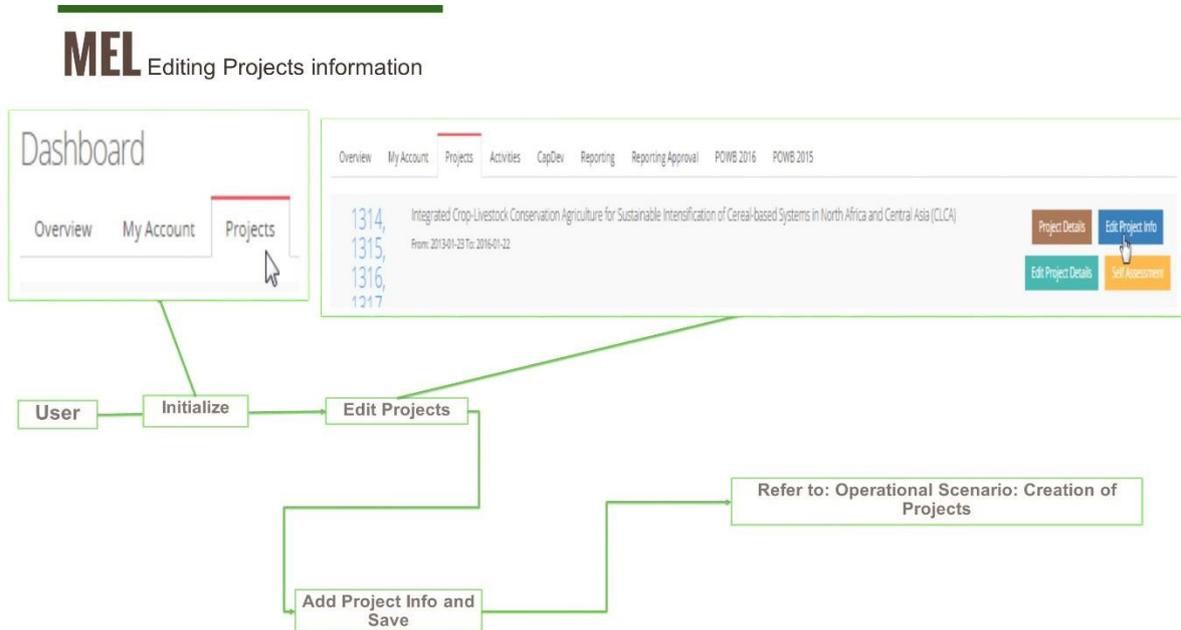


Figure 53. Editing Activities Information Operation

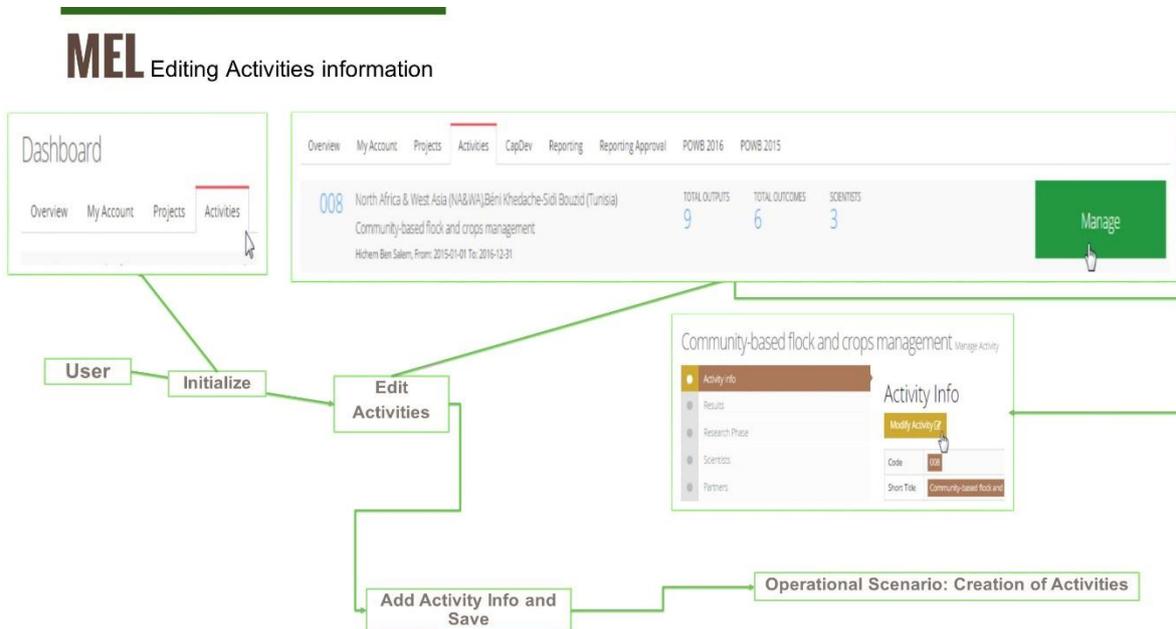


Figure 54. Planning within Projects and Activities Operation

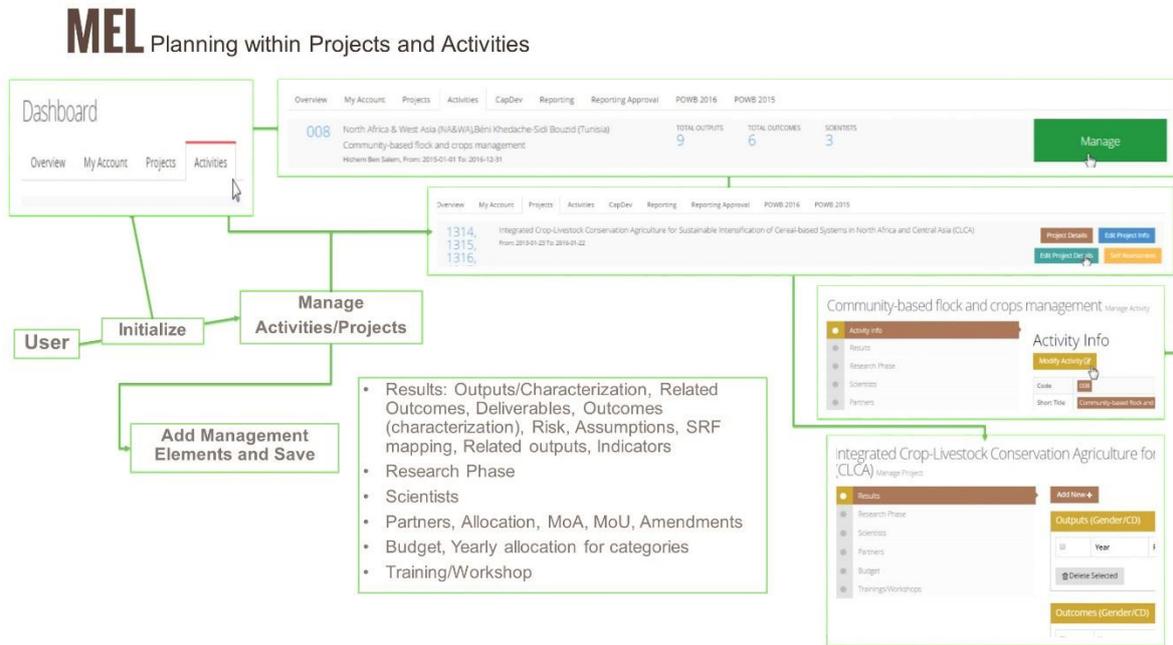


Figure 55. Reporting Operation

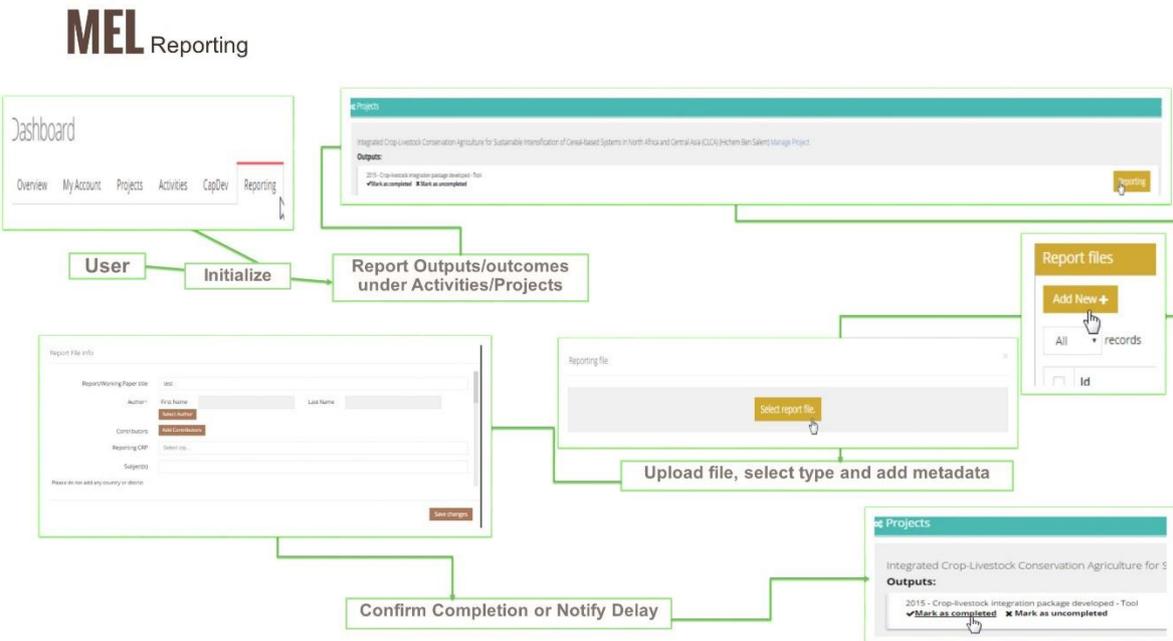


Figure 56. Self-Assess a project Operation

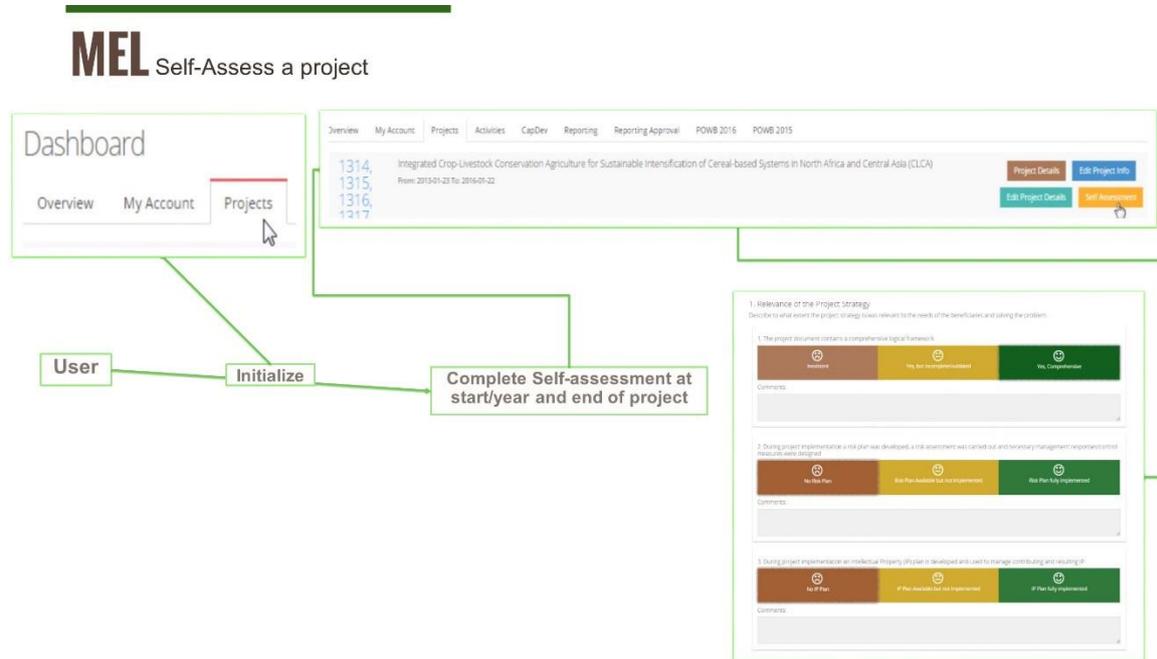


Figure 57. Launching and Reviewing a Survey Operation

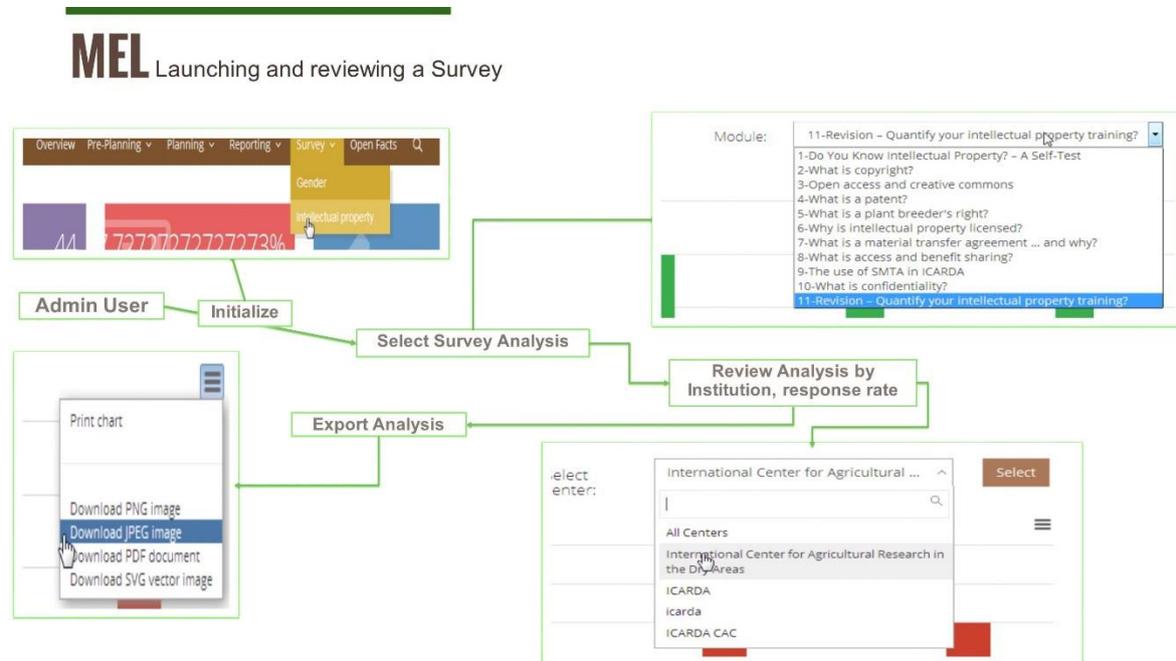




Figure 58. Consulting Overview Operation

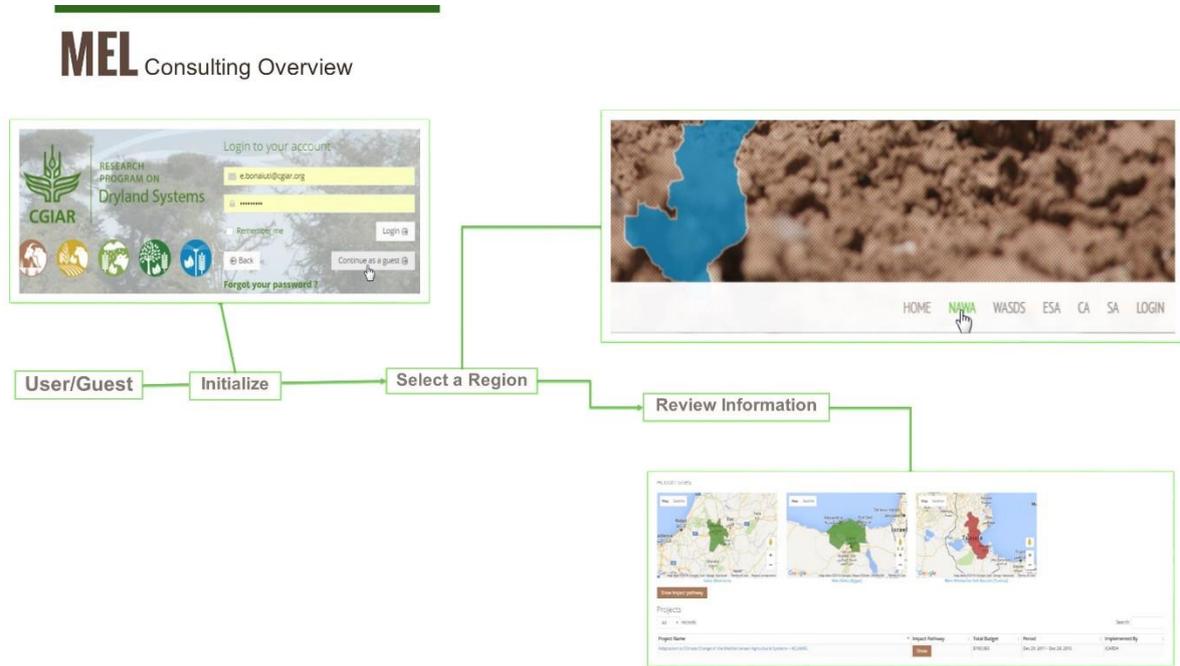


Figure 59. Consulting Open Facts Operation

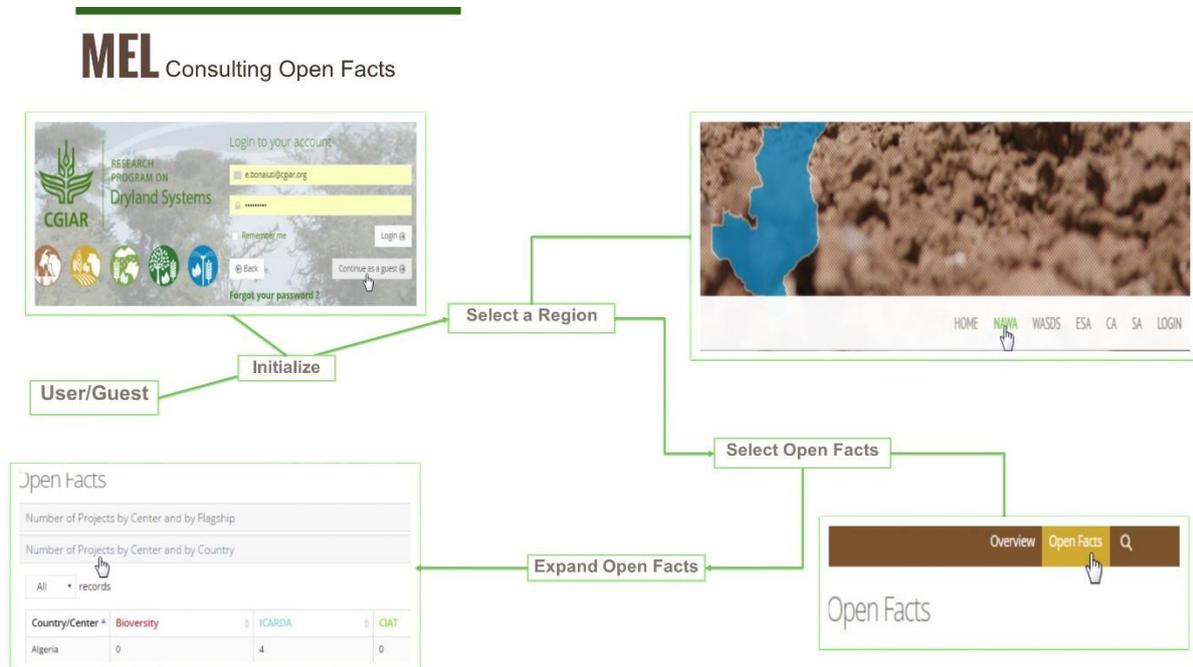


Figure 60. Exporting Data Operation

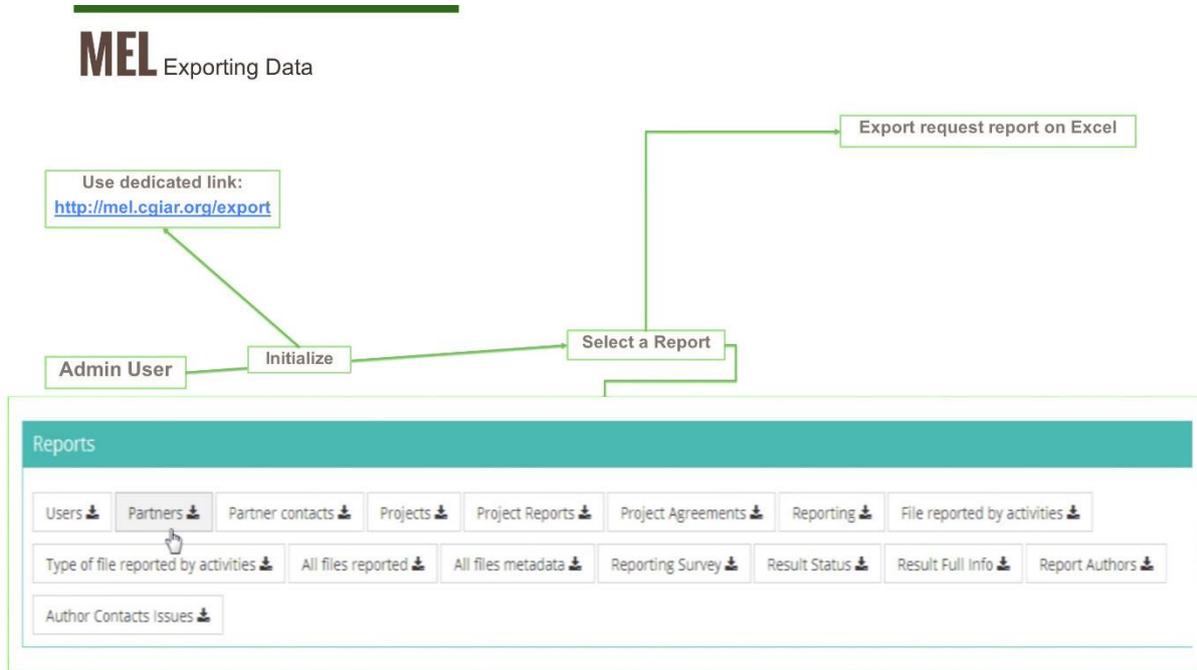


Figure 61. FP and CoA Leaders Approval Operation



Figure 62. Partners and Contacts Approval Operation

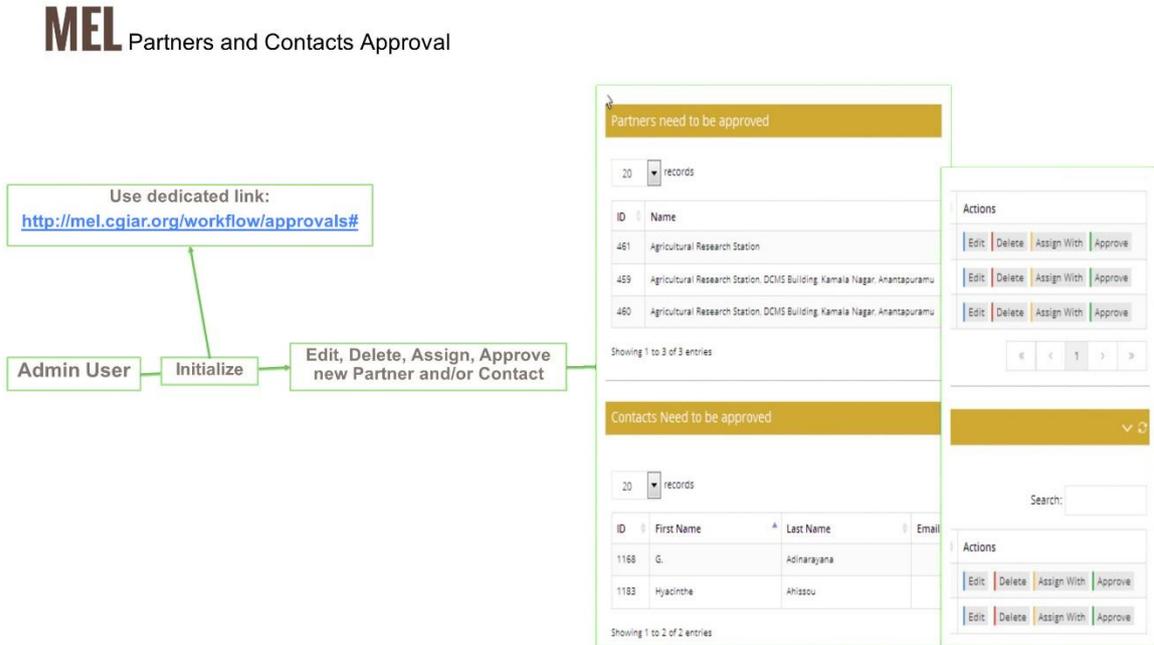


Figure 63. Open Access Approval Operation

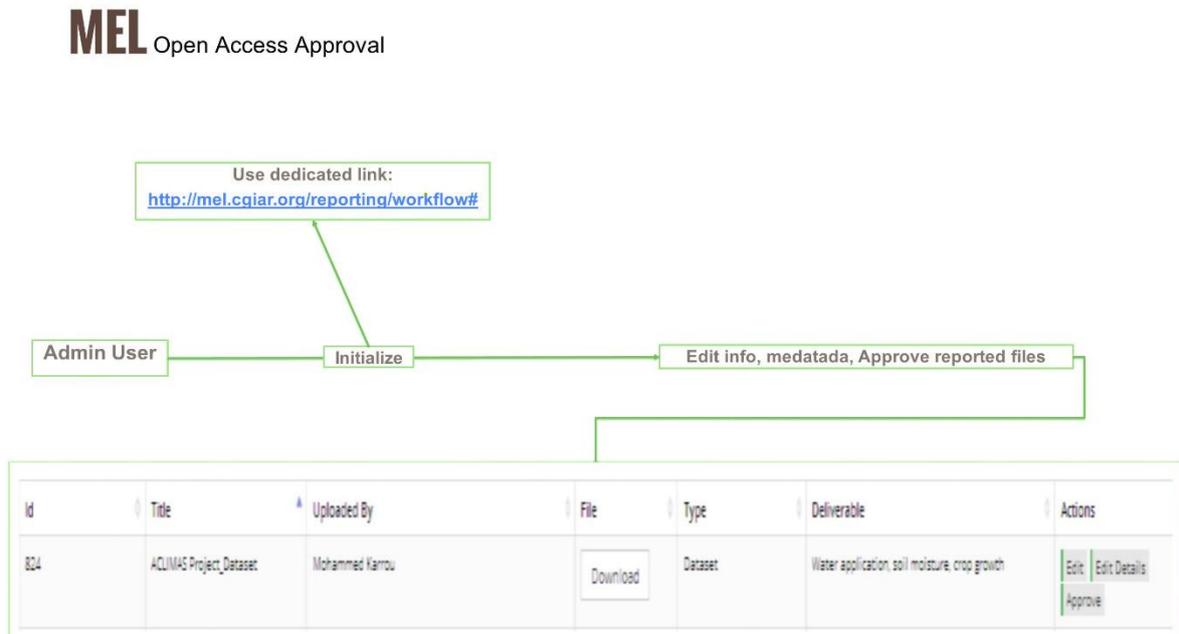


Figure 64. Discussion Forum Operation

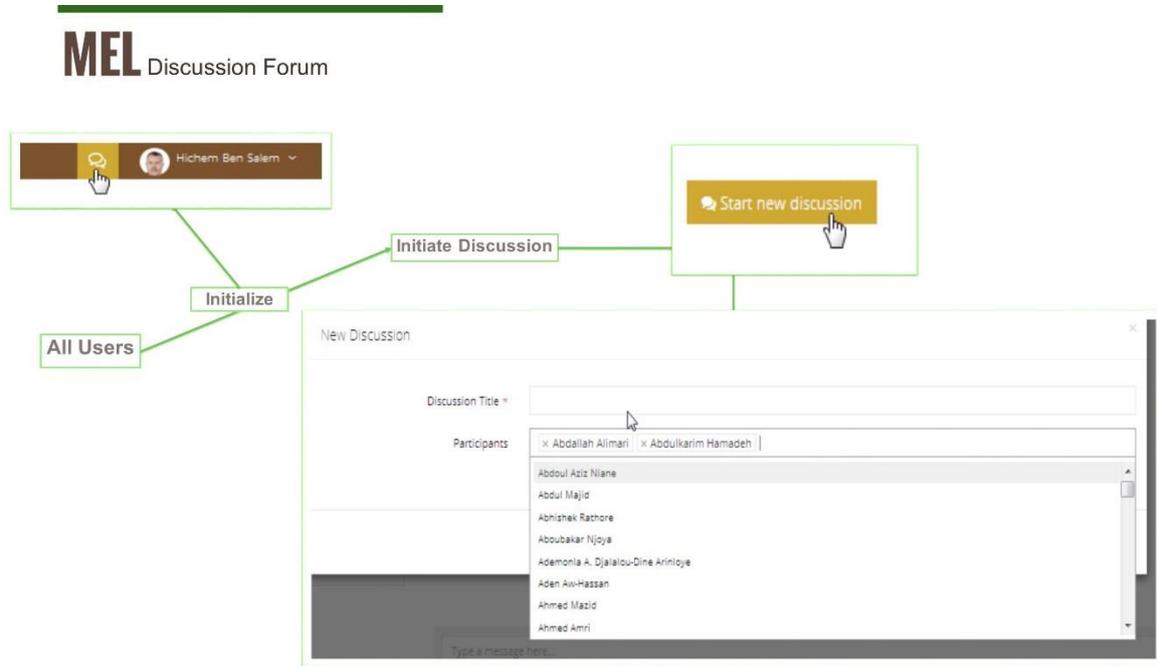
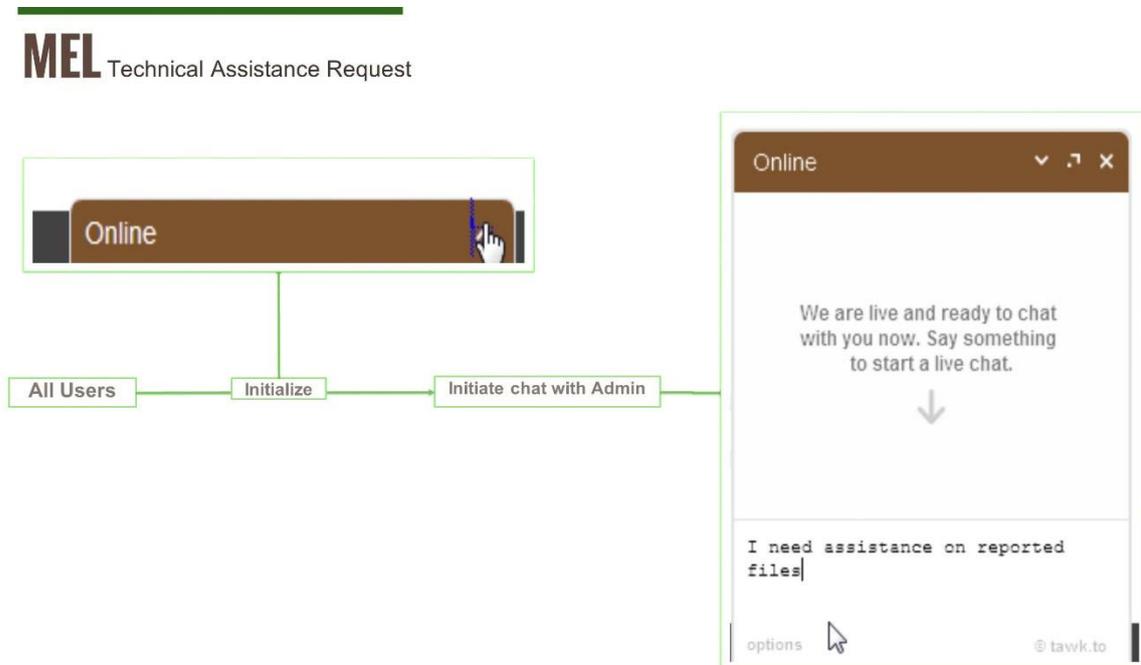


Figure 65. Technical Assistance Request Operation



6.3 Data Flow Diagrams

Instructions: Provide different levels of DFDs; summary of top-level, system level (between system(s)/users/modules) for each software module, and one layer inside the software module.

The following flowchart shows Administrator, Activity/Project Leader, Cluster of Activity/Action Site Leader, and Flagship Leader data flow diagram:

Figure 66. Activity/Project Creation Data Flow Diagram

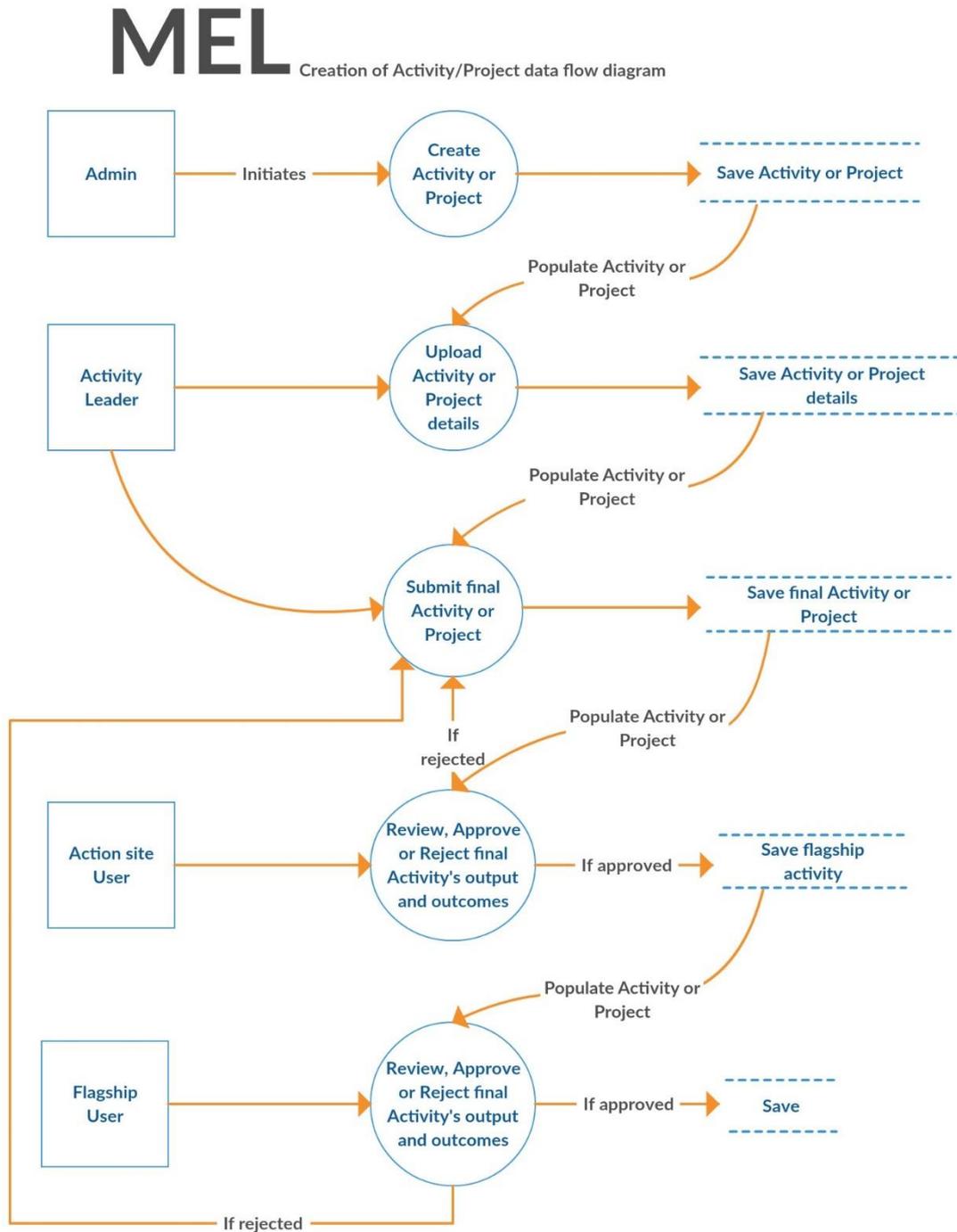


Figure 67. Pre-planning, Exporting, Open Facts Consulting Data Flow Diagram for Administrator, Activity/Project Leader, Cluster of Activity/Action Site Leader, and Flagship Leader

MEL

Pre-planning, Exporting, Open Facts consulting data flow diagram

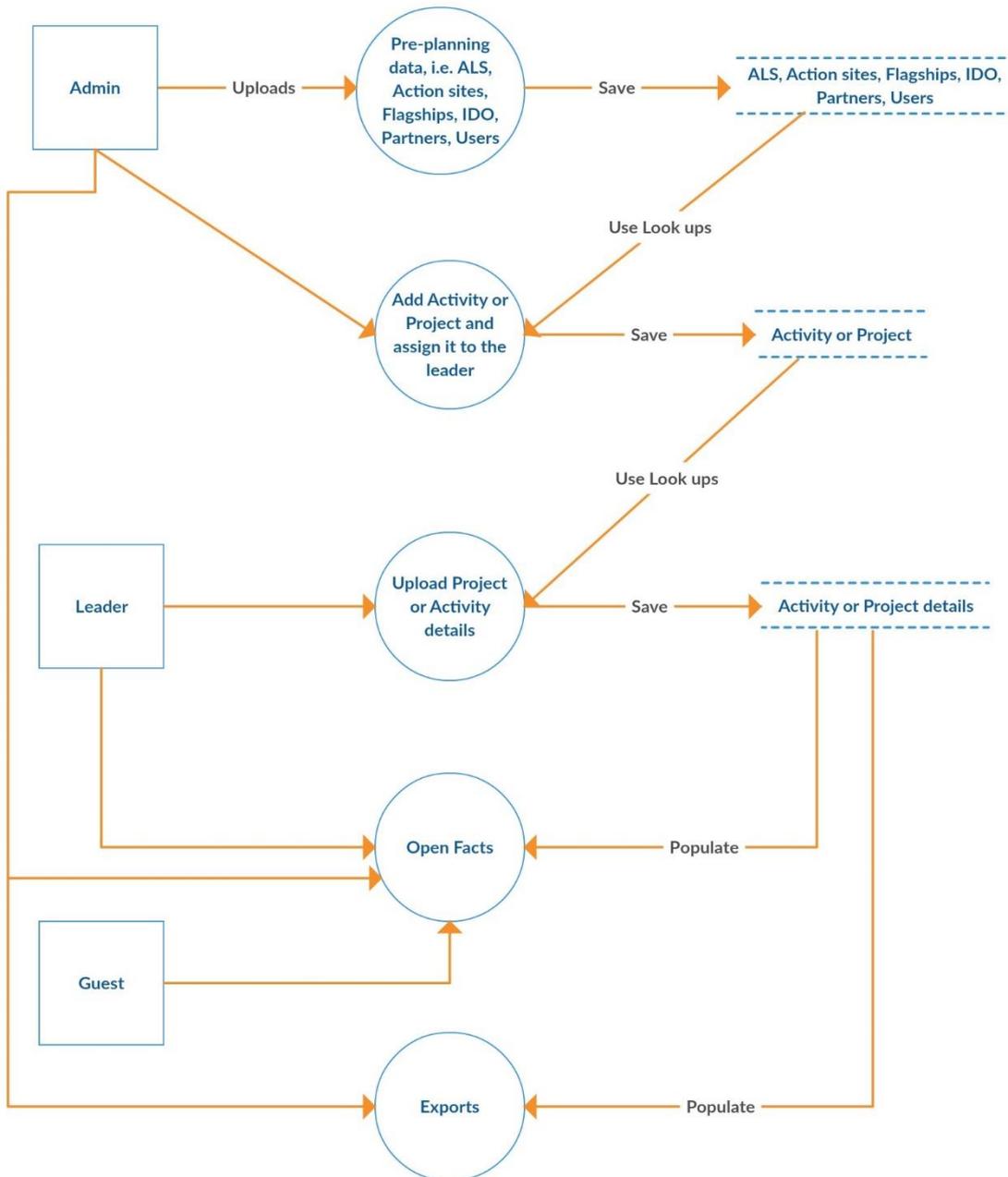


Figure 68. Reporting Data Flow Diagram for Administrator, Activity/Project Leader, Cluster of Activity/Action Site Leader, and Flagship Leader

MEL

Reporting data flow diagram

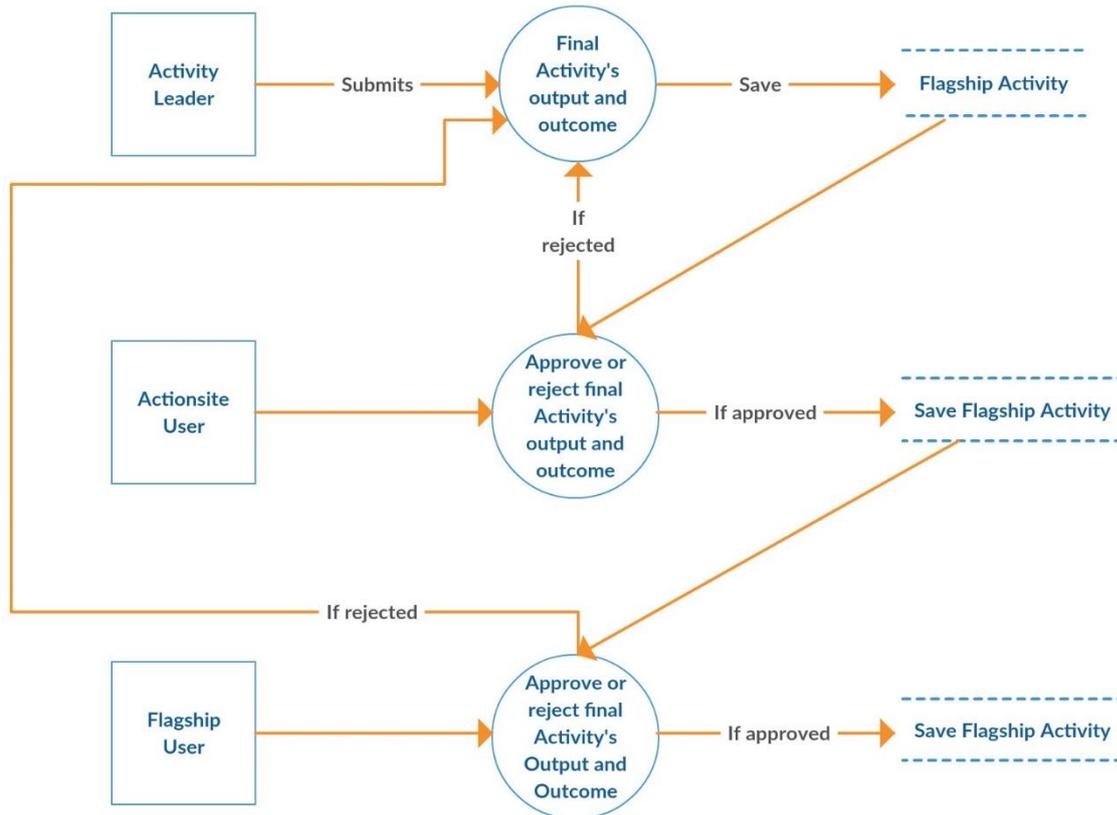


Figure 69. Open Access Approval data flow diagram for Administrator, Activity/Project Leader, Cluster of Activity/Action Site Leader, and Flagship Leader

MEL

Open access approval data flow diagram

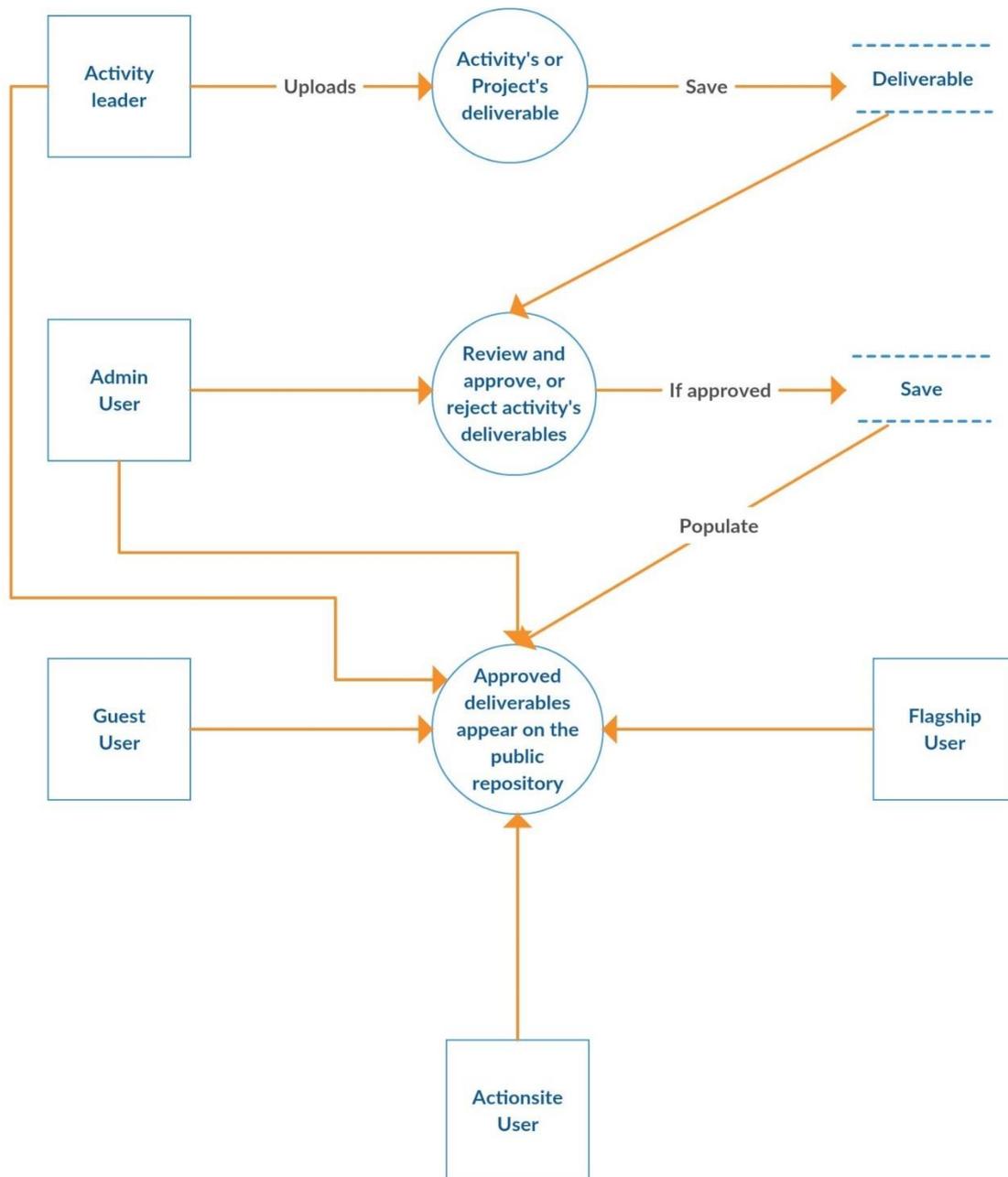
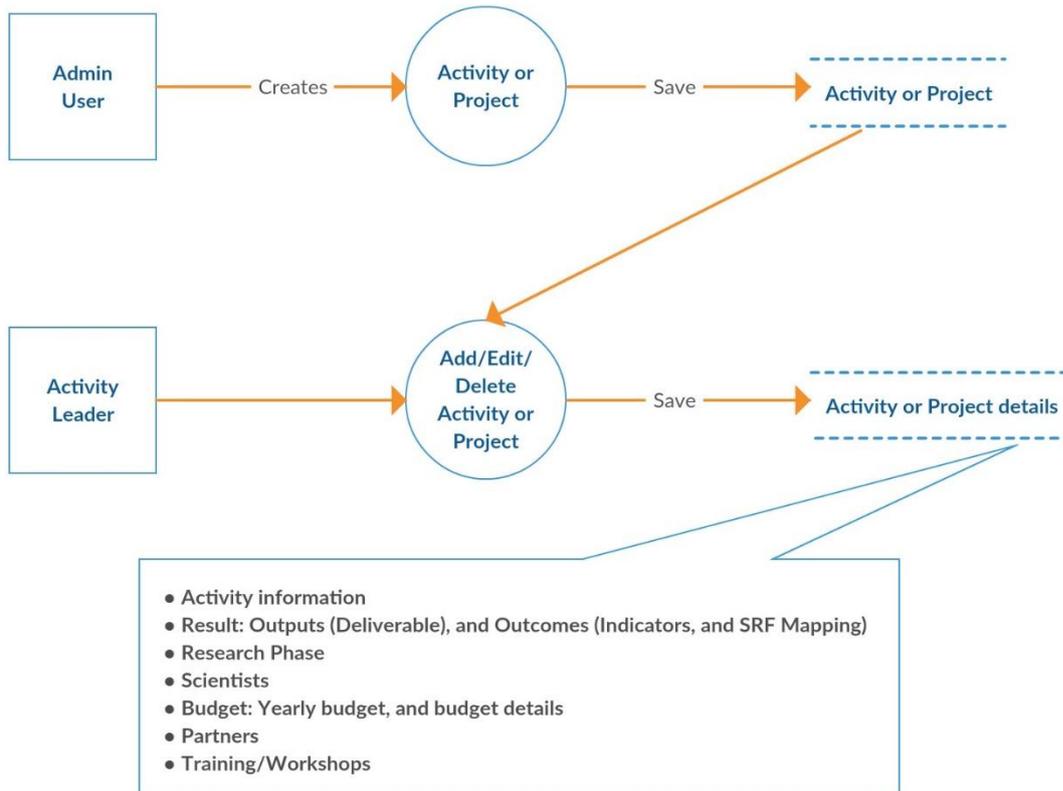


Figure 70. Editing Projects/Activities Data Flow Diagram for Administrator, Activity/Project Leader, Cluster of Activity/Action Site Leader, and Flagship Leader

MEL

Editing Projects/Activities information





7. Detailed Design

Instructions: Provide the information needed for a system development team to actually build and integrate the hardware components, code and integrate the software components, and interconnect the hardware and software segments into a functional product. Additionally, address the detailed procedures for combining separate COTS packages into the system.

7.1 Quick Installation Guide

This guide is for rapid deployment on Ubuntu operating systems.

- **System requirements:**

On all Operating systems available, development environment needs to have the following installed before running Zend Framework:

- Apache, MySQL, PHP 5.2.11 or later.

- **Installation Guide**

- Zend Framework requires the PHP rewrite module to be installed and enabled, the following command will enable it:

sudo a2enmod rewrite

- Create Database by the following:

***mysql -u RootUser -p
create database DbName;***

- MEL System Database has some defined users on it, you will need to create those users, and grant access by the following:

***create user 'jalal'@'%';
create user 'remote'@'%';
grant all on *.* to 'jalal'@%' identified by 'root' with grant option;
grant all on *.* to 'remote'@%' identified by 'root' with grant option;
flush privileges;***

- Finally, you will need to update the definer on your DB:

UPDATE mysql.proc SET definer = 'root@localhost' WHERE db = 'DbName';

- Place your code instance inside your local server directory, which will be under:

var/www/html

Or

htdocs/

- Go to your MEL system root directory /application/configs/application.ini, and change the following database parameters to what you have on your system:

***resources.db.params.username = DbUserName
resources.db.params.password = DbPassword
resources.db.params.dbname = DbName***

- Ubuntu system users of MEL has to symlink the following directories to come over case-sensitivity on Ubuntu:

application/model > Make Link > Then rename the Link to Model

Symlink application/Model/mapper > Make Link > Then rename the Link to Mapper



- MEL system can be accessed via the webserver
http://localhost/MELRoot

7.2 Conceptual Infrastructure Design

Instructions: Describe the infrastructure selected and why. Pay attention to Operating Systems, Versions of Software, Fault tolerance in the setup, Service Level Agreements and User Loads

MEL is using the following operating system, and software versions:

- **OS:** Windows Server 2012.
- **Apache:** 2.4.17
- **PHP:** 5.6.14
- **MySQL:** 5.7

Zend Framework was chosen due to development team expertise, extensibility, standards firmness, and that it's supported by Zend (The PHP Company). Windows Server was chosen because the already existed server is Windows as ICARDA uses GIS.

7.3 Hardware Detailed Design

Instructions: Provide enough detailed information about each of the individual hardware components to correctly build and/or procure all the hardware for the system (or integrate COTS items). If there are many components or if the component documentation is extensive, place it in an appendix. Add additional diagrams and information, if necessary, to describe each component and its functions adequately. Industry-standard component specification practices should be followed. For COTS components, identify specific vendor and appropriate item names and model numbers. Include the following information in the detailed component designs, as applicable:

- **Application Locations**

MEL is using XAMPP package and all installation locations are based on default one (c:\xampp).
Location: CGNET Server.

Table 14. Application Locations

Application Component	Description	Location	Type
MySQL database	Database	CGNET Server:\xampp\mysql	Data
SVN	Version control	CGNET Server	Code repository
PHP	programming language	CGNET Server:\xampp\php	Backend programming language
XAMPP	Web server	c:\xampp\htdocs	Cross-platform server solution

- **Server Hardware**

- Processor: Intel(R) Xeon(R) E5-2603 v3 @ 1.60GHz.
- Installed memory (RAM): 16.0 GB (15.7 GB usable).

7.4 Application Users

Instructions: Provide a description of each user class or role associated with the system. A user class is distinguished by the ways in which users interact with the proposed system or situation. Factors that distinguish a user class include common responsibilities, skill levels, work activities, and modes of interaction with the system. In this context, a user is anyone who interacts with the proposed system, including operational users, data entry personnel, system operators, operational support personnel, system maintainers, and trainers. For each user class, provide estimates of the total number of users anticipated, a maximum number of concurrent users, and the number of external users.

MEL roles are divided as: Admin, User, and Guest. User has some assertion rules that will give some users privileges over other users. A detailed explanation for each assertion rule can be found at Section 7: Software Modules (Zend_Acl). A role and resource mapping can be found at (Table 18).

Table 15. Role and Resource Mapping

Resources	Admin	User	Guest	Action	Assert
Index	Allow	Allow	Allow	All	Not Asserted
Api	Allow	Allow	Allow	All	Not Asserted
Overview	Allow	Allow	Allow	All	Not Asserted
Dataanalysis	Allow	Allow	Allow	All	Not Asserted
User	Allow	Partial allow	Partial allow	Index, deluser, deluserdiscipline, submituserdiscipline.	Not Asserted
Error	Allow	Allow	Allow	All	Not Asserted
Preplanning	Allow	Partial allow	Deny	All submit actions, All del actions	App_Acl_Assert_Preplanning
Planning	Allow	Partial allow	Deny	Index, Submitflagshipactivity	App_Acl_Assert_Planning
planning_leader	Deny	Deny	Deny	All	Not Asserted
Project	Allow	Partial allow	Partial allow	details, getallagreements, edit, form, submitproject	Custom Assertion
Selfassessment	Allow	Partial allow	Deny	All	App_Acl_Assert_SelfAssessment
Export	Allow	Partial allow	Deny	centerresults	Not Asserted
Cd	Allow	Allow	Deny		Not Asserted
Reporting	Partial allow	Partial allow	Partial allow	Workflow, download	Not Asserted
Discussion	Allow	Allow	Deny	All	Not Asserted
Graph	Allow	Allow	Allow	All	Not Asserted
Gender	Allow	Allow	Allow	All	Not Asserted



Resources	Admin	User	Guest	Action	Assert
Log	Allow	Deny	Deny	All	Not Asserted
Ip	Allow	Allow	Allow	All	Not Asserted
Import	Allow	Deny	Deny	All	Not Asserted

7.4.1 Inputs

Instructions: Provide a description of the input media used by the user/operator for providing information to the system. Show a mapping to the high-level data flows (e.g., data entry screens). If appropriate, the input record types, file structures, and database structures provided in the section for Data Design, may be referenced. Include data element definitions, or refer to the data dictionary. Provide the layout of all input data screens or graphical user interfaces (GUIs) (e.g., windows). Define all data elements associated with each screen or GUI, or reference the data dictionary. Provide edit criteria for the data elements, including specific values, range of values, mandatory/optional, alphanumeric values, and length. Also address data entry controls to prevent edit bypassing.

MEL system users use the web-based application interface to input projects related data, on the pre-planning, planning, and surveying phases. This interface includes tables that are filled by editing the required part of the project and/or program.

Reporting phase has its own uploading feature, allowing users to input different deliverables files i.e. 'jpg','jpeg','pdf','doc','docx','xls','xlsx','tif','ppt','pptx','zip','rar','png','gif','mp4','3gpp'.

While the allowed extensions in the related files section are:
'jpg','jpeg','pdf','doc','docx','xls','xlsx','tif','ppt','pptx','zip','rar','png','gif','mp4'.

There is no limitation on which operating system users have to use, as MEL is a web-based application. Best tested browsers are: Chrome, and Mozilla Firefox.

7.4.2 Outputs

Instructions: Describe the system output design relative to the user/operator. Show a mapping to the high-level data flows. System outputs include reports, data display screens and GUIs, query results, etc. The output files described in the section for Data Design may be referenced. The following should be provided, if appropriate:

- a) Identification of codes and names for reports and data display screens
- b) Description of report and screen contents (provide a graphical representation of each layout and define all data elements associated with the layout or reference the data dictionary)
- c) Description of the purpose of the output, including identification of the primary users
- d) Report distribution requirements, if any (include frequency for periodic reports)
- e) Description of any access restrictions or security considerations

All the projects, programs, deliverables, outputs, outcomes from the completed projects gives a repository of knowledge that ICARDA can re-use to get more projects, and to show high potential achievement history.



7.5 Software Detailed Design

Instructions: Provide a detailed description for each system software Service that addresses the following software Service attributes. Much of the information that appears in this section should be contained in the headers/prologues and comment sections of the source code for each component, subsystem, module, and subroutine. If so, this section may largely consist of references to or excerpts of annotated diagrams and source code.

Service Identifier – the unique identifier and/or name of the software Service.

Purpose –

Summary of Functions -

Classification – the kind of Service (e.g., application, data service, etc.)

Definition – the specific purpose and semantic meaning of the Service.

Requirements – the specific functional or non-functional requirements that the Service satisfies.

Data Definitions - Internal / External Data Structures – the internal/external data structures for the Service.

Constraints – any relevant, assumptions, limitations, or constraints for the Service. This should include constraints on timing, storage, or Service state, and might include rules for interacting with the Service (encompassing pre-conditions, post-conditions, invariants, other constraints on input or output values and local or global values, data formats and data access, synchronization, exceptions, etc.)

Composition – a description of the use and meaning of the subservices that are a part of the Service.

Users/Interactions – a description of the Service’s collaborations with other Services. What other Services is this entity used by? What other Services do this entity use (including any side-effects this Service might have on other parts of the system)? This includes the method of interaction, as well as the interaction itself. Object-oriented designs should include a description of any known or anticipated sub-classes, super-classes, and meta-classes.

Processing – a description of precisely how the Service goes about performing the duties necessary to fulfil its responsibilities. This should encompass a description of any algorithms used; changes or state; relevant time or space complexity; concurrency; methods of creation, initialization, and clean-up; and handling of exceptional conditions.

Language/Implementation approach/Error Handling

Models/Views/Controllers

Execution Location/List of Source Files

Diagrams

Interfaces/Exports – the set of services (resources, data types, constants, subroutines, and exceptions) that are provided by the Service. The precise definition or declaration of each such element should be present, along with comments or annotations describing the meanings of values, parameters, etc. For each service element described, include or provide a reference in its discussion to a description of its important software Service attributes (Component Identifier, Classification, Language, SLOC Estimate, Definition, Responsibilities, Requirements, Internal Data Structures, Constraints, Composition, Uses/Interactions, Resources, Processing, and Interfaces/Exports).

Reporting Design and Integration – if built in, provide details on data traffic and volumes.

7.5.1 Authentication (Zend_Auth)

Zend_Auth is concerned with determining whether an entity actually is what it purports to be (i.e., identification), based on some set of credentials. It is used to authenticate against a particular type of authentication service, such as LDAP, RDBMS, or file-based storage. Additionally, it comes with three different adapters that can be used to define custom authentication methods; Zend_Auth_Adapter_DbTable, Zend_Auth_Adapter_Digest, and Zend_Auth_Adapter_Http.

At MEL, Zend_Auth_Adapter_DbTable adapter is used to authenticate against RDBMS via instantiating this adapter at "library/App/Auth.php".

MEL has its custom authentication class "App_Auth" located at "library/App/Auth.php", this class defines one method, authenticate(), which is implemented to perform an authentication query.

```

public function authenticate($user, $password) {
    $authAdapter = new Zend_Auth_Adapter_DbTable (
Zend_Db_Table::getDefaultAdapter (), $this->_name, $this->_identityColumn, $this-
>_passwordColumn );
    try {
        $_userMapper= new Model_Mapper_User();
        $UserByEmail = $_userMapper->fetchOne(array('email'=>$user));
    } catch ( Exception $e ) {
        return -1;
    }
    $credential= sha1 ( $password . $UserByEmail->salt );
    $authAdapter->setIdentity ( $user )->setCredential ( $credential );

    $result = $authAdapter->authenticate ();

    if ($result->isValid () ) {

        $this->_storage = $storage = $this->_auth->getStorage ();

        $storage->write ( $UserByEmail );
    }
    return $result->getCode();
}

```

Prior to calling authenticate(), class "App_Auth" is prepared by setting up credentials (e.g., username and password) and defining values for adapter-specific configuration options, such as database connection settings for a database table adapter. This preparation is done by the following:

1. Defining adapter-specific configuration options as protected class properties:

```

class App_Auth {
    protected $_name;
    protected $_identityColumn;
    protected $_passwordColumn; }

```

2. Passing them to the class Constructor method, to be required on App_Auth class instantiating:

```

public function __construct($tableName, $identityColumn, $passwordColumn) {
    $this->_name = $tableName;
    $this->_identityColumn = $identityColumn;
    $this->_passwordColumn = $passwordColumn;
}

```



}

Previous configuration options include:

- **tableName:** This is the name of the database table that contains the authentication credentials, and against which the database authentication query is performed.
- **identityColumn:** This is the name of the database table column used to represent the identity. The identity column must contain unique values, such as a username or e-mail address.
- **passwordColumn:** This is the name of the database table column used to represent the credential. Under a simple identity and password authentication scheme, the credential value corresponds to the password.

As MEL is using `Zend_Auth_Adapter_DbTable` to authenticate against credentials stored in a database table, and because `Zend_Auth_Adapter_DbTable` requires an instance of `Zend_Db_Adapter_Abstract` to be passed to its constructor -that serves as the database connection to which the authentication adapter instance is bound-, each instance is bound to a particular database connection. Other configuration options may be set through the constructor and through instance methods, one for each option.

Next, class `App_Auth` will implement the `authenticate()` method to perform an authentication query by the following:

1. Creation of an adapter to authenticate against credentials stored in a database table. Instantiating `Zend_Db_Table::getDefaultAdapter` and passing the three previous class properties (`$tableName`, `$identityColumn`, and `$passwordColumn`):

```
$authAdapter = new Zend_Auth_Adapter_DbTable (
    Zend_Db_Table::getDefaultAdapter (), $this->_name, $this->_identityColumn, $this->_passwordColumn );
```

At this point, the authentication adapter instance is ready to accept authentication queries.

2. In order to formulate an authentication query, the input credential values (`$user`, `$password`) are passed to the adapter:

```
$authAdapter->setIdentity ( $user )->setCredential ( $credential );
```

But, before this we need to retrieve the table row upon authentication success which is done by:

```
try {
    $_userMapper= new Model_Mapper_User();
    $userByEmail = $_userMapper->fetchOne(array('email'=>$user));
} catch ( Exception $e ) {
    return -1;
}
```

Now, we know that we are setting the user identity "`setIdentity ($user)`" to its email, what are we setting at "`setCredential ($credential)`"?



We are setting the user password, which is calculated via:

```
$credential= sha1 ( $password . $UserByEmail->salt );
```

This manipulation must be identical with what the User model is setting for the user password, at "application/model/User.php":

```
public function _setRawpassword($password)
{
    $_salt = $this->salt;
    $_saltedPassowrd = sha1($password . $_salt);
    $this->password = $_saltedPassowrd;
}
```

Lastly, Zend_Auth_Adapter_DbTable returns the identity supplied back to the auth object upon successful authentication.

The following code is storing an instance of Zend_Auth_Result which is what Zend_Auth adapters return to represent the results of an authentication attempt.

```
$result = $authAdapter->authenticate ();
```

To determine if this authentication attempt went successfully, isValid() method is used as follows:

```
if ($result->isValid ()) {
    $this->_storage = $storage = $this->_auth->getStorage ();
    $storage->write ( $UserByEmail );
}
return $result->getCode();
}
```

After checking if the attempt was successful, we are getting the Zend_Auth's default storage and storing in it the user information. We store data to the auth adapter for use in all subsequent requests. The default storage is a session with namespace Zend_Auth.

For the getCode() method: It returns a Zend_Auth_Result constant identifier for determining the type of authentication failure or whether success has occurred. We are using return statement to operate upon its code at "application/modules/default/controllers/UserController.php" as follows:

```
if ($result > 0)
    $this->_helper->json->sendJson(array(
        'message' => 'success'
    ));
elseif ($result == -1)
    $this->_helper->json->sendJson(array(
        'message' => 'Account not Found'
    ));
elseif ($result == -99)
```

```
$this->_helper->json->sendJson(array(
    'message' => 'Account not Active'
));
else
$this->_helper->json->sendJson(array(
    'message' => 'Login Failed'
));
```

Till now, MEL is using the Zend_Auth adapters in a direct way through the adapter's authenticate() method.

The following illustrates how MEL is using Zend_Auth adapter indirectly, through Zend_Auth::authenticate():

- This is the logoutAction located at (application/modules/default/controllers/UserController.php). It's clearing the identity from Zend_Auth, which is also clearing all data from the Zend_Auth session namespace. And, redirecting back to the home page.

```
public function logoutAction()
{
    Zend_Auth::getInstance()->clearIdentity();
    Zend_Session::forgetMe();
    $this->_redirect('/');
}
```

Note: The Zend_Auth class implements the Singleton pattern - only one instance of the class is available - through its static getInstance() method. This means that using the new operator and the clone keyword will not work with the Zend_Auth class; use Zend_Auth::getInstance() instead.

- Another example of MEL using Zend_Auth indirectly is at (application/modules/default/controllers/ProjectController.php). Here we are trying to get a reference to the singleton instance of Zend_Auth stored in \$_auth, then checks if the identity exists to forward to the deniedAction() inside UserControlled.php as follows:

```
$_auth = Zend_Auth::getInstance();
if($_auth->hasIdentity())
return $this->forward('denied','user');
else
return $this->forward('login','user');
}
```

7.5.2 Authorization (Zend_Acl)

Authorization, is the process of deciding whether to allow an entity to access to, or to perform operations upon, other entities. Access Control List (ACL) indicates who has access to do what on a given resource, this list is used to control access to certain protected objects by other requesting objects.



There are two main concepts at in Zend_Acl: Resources and Roles. A Resource is something that needs to be accessed and a Role is the thing that is trying to access the Resource. To have access to a resource, you need to have the correct Role.

- **Resources**

For Zend_Acl to recognize an object as a resource; a class needs to implement Zend_Acl_Resource_Interface interface, which consists of a single method, getResourceId(). Zend_Acl_Resource is provided by Zend_Acl as a basic resource implementation to be extended as needed.

You can add multiple resources to Zend_Acl, these resources will be added as tree structure. Tree structure allows to organize resources from general to the specific. The resource on the top of tree structure will have general privileges, while nodes down in the tree structure will have privileges that are more specific.

Zend_Acl provides a tree structure to which multiple resources can be added. Since resources are stored in such a tree structure, they can be organized from the general (toward the tree root) to the specific (toward the tree leaves). Queries on a specific resource will automatically search the resource's hierarchy for rules assigned to ancestor resources, allowing for simple inheritance of rules. A resource may inherit from only one parent resource, though this parent resource can have its own parent resource, etc.

- **Roles**

For Zend_Acl to recognize an object as a role; a class needs to implement Zend_Acl_Role_Interface interface, which consists of a single method, getRoleId(). Zend_Acl_Role is provided by Zend_Acl as a basic role implementation to be extended as needed. MEL User model is implementing Zend_Acl_Role_Interface and is overriding getRoleId() ar (application/model/User.php) as follows:

```
class Model_User extends App_Model_ModelAbstract implements
Zend_Acl_Role_Interface {
    public function getRoleId()
    {
        return $this->role;
    }
}
```

A Role may inherit from one or more Roles. This is to support inheritance of rules among roles. When specifying multiple parents for a role, keep in mind that the last parent listed is the first one searched for rules applicable to an authorization query.

- **MEL Access Control List**

At (library/App/Acl/Acl.php), MEL is creating its ACL by extending Zend_Acl as follows:

```
class App_Acl_Acl extends Zend_Acl { }
```

To get the list created at the instantiating time of class App_Acl_Acl; Roles registering, Resources assigning, and rules creation are done inside this class Constructor method.

- **Registering Roles**

At MEL, we have three main access requesting objects (Roles) that we need to grant access for to resources, here is how MEL is registering those Roles to the ACL along with the ACL for MEL:

Table 16. MEL Access Controls

Role Name	Inherits Permissions From
Guest	N/A
User	Guest
Admin	User

1. MEL is creating Roles using Zend_Acl_Role, but any object that implements Zend_Acl_Role_Interface can do the same.

```
public function __construct() {
    $guest = new Zend_Acl_Role ( "guest" );
    $user = new Zend_Acl_Role ( "user" );
    $admin = new Zend_Acl_Role ( "admin" );
}
```

2. MEL is defining Roles. Some Roles are defined with an additional argument, this argument specifies what role the new role inherits from. Thus, as we apply privileges for one role, any role that inherits from that role will also receive those privileges.

```
$this->addRole ( $guest )->addRole ( $user, "guest" )->addRole ( $admin,
"user" );
```

Thus, guest Role inherits no Role, user Role inherits guest, and admin inherits user Role which already inherits guest.

Any allowed Rule for guest will be by default allowed for user, and any allowed Rule for both user and guest Roles will be allowed by default to admin.

- **Adding Resources**

In Zend, resource can be a "module" or "controller" or "controller action" or file or any block of code. At MEL, we are defining our Resources as Controllers, except that we have a custom Rule created for checking purposes which is explained later on this module. Here is how MEL is creating and adding Resources for the Rules:

1. Resources creation via instantiation class App_Acl_Resource, and specifying the controller name:

```
$indexResource = new App_Acl_Resource ( "index" );
$apiResource = new App_Acl_Resource ( "api" );
$overviewResource = new App_Acl_Resource ( "overview" );
$dataanalysisResource = new App_Acl_Resource ( "dataanalysis" );
$userResource = new App_Acl_Resource ( "user" );
$errorResource = new App_Acl_Resource ( "error" );
$preplanningResource = new App_Acl_Resource ( "preplanning" );
$planningResource = new App_Acl_Resource ( "planning" );
$planningLeaderResource = new App_Acl_Resource ( "planning_leader" );
```

```

$projectResource = new App_Acl_Resource ( "project" );
$selfassessmentResource = new App_Acl_Resource ( "selfassessment" );
$exportResource = new App_Acl_Resource ( "export" );
$cdResource = new App_Acl_Resource ( "cd" );
$reportingResource = new App_Acl_Resource ( "reporting" );
$discussionResource = new App_Acl_Resource ( "discussion" );
$graphResource = new App_Acl_Resource ( "graph" );
$genderResource = new App_Acl_Resource ( "gender" );
$logResource = new App_Acl_Resource ( "log" );
$ipResource = new App_Acl_Resource ( "ip" );
$importResource = new App_Acl_Resource ( "import" );

```

Class App_Acl_Resource located at (library/App/Acl/Resource.php) is just the same as Zend_Acl_Resource, as it's extending it with no overrides, neither added properties/methods - It might be extended for a new feature to be added but was not done at that time.

```

class App_Acl_Resource extends Zend_Acl_Resource { }

```

2. Resources adding to the ACL.

```

$this->addResource ( $indexResource )->addResource ( $overviewResource )-
>addResource ( $dataanalysisResource )->addResource ( $userResource )-
>addResource ( $apiResource )->addResource ( $errorResource )->addResource (
$preplanningResource )->addResource ( $planningResource )->addResource (
$projectResource )->addResource ( $planningLeaderResource )-
>add($selfassessmentResource)->addResource($exportResource)-
>addResource($cdResource)->addResource($reportingResource)-
>addResource($discussionResource)->addResource($graphResource)-
>addResource($genderResource)->addResource($logResource)-
>addResource($ipResource)->addResource($importResource);

```

- **Defining Privileges (Rules):**

Rules are established to define how resources may be accessed by roles. Which is to specify privileges available on each resource based on the role accessing the resource. This is done via the allow() method.

Zend_Acl suggests implementing rules from general to specific, to minimize the number of rules needed -Resources and Roles inherit rules defined upon their ancestors-. Zend_Acl also obeys a rule if and only if a more specific rule does not apply.

At MEL, Rules/Privileges are controller's actions, the following snippet of code is defining Rules for each Role.

```

$this->allow ( $guest, $indexResource );
$this->allow ( $guest, $overviewResource );
$this->allow ( $guest, $dataanalysisResource );
$this->allow ( $guest, $userResource );
$this->allow ( $guest, $graphResource );
$this->allow ( $guest, $genderResource );

```

```

$this->allow ( $user, $reportingResource );
$this->deny ( $user, $reportingResource ,array('workflow'));
$this->allow ( $admin, $reportingResource ,array('workflow'));
$this->allow ( $guest, $reportingResource,array('download') );
$this->allow ( $guest, $ipResource );
$this->deny ( $guest, $userResource, array (
    'index',
    'deluser',
    'deluserdiscipline',
    'submituserdiscipline'
));
$this->allow ( $guest, $apiResource );
$this->allow ( $guest, $errorResource );
$this->allow ( $guest, $projectResource, array (
    'details'
));
$this->allow ( $user, $projectResource, array (
    'getallagreements',
    'edit',
    'form' ,
    'submitproject'
));

$this->allow ( $guest, $projectResource, array ('edit_check'), new
App_Acl_Assert_Project());

$this->allow ( $user, $preplanningResource, null, new App_Acl_Assert_Preplanning
());
$this->deny ( $user, $planningResource,'index');
$this->allow ( $user, $planningResource,null, new App_Acl_Assert_Planning ( ) );
$this->allow ( $user, $selfassessmentResource,null, new
App_Acl_Assert_SelfAssessment ( ) );
$this->allow ( $user, $cdResource);
$this->allow ( $user, $discussionResource);
$this->allow ( $user, $exportResource,array('centerresults'));
$this->allow ( $admin, $preplanningResource );
$this->allow ( $admin, $planningResource );
$this->allow ( $admin, $projectResource );
$this->allow ( $admin, $userResource );
$this->allow ( $admin, $selfassessmentResource );
$this->allow ( $admin, $exportResource);
$this->allow ( $admin, $cdResource);
$this->allow ( $admin, $logResource);
$this->allow ( $admin, $importResource);

```

This Rule allows guest Role to access all IndexController.php actions:



```
$this->allow ( $guest, $indexResource );
```

This Rule allows guest Role to access donwloadAction only from the ReportingController.php:

```
$this->allow ( $guest, $reportingResource,array('download') );
```

This Rule denies user Role from accessing the PlanningController.php indexAction:

```
$this->deny ( $user, $planningResource,'index');
```

This Rule allows user Role to access multiple actions (getallagreementsAction, editAction, formAction, submitprojectAction) from the ProjectController.php:

```
$this->allow ( $user, $projectResource, array (
    'getallagreements',
    'edit',
    'form' ,
    'submitproject'
));
```

- This long list of privileges can be easily enhanced by following Zend's best practices by implementing rules from general to specific, to minimize the number of rules needed. i.e. admin Role can inherit from no other Role but has it's Rule as follows:

```
$this->allow ($admin);
```

This indicates that admin inherits nothing, but allowed to all controllers and actions.

- MEL ACL Plugin (A Front Controller Plugin)
In Zend Framework applications, the resource and privilege can often be determined from the request object. Thus, we need to automatically check when there is a request for some controller action to be checked against the acl. This checking takes place in preDispatch() method that is called before every call to the controller action.

MEL has this done via a plugin which checks the acl and located at (application/plugins/Acl.php):

```
class Plugin_Acl extends Zend_Controller_Plugin_Abstract {
    public function preDispatch(Zend_Controller_Request_Abstract $request) {

        $module = $request->getModuleName();
        $controller = $request->getControllerName();
        $action = $request->getActionName();
        $_auth=Zend_Auth::getInstance ();

        if($_auth->hasIdentity())
        {
            $_user=$_auth->getIdentity();
            Zend_Registry::set('user', $_user);
        }
    }
}
```

```

        $_loggedIn=true;
    }
    else{
        Zend_Registry::set('user', null);
        $_user=new Model_User();
        $_user->user_id=-1;
        $_user->name='Guest';
        $_user->role='guest';
        $_loggedIn=false;
    }
    Zend_Registry::set('role', $_user->role);
    $_acl=new App_Acl_Acl();
    $_acl->setContextValue('request',$request);
    //return $request->setActionName('message')-
    >setControllerName('user')->setDispatched(true);
    if(!$acl->isAllowed($_user,$controller,$action)){
        if($_loggedIn)
            return $request->setActionName('denied')-
            >setControllerName('user')->setDispatched(true);
        else
            return $request->setActionName('login')-
            >setControllerName('user')->setDispatched(true);
    }

    parent::preDispatch($request);
}
}

```

In the code above, Plugin_Acl is creating a predispatch function that takes the current request as a parameter:

```

public function preDispatch(Zend_Controller_Request_Abstract $request)

```

Here MEL is getting the current request parameters, module name(which will always be default), controller name, and action name:

```

    $module = $request->getModuleName();
    $controller = $request->getControllerName();
    $action = $request->getActionName();

```

Here MEL is getting the Role from the Zend_Auth, if no Role is detected then it sets the Role to guest by default:

```

    $_auth=Zend_Auth::getInstance ();

    if($_auth->hasIdentity())
    {
        $_user=$_auth->getIdentity();
    }

```

```

        Zend_Registry::set('user', $_user);
        $_loggedIn=true;
    }
    else{
        Zend_Registry::set('user', null);
        $_user=new Model_User();
        $_user->user_id=-1;
        $_user->name='Guest';
        $_user->role='guest';
        $_loggedIn=false;
    }

```

MEL is setting the Role, and creating a new ACL:

```

    Zend_Registry::set('role', $_user->role);
    $_acl=new App_Acl_Acl();

```

MEL is requesting an access permission from the ACL to the controller and action for the provided role, if access is denied then it checks if the user is logged in it will be redirected to the deniedAction() in the UserController.php, otherwise a login is required. Lastly, even if we override the preDispatch() method, we are calling the parent method before exiting:

```

    if(!$acl->isAllowed($_user,$controller,$action)){
        if($_loggedIn)
            return $request->setActionName('denied')-> setControllerName('user') -
            >setDispatched(true);
        else
            return $request->setActionName('login')->setControllerName('user')-
            >setDispatched(true);
    }
    parent::preDispatch($request);

```

- **Registering the ACL plugin**

Zend will recognize the new ACL plugin by adding the following line inside (application/configs/application.ini):

```
resources.frontController.plugins.user = "Plugin_Acl"
```

- **Assertions**

Some MEL ACL Rules depends on further conditions to allow or deny access to a specific resource. Hence, MEL implemented Zend_Acl_Assert_Interface to support conditional rules at (library/App/Acl/Assert/Interface.php).

Zend_Acl_Assert_Interface usage is done by implementing it by the interface App_Acl_Assert_Interface, any implementing class for App_Acl_Assert_Interface needs to override the assert() method.

```
interface App_Acl_Assert_Interface extends Zend_Acl_Assert_Interface { }
```

All MEL assertions are mentioned below with a detailed explanation for each conditional Rule:

```

$this->allow ( $guest, $projectResource, array ('edit_check'), new
App_Acl_Assert_Project());
$this->allow ( $user, $preplanningResource, null, new App_Acl_Assert_Preplanning
());
$this->allow ( $user, $planningResource, null, new App_Acl_Assert_Planning ( ) );
$this->allow ( $user, $selfassessmentResource, null, new
App_Acl_Assert_SelfAssessment ( ) );

```

Zend ACL allow() public function is defined at (library/Zend/Acl.php), and has the following syntax:

```

public function allow($roles = null, $resources = null, $privileges = null,
Zend_Acl_Assert_Interface $assert = null)

```

Note: NULL value indicates application to all roles, resources (Controllers), and privileges (Actions).

- The \$preplanningResource assertion interface is located at (library/App/Acl/Assert/Preplanning.php), and has the following code inside:

```

class App_Acl_Assert_Preplanning implements App_Acl_Assert_Interface
{
    public function assert(Zend_Acl $acl,
        Zend_Acl_Role_Interface $role = null,
        Zend_Acl_Resource_Interface $resource = null,
        $privilege = null)
    {
        $_auth = Zend_Auth::getInstance();
        if (!$_auth->hasIdentity()) {
            return false;
        }
        $_user = $_auth->getIdentity();

        if(substr($privilege, 0,6)=='submit' || substr($privilege, 0,3)=='del')
            return false;

        return true;
    }
}

```

Once an assertion class is available (App_Acl_Assert_Preplanning), an instance of the assertion class (new App_Acl_Assert_Preplanning ()) must be supplied when assigning conditional rules. A rule that is created with an assertion only applies when the assertion method returns TRUE. This is done at (library/App/Acl/Acl.php) by this line:

```

$this->allow ( $user, $preplanningResource, null, new App_Acl_Assert_Preplanning
());

```



The above code creates a conditional allow Rule that allows the user Role to access the PreplanningController located at:

(application/modules/default/controllers/PreplanningController.php) on all its actions (NULL), access is asserted for authenticated users with identity only (logged in) if the action name starts with "submit", or "del".

As a result, the NULL value provided at the assertion Rule is overridden by the code inside the assertion interface, so only the following actions are permitted by this assertion:

- **Actions (methods) adheres if(substr(\$privilege, 0,6)=='submit'):**

```
public function submitdoAction()
public function submitalsAction()
public function submitorganizationAction()
public function submitflagshipAction()
public function submitactivityAction()
public function submitpartnerAction()
public function submitpartnercontactAction()
public function submitactionsiteAction()
public function submitdoindicatorAction()
public function submitflagshipindicatorAction()
public function submitdoindicatorvalueAction()
public function submitflagshipindicatorvalueAction()
public function submitflagshipinfoAction()
public function submitactionsiteinfoAction()
public function submitfieldsiteinfoAction()
public function submitfieldsiteAction()
public function submitactionsiteindicatorAction()
public function submitactionsiteindicatorvalueAction()
```

- **Actions adheres if(substr(\$privilege, 0,3)=='del')**

```
public function deleterelatedfileAction()
public function deleteentityfileAction()
protected function delidoAction()
protected function delalsAction()
protected function delorganizationAction()
protected function delflagshipAction()
protected function delactivityAction()
protected function delpartnerAction()
protected function delpartnercontactAction()
protected function delactionsiteAction()
protected function delidoindicatorAction()
protected function delflagshipindicatorAction()
protected function delidoindicatorvalueAction()
protected function delflagshipindicatorvalueAction()
protected function delfieldsiteAction()
protected function delactionsiteindicatorAction()
protected function delactionsiteindicatorvalueAction()
```

- The \$planningResource assertion interface is located at (library/App/Acl/Assert/Planning.php), and has the following code inside:

```
class App_Acl_Assert_Planning implements App_Acl_Assert_Interface {
```

```
public function assert(Zend_Acl $acl, Zend_Acl_Role_Interface $role = null,
Zend_Acl_Resource_Interface $resource = null, $privilege = null) {
```

```
    $_auth = Zend_Auth::getInstance ();
```

```
    if (! $_auth->hasIdentity ()) {
        return false;
    }
```

```
    $_user = $_auth->getIdentity ();
```

```
    if (! $acl->hasValue ( 'request' ))
        return true;
```

```
    $_request = $acl->getContextValue ( 'request' );
    $_id = $_request->getParam ( 'id', 0 );
    $_aid = $_request->getParam ( 'aid', 0 );
```

```
    $_flagshipActivityMapper = new Model_Mapper_FlagshipActivity ();
    //Hello
```

```
    if($privilege=='submitflagshipactivity'){
        if( $_user->role == 'admin')
            return true;
```

```
        $_flagshipActivityId=$_request->getParam (
'flagship_activity_id', 0 );
        if($_flagshipActivityId==0 || $_flagshipActivityId=="")
            return false;
```

```
        $_flagshipActivity = $_flagshipActivityMapper->fetchOne (
array (
            'flagship_activity_id' => $_flagshipActivityId
        ));
        if ($_flagshipActivity->focalpoint_id == $_user->user_id)
            return true;
```

```
        return false;
```

```
    }
```

```
    if ($_id == 0 && $_aid == 0)
        return true;
```

```
    if ($_id != 0)
        $_activityId = $_id;
```

```
    if ($_aid != 0)
        $_activityId = $_aid;
```

```

        try {
            $_flagshipActivity = $_flagshipActivityMapper->fetchOne (
                array (
                    'flagship_activity_id' => $_activityId
                ));
            if ($_flagshipActivity->focalpoint_id == $_user-
                >user_id || $_user->role == 'admin')
                return true;
        } catch ( Exception $e ) {
        }
        return false;
    }
}

```

Once an assertion class is available (App_Acl_Assert_Planning), an instance of the assertion class (new App_Acl_Assert_Planning ()) must be supplied when assigning conditional rules. A rule that is created with an assertion only applies when the assertion method returns TRUE. This is done at (library/App/Acl/Acl.php) by this line:

```

$this->allow ( $user, $planningResource,null, new App_Acl_Assert_Planning () );

```

The above code creates a conditional allow Rule that allows the user Role to access the PlanningController located at (application/modules/default/controllers/PlanningController.php) on all it's actions (NULL), access is asserted for authenticated users with identity only (logged in) if the action name is submitflagshipactivityAction() then admin Role is granted access, or if the user Role is a focal point.

As a result, the NULL value provided at the assertion Rule is overridden by the code inside the assertion interface, so only the following action is permitted by this assertion: protected function submitflagshipactivityAction()

- The \$selfassessmentResource assertion interface is located at: (library/App/Acl/Assert/Selfassessment.php), and has the following code inside:

```

class App_Acl_Assert_SelfAssessment implements App_Acl_Assert_Interface {
    public function assert(Zend_Acl $acl, Zend_Acl_Role_Interface $role = null,
        Zend_Acl_Resource_Interface $resource = null, $privilege = null) {
        $_auth = Zend_Auth::getInstance ();
        if (! $_auth->hasIdentity ()) {
            return false;
        }
        $_user = $_auth->getIdentity ();
        if (! $acl->hasValue ( 'request' ))
            return true;

        $_request = $acl->getContextValue ( 'request' );
        $_id = $_request->getParam ( 'id', 0 );
        $_pid = $_request->getParam ( 'project_id', 0 );
    }
}

```

```

$_projectMapper = new Model_Mapper_Project ();

if ($_id == 0 && $_pid == 0)
    return true;

if ($_id != 0)
    $_projectId = $_id;

if ($_pid != 0)
    $_projectId = $_pid;

try {

    $_projectEntity = $_projectMapper->fetchOne ( array (
        'project_id' => $_projectId
    ));
    if ($_projectEntity->project_manager_id == $_user->user_id ||
$_user->role == 'admin')
        return true;
    } catch ( Exception $e ) {
    }
    return false;
}
}

```

Once an assertion class is available (App_Acl_Assert_Selfassessment), an instance of the assertion class (new App_Acl_Assert_Selfassessment ()) must be supplied when assigning conditional rules. A rule that is created with an assertion only applies when the assertion method returns TRUE. This is done at (library/App/Acl/Acl.php) by this line:

```

$this->allow ( $user, $selfassessmentResource,null, new
App_Acl_Assert_SelfAssessment () );

```

The above code creates a conditional allow Rule that allows the user Role to access the SelfassessmentController located at: (application/modules/default/controllers/SelfassessmentController.php) on all its actions (NULL), access is asserted for authenticated users with identity only (logged in) if the Role is an admin, or if the user Role is assigned as a project manager.

- **MEL Custom Rule**

MEL has this custom Rule that takes the ProjectController as its Resource, but doesn't take any of its actions as a Privilege:

```

$this->allow ( $guest, $projectResource, array ('edit_check'), new
App_Acl_Assert_Project());

```

This custom Rule is used twice at: (application/modules/default/controllers/ProjectController.php).

This snippet of code is using the `isAllowed()` method to check the current Role status against the ACL, it's sending 'guest' as a role but the current Role will never be a guest.

This code is creating a new ACL by instantiating `App_Acl_Acl()`, checking if the Rule is allowed, if allowed it will get the Role from the `Zend_Auth`, if no Role is detected then it will redirect to user login page.

```

$_acl=new App_Acl_Acl();
    $_acl->setContextValue('project_id',$_projectEntity-
>project_manager_id);
        if(!$acl->isAllowed('guest','project','edit_check')){
            $_auth = Zend_Auth::getInstance();
            if($_auth->hasIdentity())
                return $this->forward('denied','user');
            else
                return $this->forward('login','user');
        }
    }

```

- **MEL Custom Rule Assertion**

The `$projectResource` assertion interface is located at `(library/App/Acl/Assert/Project.php)`, and has the following code inside:

```

class App_Acl_Assert_Project implements App_Acl_Assert_Interface
{
    public function assert(Zend_Acl $acl,
        Zend_Acl_Role_Interface $role = null,
        Zend_Acl_Resource_Interface $resource = null,
        $privilege = null)
    {
        $_auth = Zend_Auth::getInstance();
        if (!$auth->hasIdentity()) {
            return false;
        }
        $_user = $_auth->getIdentity();

        if($acl->getContextValue('project_id')==$_user->user_id || $_user-
>role=='admin')
            return true;

        return false;
    }
}

```

Once an assertion class is available (`App_Acl_Assert_Project`), an instance of the assertion class (`new App_Acl_Assert_Project ()`) must be supplied when assigning conditional rules. A rule that is created with an assertion only applies when the assertion method returns TRUE. This is done at `(library/App/Acl/Acl.php)` by this line:



```
$this->allow ( $guest, $projectResource, array ('edit_check'), new  
App_Acl_Assert_Project());
```

The above code creates a conditional allow Rule that allows the guest Role (Please read MEL Custom Rule section for more clarification) to access the ProjectController located at: (application/modules/default/controllers/ProjectController.php) on all its edit_check action (Not a real action), access is asserted for authenticated users with identity only (logged in) if the Role is an admin, or if the user Role is assigned to that project.

7.5.3 System Layout (Zend_Layout)

MEL system uses Zend_Layout component to manages rendering of a master layout script, which contains content placeholders for embedding content generated by actions or other view scripts. As with all other Zend framework components, Zend_Layout works with minimum configuration for most use cases, but if the requirements are more specialized, it is very flexible.

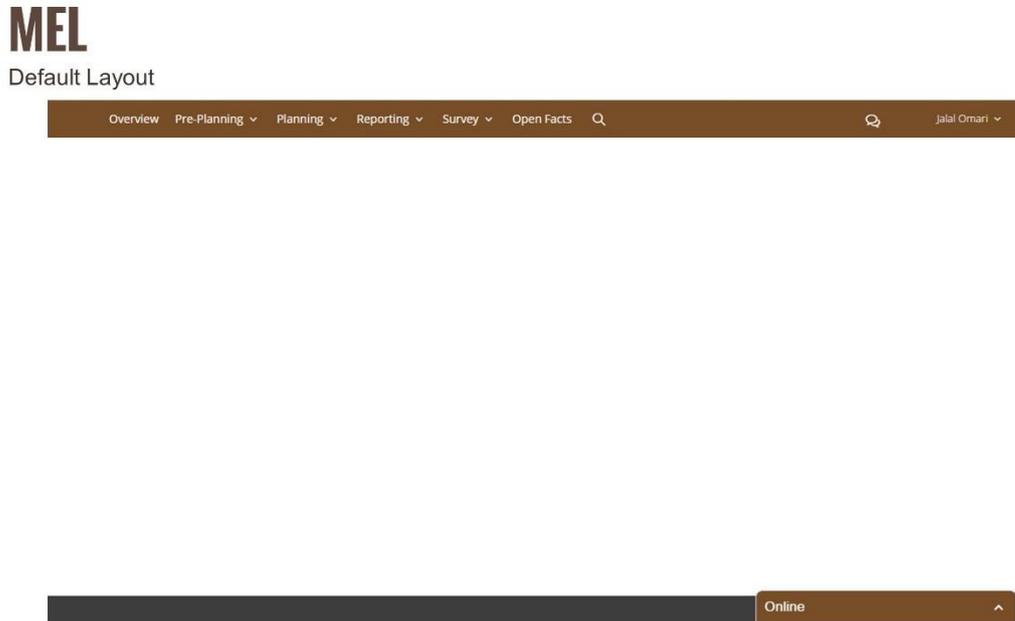
When using Zend_Layout with the MVC components, the startMvc() method is used to initialize it. This is done in the bootstrap file like this:

```
Zend_Layout::startMvc (array ('layoutPath' => '/path/to/layouts'));
```

Behind the scenes, startMvc() creates a Singleton instance of Zend_Layout and registers a front controller plug-in and an action helper that can be interfaced with from the rest of the application. The front controller plug-in, Zend_Layout_Controller_Plugin_Layout, has a postDispatch() hook function that renders the layout template at the end of the last dispatched action. The action helper, Zend_Layout_Controller_Action_Helper_Layout, is used to provide easy access to the Zend_Layout object from within a controller.

The main layout for the system is **default.phtml** (`/application/views/layouts/default.phtml`) serves as the holder for all the content to be displayed, so it only contains the fundamental structure of the page design and then delegates the rest of the content to other files. By default, **default.phtml** is rendered by default by Zend_Layout.

Figure 71. MEL System Default Layout



The structure of the file consists of two sections (header and body) as follow:

- At the beginning of the file there is php code to check the signed in user is registered in Zend_Registry scope (Which is a container for storing objects and values in the application space. By storing the value in the registry, the same object is always available throughout the application. This mechanism is an alternative to using global storage), html code, metronic and CSS files for styling plus Zend head view helpers:

```
<?php
$_menuShowAdmin=false;
$_menuShowUser=false;
$_menuShowGenderSurvey = false;
$_menuShowIpSurvey = false;
if(Zend_Registry::get('user')!=null){
    $_user=Zend_Registry::get('user');
    if($_user->role=='admin')
        $_menuShowAdmin=true;
    else
        $_menuShowUser=true;

    $_menuShowGenderSurvey = false;
    $_menuShowIpSurvey = false;
    if($_user->isMemberOf(5) || $_user->isMemberOf(6)){
        $_menuShowGenderSurvey = true;
    }
    if($_user->isMemberOf(11)){
        $_menuShowIpSurvey = true;
    }
}
?>
```

The above PHP script use Zend_Registry::get('user') to check if there is user object registered and assign to \$_user variable then check the role of the user and according to that change the variable for showing the designated menu

- Inside head html tag the normal html tags included in addition to Zend view helpers:
 - headMeta() is used to set all the <meta> tags in the <head> section of the page. There are two types of meta tags: name and http-equiv, so there are two sets of functions, as shown in the following table:

Table 17. headMeta() types

Name Version	Http-equiv version
appendName(\$keyValue, \$content, \$conditionalName)	appendHttpEquiv(\$keyValue, \$content, \$conditionalHttpEquiv)
prependName(\$keyValue, \$content, \$conditionalName)	prependHttpEquiv(\$keyValue, \$content, \$conditionalHttpEquiv)
setName(\$keyValue, \$content, \$modifiers)	setHttpEquiv (\$keyValue, \$content, \$modifiers)
offsetSetName (\$index, \$keyValue, \$content, \$conditionalName)	offsetSetHttpEquiv(\$index, \$keyValue, \$content, \$conditionalHttpEquiv)

- **\$keyValue** field sets either the name or the http-equiv key for the tag. The **\$content** parameter is used for the value attribute of a name tag or the content attribute of an http-equiv tag, and the **\$modifiers** parameter is an associative array that can contain the lang and scheme attributes if required
- **headLink()** manages <link> elements in the <head> section of the document. This includes CSS stylesheets, favicons, RSS feeds, and trackbacks. It aggregates the elements together while rendering each view script and is later used to render the elements into the layout.
- **placeholder()** is a generic helper to aggregate content and render it in custom ways, here in the head section "<?=\$this->placeholder('styleVars'); ?>" loads different actions custom inline styles.
- The <body> tag contains the visible page of the action with the default html layout: (the explanation for the tags inside html comments)



Monitoring, Evaluation and Learning Platform: System Design & Architecture

- The footer section inside the <body> tag which consists of plain html code and the discussion form including javascript libraries:



Monitoring, Evaluation and Learning Platform: System Design & Architecture

```
<?php if(!$this->hideLayoutElements):?> <!-- Check if the footer html section set to not be hidden
so it can be displayed, to set hideLayoutElements value from the controller with this line of code
$this-> helper->layout()->disableLayout(); -->
<!-- BEGIN FOOTER -->
<div class="page-footer">
  <div class="page-footer-inner">
  </div>
  <div class="page-footer-tools">
    <span class="go-top">
      <i class="fa fa-angle-up"></i>
    </span>
  </div>
</div>
<?php endif;?>
<div id="discussion_modal" class="modal container fade" tabindex="-1">
  <div class="modal-header">
    <button type="button" class="close" data-dismiss="modal" aria-hidden="true"></button>
    <h4 class="modal-title">New Discussion</h4>
  </div>
  <div class="modal-body">
    <div class="row">
      <div class="col-md-12 form">
        <form action="<?=$this->url(array('module'=>'default','controller'=>'discussion','action'=>'startdiscussion'),null,true)?>
" id="start_discussion_form" method="post" class="form-horizontal"> <!-- Output live chatting
action url to forward input fields value -->
.
.
.
.
<?=$this->placeholder('modals');?> <!-- Output modals placeholder values set by the different
script views -->
<!-- Google Analytics Javascript Code -->
<script>
  (function(i,s,o,g,r,a,m){i['GoogleAnalyticsObject']=r;i[r]=i[r]||function(){
  (i[r].q=i[r].q||[]).push(arguments)},i[r].l=1*new Date();a=s.createElement(o),
  m=s.getElementsByTagName(o)[0];a.async=1;a.src=g;m.parentNode.insertBefore(a,m)
  })(window,document,'script','//www.google-analytics.com/analytics.js','ga');

  ga('create','UA-65705913-1','auto');
  ga('send','pageview');

</script>
<script>
base_url = "<?=$this->baseurl()?>";
var discussionsHomeLink = '<?=$this->url(array('module'=>'default','controller'=>'discussion','action'=>'index'),null,true)?>';
var userDataLink = '<?=$this->url(array('module'=>'default','controller'=>'user','action'=>'getallusers'),null,true)?>';
var discussionCountLink = '<?=$this->url(array('module'=>'default','controller'=>'discussion','action'=>'discussionscount'),null,true)?>';
>;
</script>
<!-- END FOOTER -->
<!-- Load javascripts at bottom, this will reduce page load time -->
.
.
.
<!-- Append javascript files and plugins -->
<?php echo $this->headScript(); ?> <!-- Print all the script files appended to headScript() view
helper -->
<script>
<?=$this->placeholder('jsvars');?> <!-- Append all inline javascripts set by script views -->
  jQuery(document).ready(function() {
    Metronic.init();
    Layout.init();
    LayoutCustom.init();
    DiscussionForm.init();
    <?=$this->placeholder('jscalls');?> <!-- Append all inline javascripts set by script views -->
```

```

.
.
.
<!-- END JAVASCRIPTS -->
</body>
<!-- END BODY -->
</html>

```

- **headScript()** is used to manage JavaScript files. This allows to add the relevant files to the helper as the views are rendered.
- **baseUrl()**: In order to create the correct path to the files (stylesheets, javascripts, images and links) the root URL referred as (**\$this->baseUrl()**)
- **url()**: The url() view helper creates URL strings based on a named route. The method signature is:

Table 18. url function parameters

Parameter	Description
\$urlOptions	Array of options that are used to create the URL string.
\$name	Name of the route to use to create the URL string. If null, the route name that originally matched the current page's URL is used.
\$reset	Set to true to reset all the parameters when creating the URL string.
\$encode	Set to true to urlencode() all the parameter values in \$urlOptions.

View Helpers

The layout uses two types of view helpers to perform complex functions over and over: e.g., formatting a date, generating form elements, or displaying action links, without copying the code for a specific helper in every view page

- Zend view helpers

Zend Framework provides a set of helpers to manage the <head> section of an HTML page. These helpers set up the information in advance and output it within the layout view script. There are a variety of helpers that mostly begin with the word **head*** in the **<head>** section.

- headMeta()
- headLink()
- headTitle()
- headScript()

- Custom View Helpers

Custom view helper is a simple class prefixed with 'Zend_View_Helper_' and the last segment of the class name is the helper name; this segment should be TitleCapped and the helpers folder (*/application/views/helpers/*.php*)

- BreadCrumb.php: this helper adds breadcrumb view when used in the view script.

```

<?php
class Zend_View_Helper_BreadCrumb {
    private $_separator='>';
    public function breadCrumb() {
        $Navigation = App_Navigation::getInstance ();
        $i=0;
        $links='';
        foreach ( $Navigation as $link => $text ) {
            if($i==0 && count($Navigation)!=1)
                $links.= '<li><i class="fa fa-home"></i><a href="'. $link . '>' . $text.'</a><i class="fa fa-angle-right"></i></li>';
            else if ($i==0 && count($Navigation)==1)
                $links.= '<li><i class="fa fa-home"></i>'. $text.'</li>';
            else if ($i!=$Navigation->count()-1)
                // $links.= '<li>'. $text.'<i class="fa fa-angle-right"></i></li>';
                $links.= '<li><a href="'.str_replace('/mel/mel','/mel',$link) . '>' . $text.'</a><i class="fa fa-angle-right"></i></li>';
            else
                $links.= '<li>'. $text.'</li>';
            $i++;
        }
        return $links;
    }
}

```

In the public function breadCrumb() use the singleton class App_Navigation through its static method getInstance() that managing trees of pointers to web pages. Rendering of this class is done through helper like this `<?php echo $this->breadCrumb();?>` Note: breadcrumb is called in the main layout (default.phtml) but it's not displayed because its inside html comment tag `<!-- ... -->`

- DisplayNumber.phtml: this helper rounds numbers to two digits and adds the unit "B", "K", "M" according to the number value.

```

<?php
class Zend_View_Helper_DisplayNumber {
    public function displayNumber($number) {
        if($number>=1000000000)
            return round($number/1000000000,2) ."B";
        if($number>=1000000)
            return round($number/1000000,2) ."M";
        if($number>=1000)
            return round($number/1000,2) ."K";
    }
}

```

It can be used as `<?= displayNumber($anyNumber); ?>`

- FullURL.php: returns the complete URL in the view for chosen element (link, image, file, path).

```

<?php
class Zend_View_Helper_FullUrl extends Zend_View_Helper_Abstract {
    public function fullUrl($url='') {
        $request = Zend_Controller_Front::getInstance ()->getRequest ();
        $url = $request->getScheme () . "://" . $request->getHttpHost () . $url;
        return $url;
    }
}

```

The public function `fullUrl($url=)` gets the request instance from the Front Controller which includes the route of the page then gets the protocol and the hostname part of the path and return the complete URL. For example:

```
<a href="<?=$this->fullUrl($this->url(array('module'=>'default', 'controller'=>'gender', 'action'=>'index', 'code'=>$this->accesscode), null, true)) ?>">link here</a>
```

- `GenerateUserPopover.php`: php script to load bootstrap plugin for on-place confirm boxes using Popover.

```
<?php
class Zend_View_Helper_GenerateUserPopover extends Zend_View_Helper_Abstract {
    public function generateUserPopover($userEntity, $placement='top', $subClass='') {
        if(strip_tags($userEntity->bio)==' && $userEntity->photo=='')
            return '<a href="#" class="'. $subClass.'">'. $userEntity->name. '</a>';
        if($userEntity->photo !='')
            $photo="<img src='". $this->view->baseUrl() ."/uploads". $userEntity->photo.'" class='BioImage'
            style='width:150px;' />";
        else
            $photo='';
        return '<a href="#" class="popovers ' . $subClass.'" data-original-title="'. $userEntity->name.'"
        data-container="body" data-html="true" data-content="'. $photo.str_replace('"', '\'', $userEntity->
        bio) .'" data-placement="'. $placement.'" data-trigger="hover">' . $userEntity->name. '</a>';
    }
}
```

Action Helpers

Zend_Controller_Action_HelperBroker::getStaticHelper('FlashMessenger')->getMessages()

extends `Zend_Controller_Action` to handle the details of registering helper objects and helper paths, as well as retrieving helpers on-demand.

Zend Framework includes several action helpers by default: **AutoComplete** for automating responses for AJAX autocompletion; **ContextSwitch** and **AjaxContext** for serving alternate response formats for the actions; a **FlashMessenger** for handling session flash messages; **Json** for encoding and sending JSON responses; a **Redirector**, to provide different implementations for redirecting to internal and external pages from your application; and a **ViewRenderer** to automate the process of setting up the view object in the controllers and rendering views.

MEL system default layout use two action helpers:

- The **FlashMessenger** helper which allows to pass messages that the user may need to see on the next request. To accomplish this, `FlashMessenger` uses `Zend_Session_Namespace` to store messages for future or next request retrieval. `Zend_Session::start()` used in the bootstrap file so the different components of the system can add to the session and read from the session.

Available Methods in FlashMessenger

General methods:

- `setNamespace($namespace='default')` is used to set the namespace into which messages are stored by default.
 - `getNamespace()` is used to retrieve the name of the default namespace. The default namespace is 'default'.
 - `resetNamespace()` is used to reset the namespace name to the default value, 'default'.
- Methods for manipulating messages set in the previous request:

- `hasMessages($namespace=NULL)` is used to determine if messages have been carried from a previous request by the flash messenger. The optional argument `$namespace` specifies which namespace to look in. If the `$namespace` argument is omitted, the value returned by `getNamespace()` will be used.
- `getMessages($namespace=NULL)` is used to retrieve the messages which have been carried from a previous request by the flash messenger. The optional argument `$namespace` specifies which namespace to pull from. If the `$namespace` argument is omitted, the value returned by `getNamespace()` will be used.
- `getIterator($namespace=NULL)` wraps the return value of `getMessages()` in an instance of `ArrayObject`. If the `$namespace` argument is omitted, the value returned by `getNamespace()` will be used.
- `count($namespace=NULL)` returns the number of messages contained in the specified namespace. If the `$namespace` argument is omitted, the value returned by `getNamespace()` will be used.
- `clearMessages($namespace=NULL)` is used to clear all the messages which have been carried from a previous request by the flash messenger. The optional argument `$namespace` specifies which namespace to clear out. If the `$namespace` argument is omitted, the value returned by `getNamespace()` will be used.

Methods for manipulating messages set in the current request:

- `addMessage($message, $namespace=NULL)` is used to add a new message to the current request. `$message` contains the message to be added, and the optional argument `$namespace` will specify the namespace. If the `$namespace` argument is omitted, the value returned by `getNamespace()` will be used.
- `hasCurrentMessages($namespace=NULL)` is used to determine if messages have been added to the flash messenger during the current request. The optional argument `$namespace` specifies which namespace to look in. If the `$namespace` argument is omitted, the value returned by `getNamespace()` will be used.
- `getCurrentMessages($namespace=NULL)` is used to retrieve the messages which have been added to the flash messenger during the current request. The optional argument `$namespace` specifies which namespace to pull from. If the `$namespace` argument is omitted, the value returned by `getNamespace()` will be used.
- `clearCurrentMessages($namespace=NULL)` is used to clear all the messages which have been added to the flash messenger during the current request. The optional argument `$namespace` specifies which namespace to clear out. If the `$namespace` argument is omitted, the value returned by `getNamespace()` will be used.
- `ViewRenderer` is the second action helper used in the system to satisfy the following goals:
 - Eliminate the need to instantiate view objects within controllers; view objects will be automatically registered with the controller.
 - Automatically set view script, helper, and filter paths based on the default module (default module that is used in MEL system), and automatically associate the current module name as a class prefix for helper and filter classes.
 - Create a globally available view object for all dispatched controllers and actions.
 - Set default view rendering options for all controllers.
 - Add the ability to automatically render a view script with no intervention.



Monitoring, Evaluation and Learning Platform: System Design & Architecture

- Create different specifications for the view base path and for view script paths.

HTML Elements

The default.phtml layout is using metronic v3.5.0 which is based on bootstrap v3.3.6 that make MEL system use fluid grid system that appropriately scales up to 12 columns as the device or viewport size increases. It includes predefined classes for easy layout options.

MEL Grid systems create page layouts through a series of rows and columns that house the contents. The grid system works as following:

- Rows must be placed within a container div html element
- Use rows to create horizontal groups of columns.
- Content should be placed within columns, and only columns may be immediate children of rows.
- Predefined grid classes like .row and .col-xs-4 are available for quickly making grid layouts.
- If more than 12 columns are placed within a single row, each group of extra columns will, as one unit, wrap onto a new line.

Menu HTML Component:

Figure 72. Horizontal Menu

MEL
Horizontal Menu



The code for the horizontal menu consists of:



```
<!-- BEGIN HORIZONTAL MENU -->
<!-- Place "hor-menu-light" class after the "hor-menu" class to have a horizontal menu
with white background -->
<div class="hor-menu hor-menu-light hidden-sm hidden-xs">
  <ul class="nav navbar-nav">
    <!-- To enable the horizontal opening on mouse hover for the horizontal the
following html attributes need to be placed in the list items
- data-hover="dropdown"
- data-close-others="true"
-->
      <li class="classic-menu-dropdown">
        <a href="<?=> $this->url(array('module' => 'default', 'controller' =>
'overview', 'action' => 'index'), null, true) ?>">Overview
        </a>
      </li>
      <?php if ($_menuShowUser): ?>
      <li class="classic-menu-dropdown">
        <a href="<?=> $this->url(array('module' => 'default',
'controller' => 'user', 'action' => 'dashboard'), null, true) ?>">Dashboard
        </a>
      </li>
      <?php endif; ?>
      <?php if ($_menuShowAdmin): ?>
      <li class="classic-menu-dropdown">
        <a data-hover="dropdown" data-close-others="true" data-
toggle="dropdown" href="javascript:;">Pre-Planning
          <i class="fa fa-angle-down"></i>
        </a>
        <ul class="dropdown-menu pull-left">
          <li>
            <a href="<?=> $this->url(array('module' => 'default',
'controller' => 'preplanning', 'action' => 'als'), null, true) ?>">ALS</a>
            <a href="<?=> $this->url(array('module' => 'default',
'controller' => 'preplanning', 'action' => 'actionsites'), null, true) ?>">Action
              Sites</a>
            <a href="<?=> $this->url(array('module' => 'default',
'controller' => 'preplanning', 'action' => 'flagships'), null, true) ?>">Flagships</a>
            <a href="<?=> $this->url(array('module' => 'default',
'controller' => 'preplanning', 'action' => 'ido'), null, true) ?>">IDO</a>
            <!-- <a href="<?=> $this->url(array('module' =>
'default', 'controller' => 'preplanning', 'action' => 'activities'), null, true)
?>">General Activities</a> -->
            <!-- <a href="<?=> $this->url(array('module' =>
'default', 'controller' => 'preplanning', 'action' => 'organization'), null, true)
?>">Organizations</a> -->
            <a href="<?=> $this->url(array('module' => 'default',
'controller' => 'preplanning', 'action' => 'partners'), null, true) ?>">Partners</a>
            <a href="<?=> $this->url(array('module' => 'default',
'controller' => 'user', 'action' => 'index'), null, true) ?>">Users</a>
          </li>
        </ul>
      </li>
      <li class="classic-menu-dropdown">
        <a data-hover="dropdown" data-close-others="true" data-
toggle="dropdown" href="javascript:;">Planning
          <i class="fa fa-angle-down"></i>
        </a>
        <ul class="dropdown-menu pull-left">
          <li>
            <a href="<?=> $this->url(array('module' => 'default',
'controller' => 'planning', 'action' => 'index'), null, true) ?>">Activities </a>
            <a href="<?=> $this->url(array('module' => 'default',
```



Monitoring, Evaluation and Learning Platform: System Design & Architecture

```
'controller' => 'project', 'action' => 'agreements'), null, true) ?>">Project
    Agreements </a>
    <a href="<?=$this->url(array('module' => 'default', 'controller' =>
'project', 'action' => 'index'), null, true) ?>">Projects </a>
  </li>
</ul>
</li>
<li class="classic-menu-dropdown">
  <a data-hover="dropdown" data-close-others="true" data-toggle="dropdown"
href="javascript:;">Reporting
    <i class="fa fa-angle-down"></i>
  </a>
  <ul class="dropdown-menu pull-left">
    <li>
      <a href="<?=$this->url(array('module' => 'default', 'controller' =>
'reporting', 'action' => 'index'), null, true) ?>">Activities </a>
      <a href="<?=$this->url(array('module' => 'default', 'controller' =>
'reporting', 'action' => 'index'), null, true) ?>">Projects </a>
    </li>
  </ul>
</li>
<?php endif; ?>
<!-- According to the user group and role the Survey menu will be displayed -->
<?php if (($_menuShowGenderSurvey && $_menuShowIpSurvey) || $_menuShowAdmin): ?>
  <li class="classic-menu-dropdown">
    <a data-hover="dropdown" data-close-others="true" data-toggle="dropdown"
href="javascript:;">Survey
      <i class="fa fa-angle-down"></i>
    </a>
    <ul class="dropdown-menu pull-left">
      <li>
        <a href="<?=$this->url(array('module' => 'default', 'controller' =>
'gender', 'action' => 'report'), null, true) ?>">Gender</a>
        <a href="<?=$this->url(array('module' => 'default', 'controller' =>
'ip', 'action' => 'report'), null, true) ?>">Intellectual property</a>
      </li>
    </ul>
  </li>
  <?php elseif ($_menuShowGenderSurvey): ?>
    <li class="classic-menu-dropdown">
      <a href="<?=$this->url(array('module' => 'default', 'controller' =>
'gender', 'action' => 'report'), null, true) ?>">Gender
        survey
      </a>
    </li>
  <?php elseif ($_menuShowIpSurvey): ?>
    <li class="classic-menu-dropdown">
      <a href="<?=$this->url(array('module' => 'default', 'controller' => 'ip',
'action' => 'report'), null, true) ?>">Intellectual
        property survey
      </a>
    </li>
  <?php endif; ?>
  <li class="classic-menu-dropdown">
    <a href="<?=$this->url(array('module' => 'default', 'controller' =>
'dataanalysis', 'action' => 'index'), null, true) ?>">Open Facts</a>
  </li>
</ul>
</div>
<!-- END HORIZONTAL MENU -->
```

Search Box Html Component:

Figure 73. Search Box

MEL
Search Box



The search box is used to search the system projects for different keywords.

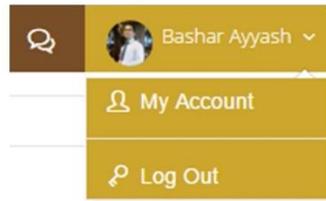
```
<!-- BEGIN HEADER SEARCH BOX -->
<!-- Apply "search-form-expanded" right after the "search-form" class to have half
expanded search box -->
<form class="search-form" action="#" method="GET">
  <div class="input-group">
    <input type="text" class="form-control" placeholder="Search..." name="query">
    <span class="input-group-btn">
      <a href="javascript:;" class="btn submit"><i class="icon-magnifier"></i></a>
    </span>
  </div>
</form>
<!-- END HEADER SEARCH BOX -->
```

User Menu Html Component:

It lists the user account and log out menu items.

Figure 74. User Menu Top Navigation Menu

MEL
User Menu





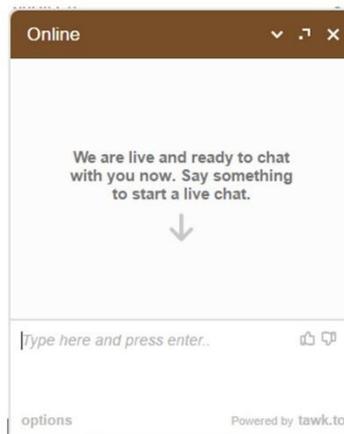
```
<!-- BEGIN TOP NAVIGATION MENU -->
<?php if(Zend_Registry::get('user')!=null):?>
<?php $_unread=Zend_Registry::get('user')->unread_messages?>
<div class="top-menu">
  <ul class="nav navbar-nav pull-right">
    <li class="dropdown dropdown-extended dropdown-inbox" id="header_inbox_bar">
      <a href="<?=$this->url(array('action'=>'index',
'controller'=>'discussion'),null,false)?>" class="dropdown-toggle" style="padding-
right:10px;">
        <i class="icon-bubbles"></i>
        <span class="badge badge-danger<?php if($_unread!=0):?> hidden<?php endif;?>"
id="chat_bubble" style="top:5px; right:25px;"><?=$_unread?></span>
      </a>
    </li>
    <!-- BEGIN USER LOGIN DROPDOWN -->
    <li class="dropdown dropdown-user">
      <a href="#" class="dropdown-toggle" data-toggle="dropdown" data-hover="dropdown"
data-close-others="true">
        <?php if(Zend_Registry::get('user')->photo!=''):?>
        <span style="background-image: url('<?=$this-
>baseurl();?>/uploads/<?=$Zend_Registry::get('user')->photo?>');"
class="avatar"></span>
        <?php else:??>
        
        <?php endif;??>
        <span class="username">
          <?=$Zend_Registry::get('user')->name?> </span> <!-- Print signed in user registered
in Registry scope -->
        <i class="fa fa-angle-down"></i>
      </a>
      <div class="clearfix"></div>
      <ul class="dropdown-menu">
        <li>
          <a href="<?=$this->url(array('action'=>'dashboard',
'controller'=>'user'),null,false)?>#tab_1_2">
            <i class="icon-user"></i> My Account </a>
          </li>
          <li class="divider">
          </li>
          <li>
            <a href="<?=$this->url(array('action'=>'logout',
'controller'=>'user'),null,false)?>">
              <i class="icon-key"></i> Log Out </a>
            </li>
          </ul>
        </li>
    <!-- END USER LOGIN DROPDOWN -->
  </ul>
</div>
<?php else:??>
<div class="hor-menu hor-menu-light pull-right">
  <ul class="nav navbar-nav pull-right">
    <li class="classic-menu-dropdown">
      <a href="<?=$this-
>url(array('module'=>'default','controller'=>'user','action'=>'login'),null,true)?>">L
ogin
      </a>
    </li>
  </ul>
</div>
<?php endif;??>
<!-- END TOP NAVIGATION MENU -->
```

Discussion Html Component:

Figure 75. Discussion Live Box

MEL

Discussion Live Chat



Is a discussion popup stick to the right bottom corner of the screen, enables live chatting between users. It uses a third-party javascript library "Tawk.to"

Figure 76. HTML Footer



```
<?php if(!isset($this->hideLayoutElements)) :?>
<!-- BEGIN FOOTER -->
<div class="page-footer">
  <div class="page-footer-inner">
  </div>
  <div class="page-footer-tools">
    <span class="go-top">
      <i class="fa fa-angle-up"></i>
    </span>
  </div>
</div>
<?php endif;?>
```

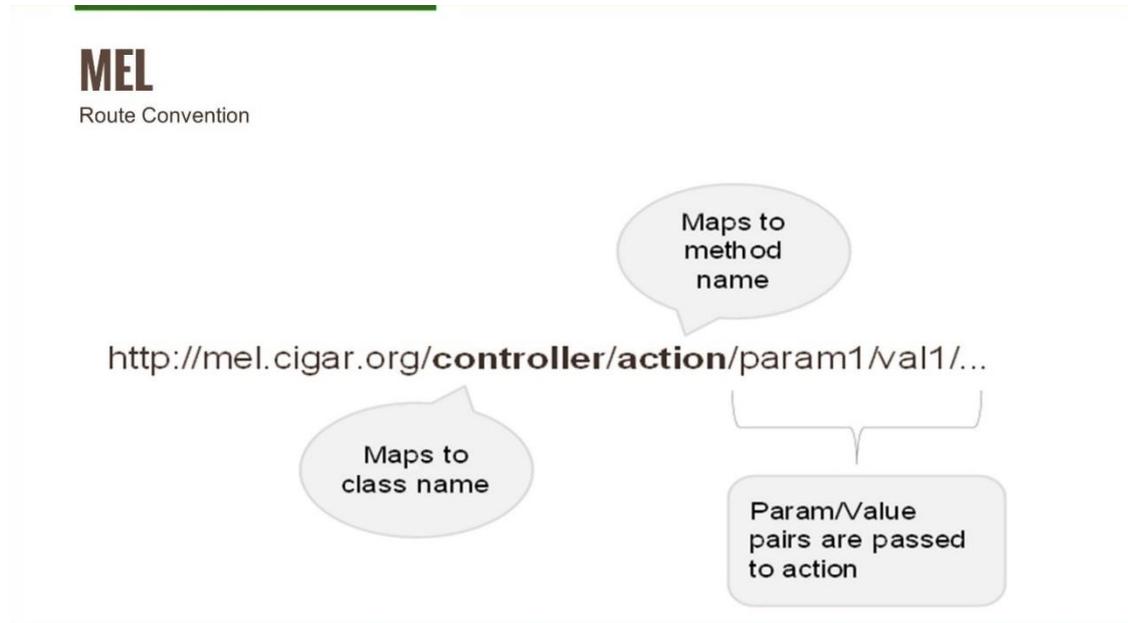
Managing Views

Zend Framework provides a suite of components (zend_View, system helpers, output filters, variable escaping) that help make the visual part of MEL system both powerful and flexible and also easy to maintain in the long term.

Routing Cycle

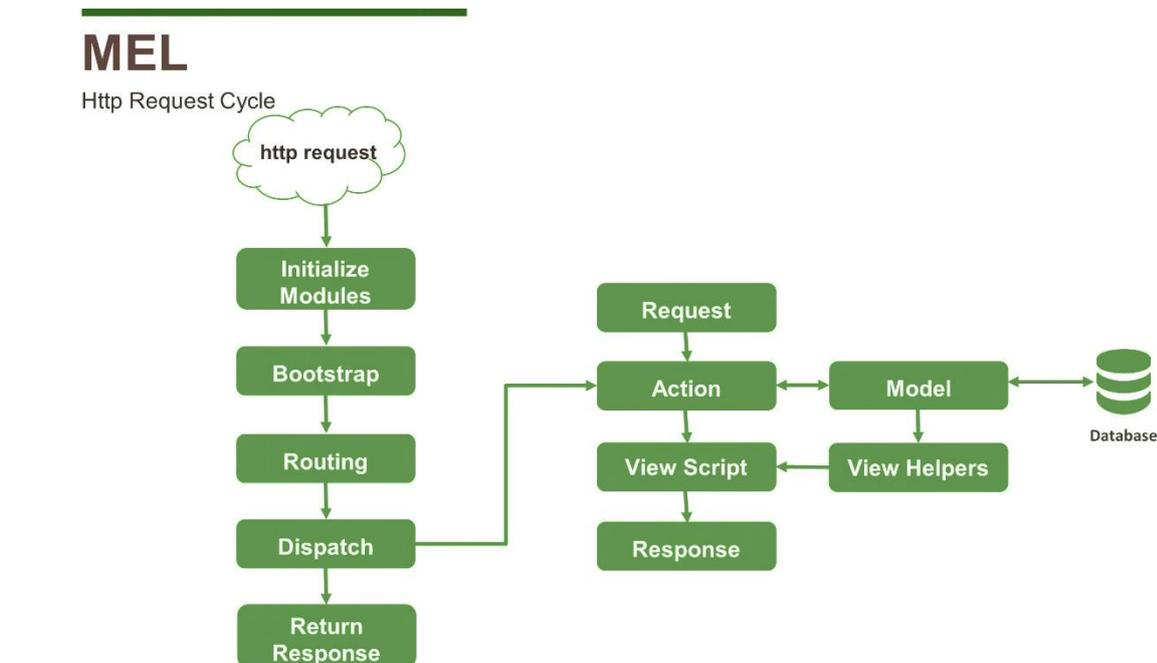
View pages in MEL system are loaded through URL routing which indicates the controller and the action that is required to run, if the action is not stated in the URL the system will load the default action (indexAction()).

Figure 77. Route Convention



MEL system uses Zend Front Controller design pattern. This means that all requests are directed to a single point public/index.php file. This is done using a .htaccess file containing rewrite rules that serves all static files (such as CSS & Javascript) and directs all other requests to the index.php. The index.php file initializes the loader and then bootstraps the application before finally return the response to the browser. The process looks like the following graph:

Figure 78. Request Cycle through Zend Application



When the controller assign variables and called `render()`, `Zend_View`, it requests the view script and executes it "inside" the scope of the `Zend_View` instance in the view script. To manipulate or

render the variables variables in the view script `$this->variable_name`, as `$this` points to the `Zend_View` instance itself. The **Zend_View** class keeps the view portion of the application separate from the rest of the application. It provides helpers, output filters, and variable escaping.

Figure 79. Multiple templates are used to build up complete page

MEL
Page Templates



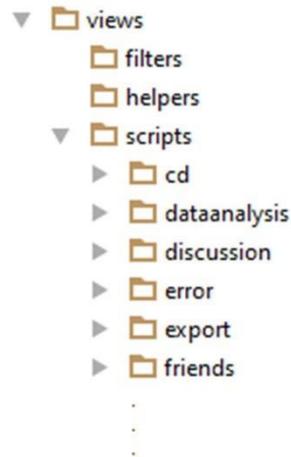
The master template (**default.phtml**) providing the overall layout and template containing the action specific content placed inside the placeholders, along with the **ViewRenderer** action helper and the **action()_view** helper, it provides a complete and very flexible display system.

The **ViewRenderer** expects that all action view scripts are stored in the **/views/scripts** subdirectory, they are further subdivided into separate subdirectories per controller, each subdirectory contains different view for each Zend Action.

Figure 80. View folder structure

MEL

View Folder Structure



Views append required files (javascript libraries, CSS stylesheets and images) for final rendered page from assets folder, the path of the assets files (`/application/assets/`)

The asset folder contains two groups of assets files:

1. Common assets group: metronic base files, theme, global scripts and custom general files from global subfolder.
2. Specific assets for each view loaded from pages subfolder

The assets files types are:

- Images
- Css files: for styling the view
- Scripts files: to manage the client side actions, the scripts use IIFE (Immediately-Invoked Function Expression) which is javascript design pattern to establish private methods for accessible functions and expose properties for later use:

```

/**
Custom module for you to write your own javascript functions
**/
var Custom = function () {

    // private functions & variables

    var myFunc = function(text) {
        alert(text);
    }

    // public functions
    return {

        //main function
        init: function () {
            //initialize here something.
        },

        //some helper function
        doSomeStuff: function () {
            myFunc();
        }

    };

}();

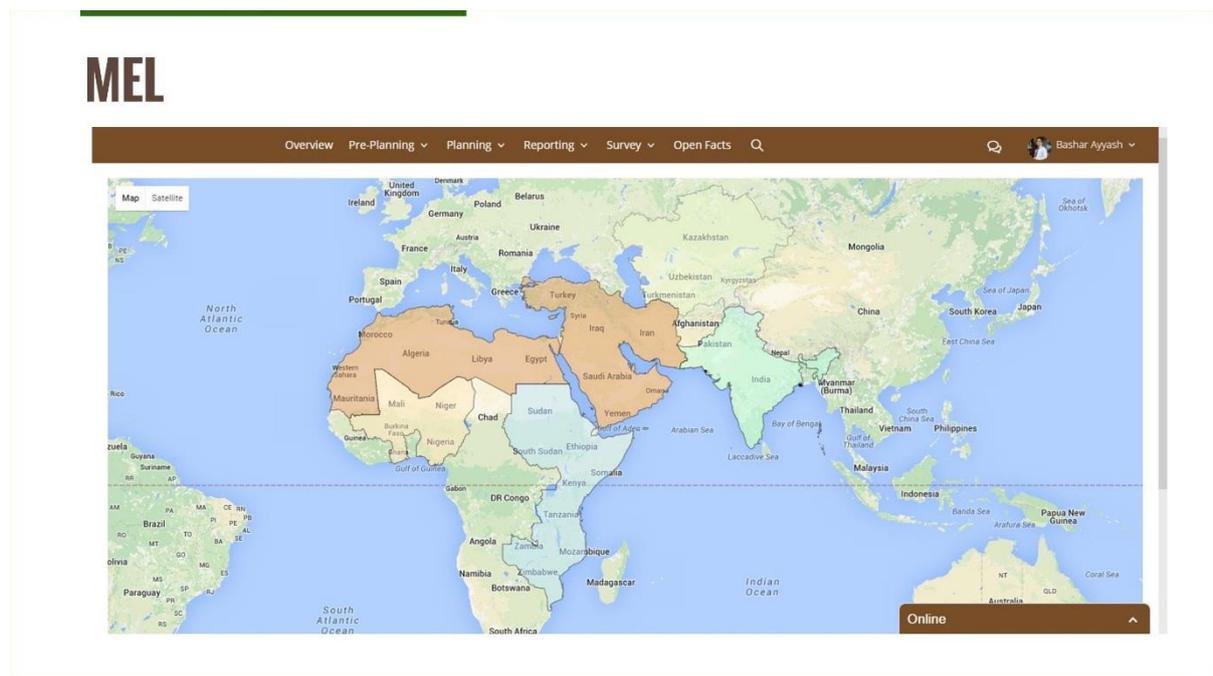
/**
Usage
***/
//Custom.init();
//Custom.doSomeStuff();

```

View Scripts

1. Overview subdirectory ([/application/modules/default/views/scripts/overview/*.phtml](http://application/modules/default/views/scripts/overview/*.phtml)):

Figure 81. Overview main page (<http://mel.cgiar.org/overview>)



The overview subfolder contains view for the google world map which shows different flagships details in different action sites **actionsite.phtml**: contains full overview of action site details, the page can be accessed through link on different pages or with direct URL (for ex. `/overview/actionsite/id/3`) and it fills the content of the page from **OverviewController.php** (`/application/modules/default/controllers/OverviewController.php`) which includes different actions and one of them is `actionsiteAction()`:

```
public function actionsiteAction()
{
    $_actionsiteId = $this->getRequest()->getParam('id', 0);
    App_Navigation::AddLink($this->view->url(array(
        'module' => 'default',
        'controller' => 'overview',
        'action' => 'actionsite'
    )), null, false), 'Action Site Overview');

    try {
        $_actionsiteMapper = new Model_Mapper_Actionsite();
        $_actionsiteEntity = $_actionsiteMapper->fetchOne(array('actionsite_id' =>
$_actionsiteId));

        $this->view->mainTitle = $_actionsiteEntity->name;
        $this->view->subTitle = 'Overview';
        $this->view->actionsite = $_actionsiteEntity;
    } catch (Exception $e) {
    }
}
```

The action function gets the actionsite data from `Model_Mapper_Actionsite()` after getting id url parameter, then assigns the result value to ZendView instance variables which will be rendered in the view script.

- **activity.phtml**: contains overview of the activity details, the page can be accessed through link on different pages or with direct URL (for ex. `/overview/activity/id/290`) and it renders the content of the page from **OverviewController.php** which includes different actions and one of them is `activityAction()`:

```

public function activityAction()
{
    $_activityId = $this->getRequest()->getParam('id', 0);
    App_Navigation::AddLink($this->view->url(array(
        'module' => 'default',
        'controller' => 'overview',
        'action' => 'actionsite'
    )), null, false), 'Action Site Overview');

    try {
        $_activityMapper = new Model_Mapper_FlagshipActivity();
        $_activityEntity = $_activityMapper->fetchOne(array('flagship_activity_id' =>
$_activityId));

        $this->view->mainTitle = 'Activity Overview';
        $this->view->subTitle = 'Overview';

        $this->view->flagshipActivity = $_activityEntity;

        $_projectYearMapper = new Model_Mapper_ProjectYear();
        $_projectYears = $_projectYearMapper->fetchMany();

        $this->view->projectYears = $_projectYears;

        $_partnershipLevelMapper = new Model_Mapper_PartnershipLevel();
        $_partnershipLevelCollection = $_partnershipLevelMapper->fetchMany();

        $this->view->partnershipLevels = $_partnershipLevelCollection;

        $_budgetSourceMapper = new Model_Mapper_BudgetSource();
        $_budgetSources = $_budgetSourceMapper->fetchMany();

        $this->view->budgetSources = $_budgetSources;

        $_idoMapper = new Model_Mapper_Ido();
        $_idos = $_idoMapper->fetchMany();

        $this->view->idos = $_idos;

    } catch (Exception $e) {
    }
}

```

The action function gets the data for activity view from:

Model_Mapper_FlagshipActivity(),
Model_Mapper_ProjectYear(),
Model_Mapper_PartnershipLevel(),
Model_Mapper_BudgetSource(),
Model_Mapper_Ido(),

sets collections variables after getting id url parameter and sets \$_activityEntity result to ZendView instance variable

- **flagship.phtml:** this view consist of Flagship Overview, Partners, Action sites, Projects, Target Countries, and IDO Indicators the page can be accessed through link on different pages or with direct URL (for ex. </overview/flagship/id/3>) and it fills the content of the page from **OverviewController.php** which includes different actions and one of them is *flagshipAction()*:



```
public function flagshipAction()
{
    $_flagshipId = $this->getRequest()->getParam('id', 0);

    $_flagshipPartnersMapper = new Model_Mapper_FlagshipPartners();
    if ($_flagshipId > 0) {
        $_partners = $_flagshipPartnersMapper->fetchMany(array('flagship_id' => $_flagshipId));
        $this->view->partners = $_partners->toArray();
    }

    App_Navigation::AddLink($this->view->url(array(
        'module' => 'default',
        'controller' => 'overview',
        'action' => 'flagship'
    )), null, false), 'Flagship Overview');

    try {
        $_flagshipMapper = new Model_Mapper_Flagship();
        $_flagshipEntity = $_flagshipMapper->fetchOne(array('flagship_id' => $_flagshipId));

        $this->view->mainTitle = $_flagshipEntity->name;
        $this->view->subTitle = 'Overview';

        $this->view->flagship = $_flagshipEntity;

        $_flagshipProjects = array();
        $_projects = $this->_projectData();

        foreach ($_projects as $_projectEntity) {
            foreach ($_projectEntity->flagship_ids as $_projectFlagshipId) {
                if ($_projectFlagshipId == $_flagshipId)
                    $_flagshipProjects[] = $_projectEntity;
            }
        }

        $this->view->projects = $_flagshipProjects;
    } catch (Exception $e) {
    }
}
```

Because this action function gets the required from different data sources, it needed to instantiate different Mappers (Model_Mapper_FlagshipPartners(), Model_Mapper_Flagship()) and assign the resulted data to ZendView instance variables.

- **Index.phtml:** It is the default view for the controller **OverviewController.php** this view doesn't require any backend access to get the required information because it displays google map and draw all the FlagShips with links to view more details about the selected flagship.

The view add these scripts to headScript() to appear in the final rendered page:

```
<?php $this->headScript ()->appendFile ("http://google.com/maps/api/js?sensor=true" );?>
<?php $this->headScript ()->appendFile ($this->baseUrl().
'/assets/admin/pages/scripts/overview/index.js' );?>

<?php $this->placeholder('jsvars')->captureStart();?>
var kml_codes=["ca","esa","nawa","sa","wasds_new_3"];
var overviewLink = '<?=$this-
>url (array ('module'=>'default', 'controller'=>'overview', 'action'=>'flagship'), null, true)?>';
<?php $this->placeholder('jsvars')->captureEnd();?>
<?php $this->placeholder('jscalls')->captureStart();?>
OverviewMap.init();
<?php $this->placeholder('jscalls')->captureEnd() ?>
```

The flagship sites are drawn using KML files in (`uploads/flagship/*.kml`) these files are XML based file format used to display flagships geographic data on the displayed google map. Index.js file contains JSON object to initialize google maps with specific options and styles:

```
var OverviewMap = function() {
    return {
        init : function() {
            var myLatLng = new google.maps.LatLng(25, 0);
            var mapOptions = {
                zoom : 3,
                center : myLatLng
            };

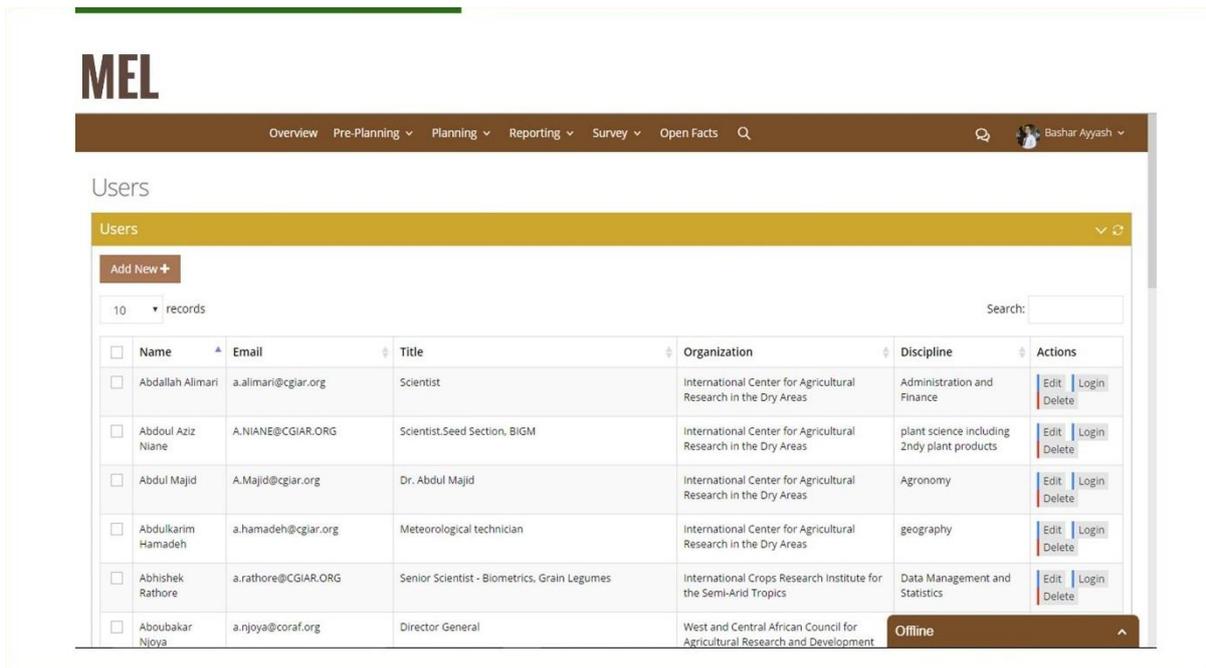
            var simpleStyle = [ {
                "featureType" : "administrative.country",
                "elementType" : "geometry",
                "stylers" : [ {
                    "visibility" : "simplified"
                }, {
                    "hue" : "#ff0000"
                } ]
            } ];
            var map = new google.maps.Map(document.getElementById('map'),
                mapOptions);

            map.setOptions({ styles : simpleStyle });

            var kmlLayer;
            for (index = 0; index < kml_codes.length; index++) {
                kmlLayer = new google.maps.KmlLayer(
                    {
                        url : 'http://mel.cgiar.org/uploads/flagship/'+ kml_codes[index] + '.kml',
                        suppressInfoWindows : true,
                        map : map,
                        preserveViewport : true
                    });
                google.maps.event.addListener(kmlLayer, 'click', function(kmlEvent) {
                    var text = kmlEvent.featureData.description;
                    window.location=overviewLink+'#id/'+text;
                });
            }
        }
    };
}();
```

1. The init() function which initialize google map with zoom degree centered on specific coordinates and style then loop over the kml files in the specified path to draw them on the rendered map.
2. User menu appears for signed user with admin role so he/she can add, delete, and edit users information.
User directory (`/application/modules/default/views/scripts/user/*.phtml`):

Figure 82. User Index Page (<http://mel.cgiar.org/user>)



- **dashboard.phtml**: contains **tabable** metronic component to display page section in tab format user overview, summary information and account settings these information is loaded from `UserController.php dashboardAction() { ... }`, because this action gets user info from many database tables it was required to implement `Model_Mapper_Project()`, `Model_Mapper_FlagshipActivity()`, `Model_Mapper_Flagship()`, `Model_Mapper_ReportFileResult()`, `Model_Mapper_ReportSurvey()`, `Model_Mapper_ResultInfo()`, `Model_Mapper_FlagshipActivityResultDeliverable()`, and `Model_Mapper_FlagshipActivityTrainingResult()` then set the results of the operations on these mappers to `Zend_View` instance so the result can be rendered in the view. In My Account tab if the user updated his information the form input elements are validated using rules set in **form-validation.js** after pressing Save Changes button and before processing the data in `submituserAction()`. The other tabs display the information inside **portlet** metronic components.
- **denied.phtml**: warning view for user with low privilege when try to access a route he/she is not authorized to access, the `deniedAction()` just disable the default layout so the rendered page will use custom view instead of the default layout, because the view does not use the layout it has all the elements of the layout and the custom view.
- **index.phtml**: the index view for `UserController indexAction()` is loaded if there is no action is specified in the URL (`/user/`). The view uses bootstrap tables and jquery data tables plugin to give advance features to the tables (such as Pagination, instant search

and multi-column ordering). The action sets mainTitle variable to the ZendView instance.

- **login.phtml:** the content of the login page, this view contains the fundamental structure of the login page design style and theme with links to continue as guest if the user has no membership.

Figure 83. Login View Page (<http://mel.cgiar.org/user/login>)

MEL

Login Page



The Action:

```
public function loginAction()
{
    if (Zend_Auth::getInstance()->hasIdentity())
        $this->_helper->_redirector('dashboard');
    else
        $this->_helper->layout()->disableLayout();
}
```

Check if the user who accessed the login view script has user object then the view object \$this will redirect the request to the dashboard action, if not then the loginAction will disable the default layout and render the login view script, after entering the login credentials the form will be submitted to UserController authAction() URL

```

public function authAction()
{
    $_userMapper = new Model_Mapper_User ();
    $usersTableName = $_userMapper->getTableName();

    $auth = new App_Auth ($usersTableName, 'email', 'password');

    $email = $this->getRequest()->getParam('email', '');
    $password = $this->getRequest()->getParam('password', '');

    $remember = $this->getRequest()->getParam('remember', 0);
    if ($remember != 0) {
        Zend_Session::rememberMe(60 * 60 * 24 * 60);
    }

    $result = $auth->authenticate($email, $password);

    if ($result > 0)
        $this->_helper->jjson->sendJson(array(
            'message' => 'success'
        ));
    elseif ($result == -1)
        $this->_helper->jjson->sendJson(array(
            'message' => 'Account not Found'
        ));
    elseif ($result == -99)
        $this->_helper->jjson->sendJson(array(
            'message' => 'Account not Active'
        ));
    else
        $this->_helper->jjson->sendJson(array(
            'message' => 'Login Failed'
        ));
}

```

The action use instance of Model_Mapper_User () to get user details stored in the database, instantiate auth object and get the credentials parameters from the request if the checkbox to remember user details was checked then the system will generate cookie with the session details and store it on user machine to 60 days. After that authenticate function will check the credentials data and compare it with the \$usersTable then return integer number according to the user state. Then compare \$result value and send the response as JSON object.

- **message.phtml:** a view with notification message to be shown when the system is down, as it has a different layout than the default layout the messageAction() disable default layout and render the view.

Figure 84. View When the System Is Down (<http://mel.cgiar.org/user/message>)

MEL

System Message

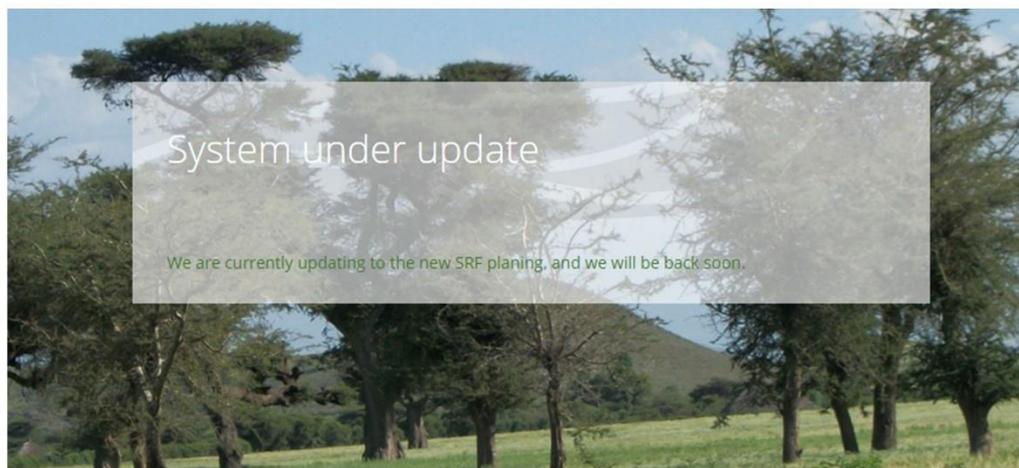


Figure 85. Login view (<http://mel.cgiar.org/user/pdgmlogin>)

MEL

Login View



- **reset.phtml:** reset password view which will be given in the email sent to user, the action `resetAction()` use `Model_Mapper_User ()` to get user info and inside the try block will get hash parameter provided by the request after that the mapper will get user row. The action will check the request object type (Post or Get). The request object is a simple value object that is passed between `Zend_Controller_Front` and the router, dispatcher, and controller classes. It packages the names of the requested module, controller, action, and optional parameters, as well as the rest of the request environment.



Monitoring, Evaluation and Learning Platform: System Design & Architecture

generateRandStr() function helper will generate random string and assign it to salt user variable and update the password in the mapper object and at the end redirect the user to his dashboard.

Figure 86. Reset Password View (<http://mel.cgiar.org/user/reset>)

The screenshot shows the "Reset Password View" interface. At the top, there is a navigation bar with "Overview" and "Open Facts" tabs, a search icon, and a "Login" link. Below the navigation bar, the page title "MEL" is displayed in large, bold letters, followed by "Reset Password View". The main content area contains two password input fields. The first field is labeled "New Password *" and has a placeholder text "Provide your password:". The second field is labeled "Confirm Password *" and has a placeholder text "Confirm your password:". A "Reset password" button is located below the input fields.



```
public function resetAction()
{
    $_userMapper = new Model_Mapper_User ();
    try {
        $_hash = $this->getRequest()->getParam('hash', '');

        $_user = $_userMapper->fetchOne(array('reset_string' => $_hash));
        $this->view->error = '';
        if ($this->getRequest()->isPost()) {
            $_password = $this->getRequest()->getParam('password', '');
            if ($_password != '') {
                $_user->salt = App_Function::generateRandStr(40);
                $_user->rawpassword = $_password;
                $_user->reset_string = '';
                $_userMapper->update($_user->toArray(), 'user_id=' . $_user->user_id);

                $_auth = Zend_Auth::getInstance();
                $_auth->getStorage()->write($_user);
                $this->_helper->redirector('dashboard');
                return;
            }
            $this->view->error = 'System Error';
        }

        $this->view->user = $_user;
        $this->view->as = $_hash;
    } catch (Exception $e) {
        $this->view->error = 'Error in link. Please make sure that you followed the correct link.'
    }
}
```

resetemail.phtml: is a reset email template assigned with forgetAction():

```
public function forgetAction()
{
    $_userMapper = new Model_Mapper_User ();
    try {
        $_email = $this->getRequest()->getParam('email', '');

        $_user = $_userMapper->fetchOne(array('email' => $_email));

        $_activationString = App_String::generateRandStr(50);

        $_user->reset_string = $_activationString;
        $_userMapper->update($_user->toArray(), 'user_id=' . $_user->user_id);

        $this->view->reset_hash = $_activationString;

        $userPasswordResetEmail = $this->view->render('user/resetemail.phtml');

        App_Mail_Sender::SendEmail(array(array('Name' => $_user->name, 'Email' => $_user->email),
        'Password reset!', $userPasswordResetEmail);

        $this->_helper->jjson->sendJson(array(
            'message' => 'success'
        ));
    } catch (Exception $e) {

        $this->_helper->jjson->sendJson(array(
            'message' => 'Reset Failed'
        ));
    }
}
```

The action will use Model_Mapper_User () to get user info and generate random string, update the mapper reset_link, associate the string with reset_hash variable.

The in the template the reset_hash variable will be used to generate complete URL using custom view helper fullURL, then use App_Mail_Sender::SendEmail function and provide the required configuration for sending the email with Password reset subject, then return success message to the helper. If the an error has been generated by the App_Mail_Sender "Reset Failed" message will be sent as json object to the helper.

- **welcomeemail.phtml:** welcome email message template sent to the user when he is added to MEL system.

```
public function welcomeAction()
{
    $_userMapper = new Model_Mapper_User ();
    try {
        $_email = $this->getRequest()->getParam('email', '');

        $_user = $_userMapper->fetchOne(array('email' => $_email));

        $_activationString = App_String::generateRandStr(50);

        $_user->reset_string = $_activationString;
        $_userMapper->update($_user->toArray(), 'user_id=' . $_user->user_id);

        $this->view->reset_hash = $_activationString;

        $userPasswordResetEmail = $this->view->render('user/welcomeemail.phtml');

        App_Mail_Sender::SendEmail(array(array('Name' => $_user->name, 'Email' => $_user->email)),
        'System Access', $userPasswordResetEmail);

        $this->_helper->jjson->sendJson(array(
            'message' => 'success'
        ));
    } catch (Exception $e) {

        $this->_helper->jjson->sendJson(array(
            'message' => 'Reset Failed'
        ));
    }
}
```

In the action it gets user information from Model_Mapper_User () and retrieve one record from the table associated with email then generate and add activation string to reset_string attribute. App_Mail_Sender::SendEmail is used to send email with subject System Access, then send success message as JSON object to the view helper if error occurred while sending an email exception will result with Send Reset Failed error message to the handler. The handler will be displayed in the view.

App_Mail_Sender class provide functionality to compose and send rich-text emails:

```

<?php
class App_Mail_Sender {
    public static function SendEmail($arrTo, $Subject, $Body,$replyTo='') {
        $SMTPHost = 'mail.cgmel.org';
        $SMTPUserName = 'noreply@cgmel.org';
        $SMTPPassword = 'n0Wc#5b5';
        $config = array (
            'auth' => 'login',
            'username' => $SMTPUserName,
            'password' => $SMTPPassword,
            'port' => 25
        );
        $transport = new Zend_Mail_Transport_Smtp ( $SMTPHost, $config );
        $mail = new Zend_Mail ( 'UTF-8' );
        $mail->setBodyText ( 'Please activate HTML view' );

        $mail->setFrom ( 'noreply@cgmel.org','CRP Dryland Systems Monitoring, Evaluation and Learning
System' );
        foreach ( $arrTo as $To )
            $mail->addTo ( $To ['Email'], $To ['Name'] );
        $mail->setSubject ( $Subject );
        $mail->addBcc('noreply@cgmel.org');

        if($replyTo!='')
            $mail->setDefaultReplyTo($replyTo);

        try{
            $viewRenderer = Zend_Controller_Action_HelperBroker::getStaticHelper('viewRenderer');
            if (null === $viewRenderer->view) {
                $viewRenderer->initView();
            }
            $view = $viewRenderer->view;
            $view->body=$Body;
            $view->name=$arrTo[0]['Name'];
            $finalBody=$view->render( 'templates/mail.phtml' );
        }catch(Exception $e){
            throw new Exception($e->getMessage());
        }
        $mail->setBodyHtml ( $finalBody );
        //$mail->send ( $transport );
    }
}

```

It contains static function SendMail (the use of static method is to give a way to reference objects in Zend_Registry scope) has 4 parameters that set the email configuration:

- \$arrTo: contains the address for the recipient of the email.
- \$Subject: the subject of the email.
- \$Body: the body of the email.
- \$replayTo: is not required parameter if it has not been provided to the function empty value will be assigned to the parameter.

Because the mail server requires SMTP authentication additional parameters provided in the body of the function provided to Zend_Mail, the authentication parameters are:

```
$SMTPHost = 'mail.cgmel.org';
$SMTPUserName = 'noreply@cgmel.org';
$SMTPPassword = 'n0Wc#5b5';
$config = array (
    'auth' => 'login',
    'username' => $SMTPUserName,
    'password' => $SMTPPassword,
    'port' => 25
);
```

Zend_Mail declared and the default configurations are set (the encoding UTF-8, the text part of the email just contains 'Please activate HTML view', the sender, the subject and the recipients of the email), inside the try catch block

Zend_Controller_Action_HelperBroker::getStaticHelper('viewRenderer') is declared.

Set the body text message to the view body, set name variable, in the array of the recipients list, to name variable for the \$view object, then render the html message template (`application/modules/default/views/scripts/templates/mail.phtml`) with the values of the \$view object and set the output to \$finalBody variable.

If everything processed without error the \$finalBody is set to the body of the html email: `$mail->setBodyHtml ($finalBody);`

Note: Zend_Mail_Transport_Smtp is declared but not registered with Zend_Mail

User views use the following files to validate the forms and get the data from mappers and display it in the view:

- **table-managed.js**: to instantiate users data tables and fill the table with data from datalink "/user/getallusers" this link is registered with `getallusersAction()` function which gets all the users details from `Model_Mapper_User()` function then returns the result set as JSON object, after the response sent back the plugin will generate table with advance controls (Search Field, Pagination, Sorting). The `initTable` function requires **jquery-1.11.0.min.js** and **jquery.dataTables.min.js** to be loaded before **table-managed.js** is executed



Monitoring, Evaluation and Learning Platform: System Design & Architecture

```
var initTable = function () {
    var table = $('#user');
    // begin first table
    table.dataTable({
        "sAjaxSource": tableDataLink,
        "sAjaxDataProp": "data",
        "columns": [{
            "mData": "user_id",
            "orderable": false,
            "render": function ( data, type, row ) {
                return '<input type="checkbox" class="checkboxes" name="user_id[]"
value="'+data+'"/>';
            }, {
            "mData": "name",
            "orderable": true
        }, {
            "mData": "email",
            "orderable": true
        }, {
            "mData": "title",
            "orderable": true
        }, {
            "mData": "organization",
            "orderable": true
        }, {
            "mData": "discipline",
            "orderable": true
        }, {
            "mData": "user_id",
            "orderable": false,
            "render": function ( data, type, row ) {
                return '<a href="#" class="btn default btn-xs blue-stripe btn-edit" data-
id="'+data+'>Edit </a> <a href="'+base_url+'/user/loginas/id/'+data+'" class="btn default btn-xs
blue-stripe" data-id="'+data+'>Login </a> <a href="#" class="btn default btn-xs red-stripe btn-
del" data-id="'+data+'>Delete </a>';
            },
            "width":100
        }
    ]],
    "lengthMenu": [
        [10, 15, 20, -1],
        [10, 15, 20, "All"] // change per page values here
    ],
    // set the initial value
    "pageLength": 10,
    "pagingType": "bootstrap_full_number",
    "language": {
        "lengthMenu": "_MENU_ records",
        "paginate": {
            "previous": "Prev",
            "next": "Next",
            "last": "Last",
            "first": "First"
        }, "columnDefs": [{ // set default column settings
            'orderable': false,
            'targets': [0]
        }
    ], {
        "searchable": false,
        "targets": [0]
    }
    ],
    "fnRowCallback": function(nRow, aaData, iDisplayIndex) {
        $(nRow).addClass('gradeX');
        $(nRow).children('td').eq(0).addClass('table-checkbox');
        return nRow;
    }, "fnDrawCallback":function(oSettings, json){
        Metronic.init();
    }, "order": [
        [1, "asc"]
    ] // set first column as a default sort by asc
});
```

- **form-validation.js:** javascript file contains validation rules for add user and add user disciplines forms, the file also contains the placement for error messages in addition to validation handlers. The validation rules function requires **jquery-1.11.0.min.js** and **jquery.validate.min.js** to be loaded before **form-validation.js** is executed, rules contain the name of the form elements and the corresponding validations, html inputs that need to be validated must have name attribute because the selectors to validate an input is the name attribute.

The structure of validation rules object:

rules:

```
{
  input_1:
  {
    validation_1: param_1,
    validation_2: param_2
  }
  input_2:
  {
    validation_3: param_3,
    validation_4: param_4
  }
}
```

Snippet of validations in the file:

```

var handleValidation = function() {
  var form1 = $('#user_form');
  var error1 = $('.alert-danger', form1);
  var success1 = $('.alert-success', form1);
  $('input[name=code]').maxlength({
    limitReachedClass: "label label-danger",
  });
  form1.validate({
    errorElement: 'span', //default input error message container
    errorClass: 'help-block help-block-error', // default input error message class
    focusInvalid: false, // do not focus the last invalid input
    ignore: "", // validate all fields including form hidden input
    rules: {
      name: {
        required: true
      },
      title: {
        required: true
      },
      gender: {
        required: true
      },
      email: {
        required: true,
        email: true
      },
      .
      .
      .
    }
  });
};

```

Figure 87. Input fields that will be validated before the form will be submitted to the action

MEL
Input Fields



The screenshot shows a 'New User' registration form with the following fields:

- First Name * (text input, example: e.g. Jack)
- Last Name * (text input, example: e.g. Jack)
- Email * (text input, example: e.g. jack@cgiar.org)
- Password * (password input, label: Provide your password.)
- Confirm Password * (password input, label: Confirm your password)
- Title * (text input, example: e.g. GIS Specialist)
- Country * (dropdown menu)
- City * (text input)
- Daily rate (in USD) (text input)
- Organization * (dropdown menu, label: Select user organization...)
- User Groups * (dropdown menu, label: Select user groups...)
- Discipline * (dropdown menu, label: Select user discipline...)

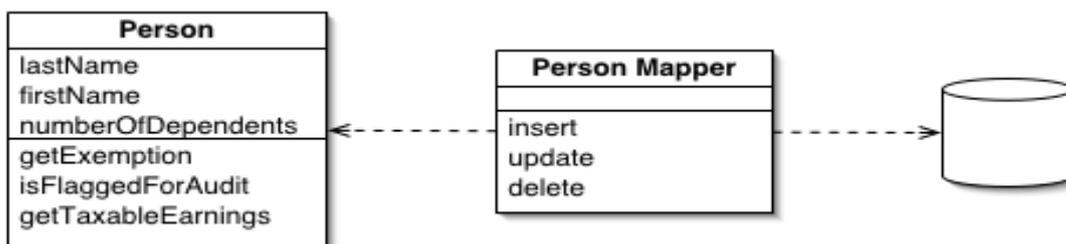
7.5.4 CRUD (Create, Retrieve, Update, Delete)

In Zend Framework’s MVC implementation, each table in the database has an associated class that manages it. In MEL we refer to this class as the Model. As Zend Framework has not defined a model interface, class, or other formalism simply -because they wanted to avoid introducing limitations without significant added value- MEL has defined its custom Model at: (library/App/Model/).

MEL is defining its custom model implementation that partially derives the Data Mapper pattern, controllers are extending these custom mapper models that in turn extends Zend_Db_Table class. The full Data Mapper pattern and a difference to MEL Data Mapper pattern will be briefly described next, MEL custom mapper model will be explained after that.

- Data Mapper (Flower, 2002)**
 A layer of Mappers that moves data between objects and a database while keeping them independent of each other and the mapper itself.

Figure 88. Data Mapper Representative Flow (Flower, 2002)



Objects and relational databases have different mechanisms for structuring data. Many parts of an object, such as collections and inheritance, are not present in relational databases. When building an object model with a lot of business logic it is valuable to use these mechanisms to better organize the data and the behaviour that goes with it. Doing so leads to variant schemas; that is, the object schema and the relational schema do not match up.

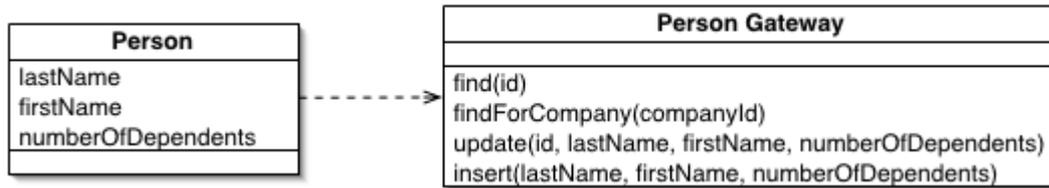
Data transfer between the two schemas will still be needed, and this data transfer becomes a complexity in its own right. If the in-memory objects know about the relational database structure, changes in one tend to ripple to the other.

The Data Mapper is a layer of software that separates the in-memory objects from the database. Its responsibility is to transfer data between the two and to isolate them from each other. With Data Mapper, the in-memory objects need not know even that there is a database present; they need no SQL interface code, and certainly no knowledge of the database schema. (The database schema is always ignorant of the objects that use it.) Since it is a form of Mapper, Data Mapper itself is even unknown to the domain layer (Flower, 2002).

The Data Mapper will use Table Data Gateway to connect to the data source, which will use Zend_Db_Table that provides this functionality.

- Table Data Gateway (Flower, Table Data Gateway, 2002)**
 An object that acts as a Gateway to a database table. One instance handles all the rows in the table.

Figure 89. Table Data Gateway Representative Flow



Mixing SQL in application logic can cause several problems. Many developers are not comfortable with SQL, and many who are comfortable may not write it well. Database administrators need to be able to find SQL easily so they can figure out how to tune and evolve the database.

A Table Data Gateway holds all the SQL for accessing a single table or view: selects, inserts, updates, and deletes. Other code calls its methods for all interaction with the database (Flower, Table Data Gateway, 2002).

Data Mapper best practices require the creation of the following three components:

1. Data Source

A based table class; a table data gateway, which is an object that acts as a Gateway to a database table. It will extend Zend_Db_Table, and provide a table name and optionally the primary key (if it is not "id").

2. Mapper Model

Maps the Domain Model (Object) to the Data source (Object). It will set and get the data source, save, insert, update, find, fetchAll, etc.. , from the Domain Model to the data source. Therefore, it will get fields data from the domain model object and will insert, update, etc., to the data source object.

3. Domain Model

Contains the table rows as fields, and sets/gets them as properties via setters and getters methods.

MEL Mapper Model Customization

MEL system is merging Data Sources with the Mapper Models, so you will find that MEL has no separate Data Source layer (Models extending Zend_Db_Table and defining table names with no other logic inside). This Data Source layer is merged with the Data Mapper layer.

MEL Data Mappers are extending the abstract class App_Model_Mapper_MapperAbstract that in turn extends Zend_Db_Table, this abstraction is found as a custome library located at (library/App/Model/Mapper/MapperAbstract.php).

```
abstract class App_Model_Mapper_MapperAbstract extends Zend_Db_Table
```

Abstract class indicates that App_Model_Mapper_MapperAbstract can't be instantiated on its own, but can only be sub classed. Class App_Model_Mapper_MapperAbstract code is explained below along with Zend_Db_Table usage.

- **MEL Mapper Model (App_Model_Mapper_MapperAbstract)**

- The Zend_Db_Table class is an object-oriented interface to database tables. It provides methods for many common operations on tables. The base class is extensible, so custom logic can be added. The Zend_Db_Table solution is an implementation of the mentioned above Table Data Gateway pattern. The solution also includes a class that implements the Row Data Gateway pattern (An object that acts as a Gateway to a single record in a data source. There is one instance per row).
- Class App_Model_Mapper_MapperAbstract defines the following three auxiliary functions to provide added functionalities/methods on Zend_Db_Table for any class that extends App_Model_Mapper_MapperAbstract:
 1. Classes that extends App_Model_Mapper_MapperAbstract can now get the database table for which extending class is defined, using the public getTableName() method to get the protected variable \$_name. \$_name value is a string, and must contain the name of the table spelled as it appears in the database.

```
public function getTableName(){
    return $this->_name;
}
```

An example of a Mapper model defines protected variable \$_name as a property can be found at (application/model/mapper/SelfAssessment.php):

```
class Model_Mapper_SelfAssessment extends
App_Model_Mapper_MapperAbstract
{
    protected $_name = 'tbl_self_assessment';
}
```

Note that all MEL Mapper models correspond to a table/view in the database, and has this protected variable \$_name defined.

A getTableName() method usage example can be found at:

(application/modules/default/controllers/UserController.php) inside the authAction() as follows:

```
public function authAction()
{
    $_userMapper = new Model_Mapper_User ();
    $usersTableName = $_userMapper->getTableName();
}
```

The UserController authAction is instantiating Model_Mapper_User() that defines protected \$_name = 'tbl_user' and is being retrieved via \$_userMapper->getTableName().

2. Classes that extends App_Model_Mapper_MapperAbstract can now set the database table name for which extending class is defined, using the public setTableName() method that takes the protected variable \$_name as a parameter.

```
public function setTableName($name){
    $this->_name=$name;
}
```



Although `setTableName($name)` method is defined in the Mapper abstract class, but it was never used in the MEL system.

3. Classes that extends `App_Model_Mapper_MapperAbstract` can now get the table primary key. Primary key column is declared for the primary key using the protected variable `$_primary` inside the Mapper models. This is either a string that names the single column for the primary key, or else it is an array of column names if the primary key is a compound key.

`getPrimary()` is retrieving the primary key by using `info()` method defined at (`library/Zend/Db/Table/Abstract.php`), which Returns table information by returning only a part of this information by supplying its key name ('primary').

```
public function getPrimary() {  
    return $this->info('primary');  
}
```

If the primary key is not specified, `Zend_Db_Table_Abstract` tries to discover the primary key based on the information provided by the `describeTable()` method.

Setting the primary key inside MEL mapper models is done by: Overriding table setup methods. Table setup methods are a set of protected methods that initialize metadata for the table called by the constructor. An example can be found at: (`application/model/mapper/ActionsiteBudget.php`)

```
class Model_Mapper_ActionsiteBudget extends  
App_Model_Mapper_MapperAbstract  
{  
    protected $_name = 'view_actionsite_budget';  
  
    protected function _setupPrimaryKey()  
    {  
        $this->_primary = 'actionsite_id';  
        parent::_setupPrimaryKey();  
    }  
}
```

`_setupPrimaryKey()` method defaults the primary key columns to those reported by `describeTable()`; checks that the primary key columns are included in the `$_cols` array. By overriding this method.

- Class `App_Model_Mapper_MapperAbstract` defines the following two auxiliary functions to provide added functionalities/methods on `Zend_Db_Table` for any class that extends `App_Model_Mapper_MapperAbstract`. These two methods manipulates the `Zend_Db_Table_Rowset_Abstract` class.

1. The `_createCollection` method takes an object of type `Zend_Db_Table_Rowset_Abstract`. A Rowset contains a collection of objects descending from `Zend_Db_Table_Row_Abstract`. Because `Zend_Db_Table_Rowset_Abstract` implements the `SeekableIterator` interface, it

provides the ability to iterate through the Rowset via foreach() and access individual Row objects, reading or modifying data in the Rows.

```

protected function
_createCollection(Zend_Db_Table_Rowset_Abstract $results)
{
    $collectionClass=str_replace('_Mapper', '',
    get_class($this)).'Collection';

    $collection = new $collectionClass;

    foreach ($results as $row) {
        $collection[] = $this->_createEntity($row);
    }

    return $collection;
}

```

\$collectionClass will contain the replace class name as follows:

It will get the current mapper class name, i.e. class Model_Mapper_ActionsiteAls, it will replace the _Mapper with nothing to it will become Model_ActionsiteAls, then will append the string 'Collection' with a final result of: Model_ActionsiteAlsCollection which is an already created Collection class inside the model/ directory.

Then it will instantiate the collection class, iterate through it saving data in rows as entities in a name = value format.

- The _createEntity method takes an object of type Zend_Db_Table_Row_Abstract, each retrieved value from the Rowset is a Zend_Db_Table_Row_Abstract object that corresponds to one record from the table. Zend_Db_Table_Row_Abstract provides accessor methods so columns can be referenced in the row as object properties:

```

protected function _createEntity(Zend_Db_Table_Row_Abstract
$results)
{
    $entityClass=str_replace('_Mapper', '', get_class($this));

    $entity=new $entityClass;
    foreach ($results as $columnName=>$columnValue)
        $entity->$columnName=$columnValue;
    return $entity;
}

```

\$entityClass will contain the replaced class name as follows:

It will get the current mapper class name, i.e. class Model_Mapper_ActionsiteAls, it will replace the _Mapper with nothing to it will become Model_ActionsiteAls, which is an already created Collection class inside the model/ directory.

Then it will instantiate the corresponding model class, iterate through it saving data in a name = value format.

At this point, an override for `_createEntity()` at: `(library/App/Model/Mapper/JsonMapperAbstract.php)` should be mentioned. This `App_Model_Mapper_JsonMapperAbstract` class is being extended by one model mapper only, which is `(application/model/mapper/Project.php)`. `_createEntity()` implementation is the same as above except that the final `$columnName`, and `$columnValue` values are decoded via `Zend_Json::decode` as follows:

```
protected function _createEntity(Zend_Db_Table_Row_Abstract $results)
{
    $entityClass=str_replace('_Mapper', '', get_class($this));

    $entity=new $entityClass;

    foreach ($results as $columnName=>$columnValue)
    {
        //if(!in_array($columnName, $this->_jsonArray))
            $entity->$columnName=$columnValue;
    }

    foreach ($this->_jsonArray as $fieldName){
        if(isset($results[$fieldName])){
            $decodedArray=Zend_Json::decode($results[$fieldName]);
            foreach ($decodedArray as $jsonFieldName => $jsonFieldValue)
                $entity->$jsonFieldName=$jsonFieldValue;
        }
    }

    return $entity;
}
```

Defining custom logic for Insert, Update, and Delete

MEL overriinge the `insert()`, `update()`, and `delete()` methods inside `App_Model_Mapper_MapperAbstract`. By this we are implementing custom code that is executed before performing the database operation. This custom code is the same for the three operations, and include logging some data about the user and operation, i.e. `table_name`, `user_id`, `date`, and other operation details. Here you can find the custom MEL `delete()` method.

```
public function delete($where)
{
    //use log table to add ne wactivity
    $_logMapper = new Model_Mapper_Log();
    $_auth = Zend_Auth::getInstance ();
    $_user = $_auth->getIdentity ();
    $_logModal = new Model_Log();
    $_logModal->action_type = 2;
    $_logModal->table_name = $this->getTableName();
    $_logModal->user_id = $_user->user_id;
    $_logModal->date = gmdate('Y-m-d H:i:s');
    $_logModal->details = json_encode(array('where' => $where));
}
```

```

$_logMapper->insert($_logModal->toArray());

return parent::delete($where);
}

```

MEL Custom search methods

MEL has a frequent need to do queries against tables with specific criterias, for this we are embedding these queries at run time by the help of the following Zend base functions customized according to our needs.

1. Querying for a single row / fetchRow()

MEL is using the following custom function to query a single row:

```

public function fetchOne($criteria=null,$order=null){
    if ($criteria==null && $order==null)
        $results = $this->fetchRow();
    else
    {
        $selectCriteria= $this->select();
        if($criteria!=null){
            foreach ($criteria as $field=>$value){
                if(!is_array($value))
                    $selectCriteria->where($field . ' = ?', $value);
                else
                    $selectCriteria->where($field . ' IN (?)', $value);
            }
        }
        if($order!=null)
            $selectCriteria->order($order);
        $results = $this->fetchRow($selectCriteria);
    }
    if($results == null)
        throw new Exception('Entity Not Found');
    else
        return $this->_createEntity($results);
}

```

As parameters show, this functions relies on a criteria: A criteria is similar to a WHERE clause in SQL, but it's not in SQL format. It uses objects instead. The function implies the normal use of fetchRow() if there is no criteria passed, and the where clause implementation based on the passed criteria.

2. Fetching a rowset / fetchAll()

In Zend, a set of rows query is performed by using any criteria other than the primary key values, via the fetchAll() method of the Table class. This method returns an object of type Zend_Db_Table_Rowset_Abstract.

MEL is using the following custom function to query a rowset:

```

public function fetchMany($criteria=null,$order=null,$count=null,$offset=-
1){
if ($criteria==null && $order==null)
    $results = $this->fetchAll();
else
{
    $selectCriteria= $this->select();
    if ($criteria!=null)
    {
        if(is_array($criteria)){
            foreach ($criteria as $field=>$value){
                if(is_numeric($field)){
                    $selectCriteria->where($value);
                }
                else {
                    if(substr($field, 0,1)!='!'){
                        if(!is_array($value))
                            if($value!=null)
                                $selectCriteria->where($field . ' = ?', $value);
                        else
                            $selectCriteria->where($field . ' IS NULL');
                        else
                            $selectCriteria->where($field . ' IN (?)', $value);
                    }
                }
            }
        }
        else{
            $fieldName=substr($field, 1);
            if(!is_array($value))
                if($value!=null)
                    $selectCriteria->where($fieldName . ' <>
?', $value);
            else
                $selectCriteria->where($fieldName . ' IS NOT
NULL');
            else
                $selectCriteria->where($fieldName . ' NOT IN
(?)', $value);
        }
    }
    else
        $selectCriteria->where($criteria);
}

if($order!=null)
    $selectCriteria->order($order);

```

```

        if($count!=null && $offset!=-1)
            $selectCriteria->limit($count,$offset);
        $results = $this->fetchAll($selectCriteria);
    }
    return $this->_createCollection($results);
}

```

The function implies the normal use of fetchAll() if there is no criteria passed, and the where clause implementation based on the passed criteria.

- App_Model_Mapper_Exception interface located at: (library/App/Model/Mapper/Exception.php) is extending App_Model_Exception located at (library/App/Model/Exception.php), this in turn extends App_Exception located at (library/App/Exception.php) which is an empty interface.
- App_Model_Mapper_RuntimeException located at: (library/App/Model/Mapper/RuntimeException.php) is extending RuntimeException a base Zend class located at (library/Zend/Http/Header/Exception/RuntimeException.php), and is implementing App_Model_Mapper_Exception interface located at: (library/App/Model/Mapper/Exception.php) is extending App_Model_Exception located at (library/App/Model/Exception.php), this in turn extends App_Exception located at: (library/App/Exception.php) which is an empty interface.

- **MEL Domain Model (App_Model_ModelAbstract)**

MEL Domain models are extending the abstract class App_Model_ModelAbstract located at (library/App/Model/ModelAbstract.php), App_Model_ModelAbstract is defining the allowed table fields as an array, along with its values then assign values to field's property at its constructor.

```

abstract class App_Model_ModelAbstract
{
    protected $_fields = array();

    protected $_values = array();

    public function __construct(array $values=array())
    {
        foreach ($this->_fields as $name) {
            $this->$name = "";
        }
        foreach ($values as $name => $value) {
            $this->$name = $value;
        }
    }
}

```

- **MEL Models Relationship Support**

MEL configures models to support relationships, by defining the relationships within the model.

Both `getParentRelation()`, and `getChildrenRelation()` define the relationship, informing Zend_Db of the existence of a column within the `$mapper` table model which stores a foreign key pointing to a row managed by the current implementing model.

Both `getParentRelation()`, and `getChildrenRelation()` takes an object from `App_Model_Mapper_MapperAbstract` which is the parent/children mapper model, `$method` which is the query needs to be done i.e. `fetchone`, `fetchCount`, etc ..

`$arguments` parameter indicates the foreign key between the two tables.

```
public function getParentRelation(App_Model_Mapper_MapperAbstract $mapper,
    $method, array $arguments){
    $modelRelation=new App_Model_Relation($mapper, $method, $arguments);
    return $modelRelation;
}
```

```
public function getChildrenRelation(App_Model_Mapper_MapperAbstract
    $mapper, $method, array $arguments){
    $modelRelation=new App_Model_Relation($mapper, $method, $arguments);
    return $modelRelation;
}
```

This Relationship support is instantiating `App_Model_Relation` located at: `(library/App/Model/Relation.php)` implementing the following Standard PHP Libraries (SPLs): `ArrayAccess`, `Countable`, `IteratorAggregate`.

The Standard PHP Library is a great addition to PHP 5. It provides a number of very useful facilities that expose some of PHP's internal functionality and allow the "userland" developer to write objects that are capable of behaving like arrays, or that transparently implement certain iterative design patterns to PHP's own core functionality, so that you, for example, use a `foreach()` construct to loop through an object as if it were an array, or even access its individual elements using the array operator `[]`.

SPL works primarily by providing a number of interfaces that can be used to implement the functionality required to perform certain operations.

1. Accessing Objects as Arrays (`ArrayAccess`)

The `ArrayAccess` interface can be used to provide a means for objects to be exposed as pseudo-arrays to PHP. To implement `ArrayAccess` four methods should be implemented: `offsetExists`, `offsetGet`, `offsetSet` and `offsetUnset`. `ArrayAccess::offsetExists` must return a boolean, `offsetGet` can return any valid PHP type while `offsetSet` and `offsetUnset` should not return any value. The following shows how `App_Model_Relation` implements them:

```
public function offsetExists($offset) {
    return array_key_exists($offset, $this->_iterator);
}
```

```

public function offsetGet($offset) {
    if ($this->_iterator === null) {
        $this->_iterator=$this->getIterator();
    }

    return $this->_iterator[$offset];
}
public function offsetSet($offset, $entity) {}

public function offsetUnset ($offset) {
}

```

2. Counting elements of an object (Countable)

The Countable interface allows to pass objects that implement it to PHP's native count function. To implement Countable Countable::count() method should be added to the implementing class and this method only needs to return an integer. Passing an object that implements Countable to the PHP count function, will force the PHP interpreter to automatically call the count method in the object. This is how App_Model_Relation is implementing Countable:

```

public function count()
{
    return count($this->getIterator());
}

```

3. Looping over objects (IteratorAggregate)

The IteratorAggregate interface defines objects behavior in a foreach loop, this will provide the ability to define custom logic which governs the values to return when an object is iterated over. Implementing IteratorAggregate requires getIterator() method and this is how App_Model_Relation is implementing getIterator():

```

public function getIterator()
{
    if ($this->_iterator === null) {
        $this->_iterator = call_user_func_array(array($this->_mapper, $this->_method), $this->_arguments);
    }

    return $this->_iterator;
}

```

This is also used to route method calls to the iterator:

```

public function __call($name, array $arguments)
{
    return call_user_func_array(array($this->getIterator(), $name), $arguments);
}

```

- App_Model_ModelAbstract is also overriding the two magic methods (__set(), and __get()), overriding indicates that MEL can use the syntax (__set) concatenated to the table field name with first letter capitalized, so if we are setting als we can use the following: __setAls(\$name, \$value). In the code the first check is trying to find a defined method named __setAls(\$name, \$value), the second one will accept and set the field value if the \$name is found in the table \$fields.

The same logic mentioned above is applied to the __get(\$name) magic method.

```

public function __set($name, $value)
{
    $method = '_set' . implode("", array_map('ucfirst', explode('_', $name)));

    if (method_exists($this, $method)) {
        $this->{$method}($value);
    } else if (in_array($name, $this->_fields)) {
        $this->_values[$name] = $value;
    } else {
        throw new App_Model_OutOfBoundsException('Field with name ' .
$name . ' does not exist');
    }
    return $this;
}

public function __get($name)
{
    $method = '_get' . implode("", array_map('ucfirst', explode('_', $name)));

    if (method_exists($this, $method)) {
        return $this->{$method}();
    } else if (in_array($name, $this->_fields)) {
        if (!array_key_exists($name, $this->_values)) {
            throw new App_Model_RuntimeException("Trying to accessing
field ' . $name . ' which value was not set yet');
        }

        return $this->_values[$name];
    } else {
        throw new App_Model_OutOfBoundsException('Field with name ' .
$name . ' does not exist');
    }
}

```

At this point, an override for _createEntity() at:

(library/App/Model/Mapper/JsonMapperAbstract.php) should be mentioned.

This App_Model_Mapper_JsonMapperAbstract class is being extended by one model mapper only which is (application/model/mapper/Project.php). _createEntity() implementation is the same as above except that the final \$columnName, and \$columnValue values are decoded via Zend_Json::decode as follows: (library/App/Model/JsonModelAbstract.php).

- `App_Model_RuntimeException` located at: `(library/App/Model/RuntimeException.php)` is extending `RuntimeException` a base Zend class located at `(library/Zend/Http/Header/Exception/RuntimeException.php)`, and is implementing `App_Model_Exception` interface located at `(library/App/Model/Exception.php)`, this in turn extends `App_Exception` located at `(library/App/Exception.php)` which is an empty interface.
- `App_Model_RowNotFoundException` located at: `(library/App/Model/RowNotFoundException.php)` is extending `RuntimeException` a base Zend class located at `(library/Zend/Http/Header/Exception/RuntimeException.php)`, and is implementing `App_Model_Exception` interface located at `(library/App/Model/Exception.php)`, this in turn extends `App_Exception` located at `(library/App/Exception.php)` which is an empty interface.
- `App_Model_OutOfBoundsException` located at: `(library/App/Model/OutOfBoundsException.php)` is extending `RuntimeException` a base Zend class located at `(library/Zend/Http/Header/Exception/RuntimeException.php)`, and is implementing `App_Model_Exception` interface located at: `(library/App/Model/Exception.php)`, this in turn extends `App_Exception` located at: `(library/App/Exception.php)` which is an empty interface.
- `App_Model_InvalidArgumentException` located at: `(library/App/Model/InvalidArgumentException.php)` is extending `InvalidArgumentException` a base Zend class located at: `(library/Zend/Http/Header/Exception/InvalidArgumentException.php)`, and is implementing `App_Model_Exception` interface located at `(library/App/Model/Exception.php)`, this in turn extends `App_Exception` located at `(library/App/Exception.php)` which is an empty interface.
- `App_Model_InvalidArgumentException` located at: `(library/App/Model/InvalidArgumentException.php)` is extending `InvalidArgumentException` a base Zend class located at: `(library/Zend/Http/Header/Exception/InvalidArgumentException.php)`, and is implementing `App_Model_Exception` interface located at `(library/App/Model/Exception.php)`, this in turn extends `App_Exception` located at `(library/App/Exception.php)` which is an empty interface.
- `App_Model_Exception` interface is located at `(library/App/Model/Exception.php)`, and is extending `App_Exception` located at `(library/App/Exception.php)` which is an empty interface.

After learning about Models here is an MEL CRUD example, a simple explanation of CRUD functions are:

- **Create:** Create rows directly with `Zend_Db_Table`'s `insert()` method (`App_Model_Mapper_MapperAbstract::insert` in our case), or creates a new row, add the data to it, and save it.
- **Read/Retrieve:** A number of methods exist for reading data, but the two most common are, first, building a select query using the `Zend_Db_Select` object and passing this to the `Zend_Db_Table` `fetch` methods and, second, fetching a row using its primary key with `Zend_Db_Table`'s `find()` method. MEL perform reads by the following methods:
`App_Model_Mapper_MapperAbstract::fetchOne`
`App_Model_Mapper_MapperAbstract::fetchMany`
`App_Model_Mapper_MapperAbstract::fetchCount`
- **Update:** `Zend_Db_Table`'s `update()` method is used to update rows, or changes can be directly made to the row and then use the `Zend_Db_Table_Row`'s `save()` method. MEL perform updates by the following method:
`App_Model_Mapper_MapperAbstract::update`

- Delete: Rows can be deleted by passing a WHERE clause to the Zend_Db_Table's delete(). MEL perform deletion by the following method:
App_Model_Mapper_MapperAbstract::delete

On the PreplanningController located at:

(application/modules/default/controllers/PreplanningController.php), the submitidoAction() is instantiating its mapper (Model_Mapper_Ido()) which will represent the table named 'tbl_ido' through the \$_idoMapper object.

The \$_idoEntity object will include the table fields array definition, with a default primary key as id. submitidoAction() will first get these parameters from the request object, instantiating both mapper and domain model, assigning values to the current object properties, then checks if the \$_idoId is changed then it will recognize this as an update and will perform the update action from App_Model_Mapper_MapperAbstract::update. Otherwise, action will be considered as a new ido insertion and App_Model_Mapper_MapperAbstract::insert will be called to create a new row in the table 'tbl_ido'.

```

public function submitidoAction()
{
    $_idoId = $this->getRequest()->getParam('ido_id', 0);
    $_code = $this->getRequest()->getParam('code', "");
    $_name = $this->getRequest()->getParam('name', "");
    $_notes = $this->getRequest()->getParam('notes', "");

    $_idoMapper = new Model_Mapper_Ido ();
    $_idoEntity = new Model_Ido ();

    $_idoEntity->code = $_code;
    $_idoEntity->name = $_name;
    $_idoEntity->notes = $_notes;

    if ($_idoId != 0) {
        $_idoEntity->ido_id = $_idoId;
        $_idoMapper->update($_idoEntity->toArray(), 'ido_id = ' . $_idoId);
        $_action = 'update';
    } else {
        $_idoMapper->insert($_idoEntity->toArray());
        $_action = 'add';
    }

    $this->_helper->json->sendJson(array(
        'message' => 'success',
        'action' => $_action
    ));
}

```

Retrieve Operation can be shown on the getalsAction() on the same PreplanningController. We first need the als object data mapper (Database connection), this is done by instantiating the

(Model_Mapper_Als()), which will represent the table named 'tbl_als' through the \$_alsMapper object. fetchOne() will retrieve a single row based on a clause.

```

public function getallalsAction()
{
    $_alsCollection = $this->_alsData();
    $_jsonArray = array();
    foreach ($_alsCollection as $_alsEntity)
        $_jsonArray [] = $_alsEntity->toArray();
    $this->_helper->json->sendJson(array(
        'data' => $_jsonArray
    ));
}

```

Delete operation can be shown on the delactionsiteAction() on the same PreplanningController. We first need the actionsite object data mapper (Database connection), this is done by instantiating the (Model_Mapper_Actionsite()), which will represent the table named 'tbl_actionsite' through the \$_actionsiteMapper object. Zend delete() action that is performed by App_Model_Mapper_MapperAbstract::delete needs a where clause.

```

protected function delactionsiteAction()
{
    $_actionsiteId = $this->getRequest()->getParam('id', 0);

    $_actionsiteMapper = new Model_Mapper_Actionsite ();

    $_actionsiteMapper->delete('actionsite_id = ' . ( int )$_actionsiteId);

    $this->_helper->json->sendJson(array(
        'result' => true
    ));
}

```

7.6 Software Functions

This subsection provides a summary of the major features of the system.

Table 19. Feature #1 Project Management

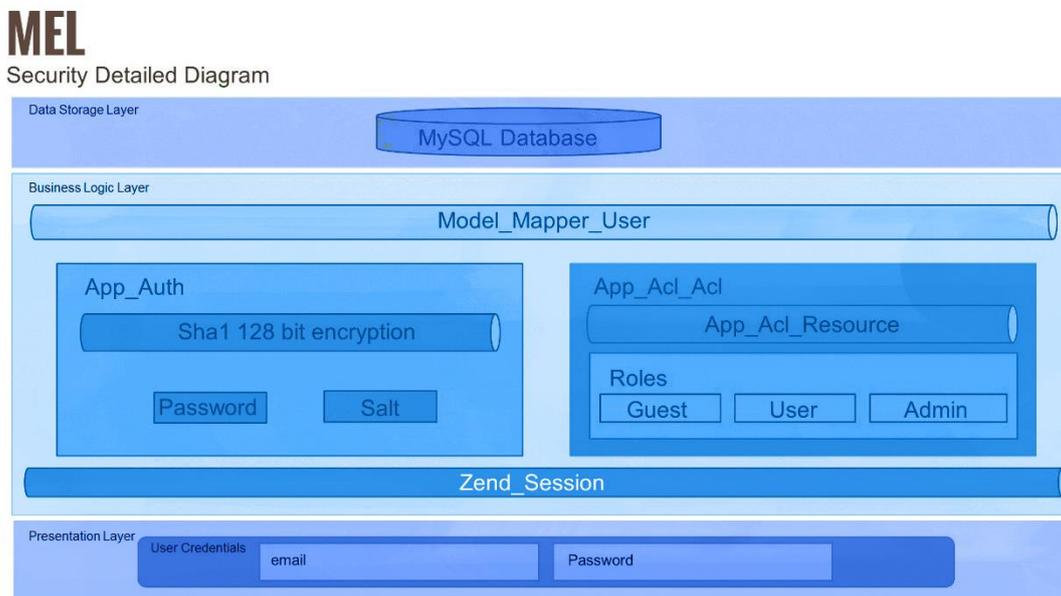
Performed by	Description	Current Functionality
Project Manager	a Flagship, or actionsite manager	<ul style="list-style-type: none"> - Edit project info. - Planning by setting Outputs and Outcomes. - Reporting project outputs, and deliverables. - Review activities, deliverables for their scientists and workers, then approve or reject.
Administrator	ICARDA's MEL administrators	<ul style="list-style-type: none"> - Create a partner (organization). - Create Contacts, those still not users. - Create a User that assigned to the created partner. - Create a FlagShip, and set some indicators. - Create an actionsite/cluster. - Create a project. - Assign a project manager/leader to the project. - Review activities, deliverables, and approve or reject.
Guest	Scientists, students, etc...	<ul style="list-style-type: none"> - Has a full overview of Flagships, actionsite, workshops, projects, who's coordinating a flagship, what ALSes do we have, focal points of each institution, the billathural projects, indicators, he can go on action sites too, and partners information.

7.7 Security Detailed Design

Instructions: Provide a graphical representation with detailed information for each of the individual security components. Specify the design for the below items as required.

- a) Authentication
- b) Authorization
- c) Logging and Auditing
- d) Encryption
- e) Network ports usage
- f) Intrusion Detection and Prevention (especially if hosted in a non-CMS data centre)

Figure 90. Security Detailed Diagram



This figure shows software security layer provided by the system. First users can access the system after entering their credentials (email and password) then App_Auth gets request parameters, and search for a user associated with the entered email through Model_Mapper_User then use sha1 encryption after combining salt record in the database and the entered password, to check user entered password. After retrieving user information and check accuracy of the password, the system checks the role of the user and finally register the status of the user to the session using Zend_Session so protected routes can get the session details and decide whether to allow or disallow user access according to the role type.

- **Authentication**

MEL system is combining the usage of Zend_auth and Zend_Auth_Adapter_DbTable inside library/App/Auth.php to provide a full authentication process for the users by email and password.

Zend_Auth provides an API for authentication and includes concrete authentication adapters for common use case scenarios, and is used to authenticate against a particular type of authentication service, such as LDAP, RDBMS, or file-based storage. Authentication is defined as determining whether an entity actually is what it purports to be (i.e., identification), based on some set of credentials.

Zend_Auth_Adapter_DbTable provides the ability to authenticate against credentials stored in a database table. Because Zend_Auth_Adapter_DbTable requires an instance of Zend_Db_Adapter_Abstract to be passed to its constructor, each instance is bound to a particular database connection.

- **Authorization**

MEL system is using the Zend_Acl component to handle the task of building a tree of roles, resources and privileges to manage and query authorization requests against. This is done at library/App/Acl/.

Zend_Acl_Assert_Interface is also implemented to dynamically determine, if some roles has access to some resources, with some optional privilege that can only be answered by the logic within the assertion. This is done at library/App/Acl/Assert/.

Authorization is defined as the process of deciding whether to allow an entity access to, or to perform operations upon other entities.

- **Encryption**

Simple hashing techniques is used once needed, i.e. sha1 is used at this domain model: (application/model/User.php).

- **Audit Trails**

MEL system is logging some important information to database while processing the custom Model Mapper objects.

MEL model is logging some information to database level. Additionally, Windows server, and Apache provides logging.

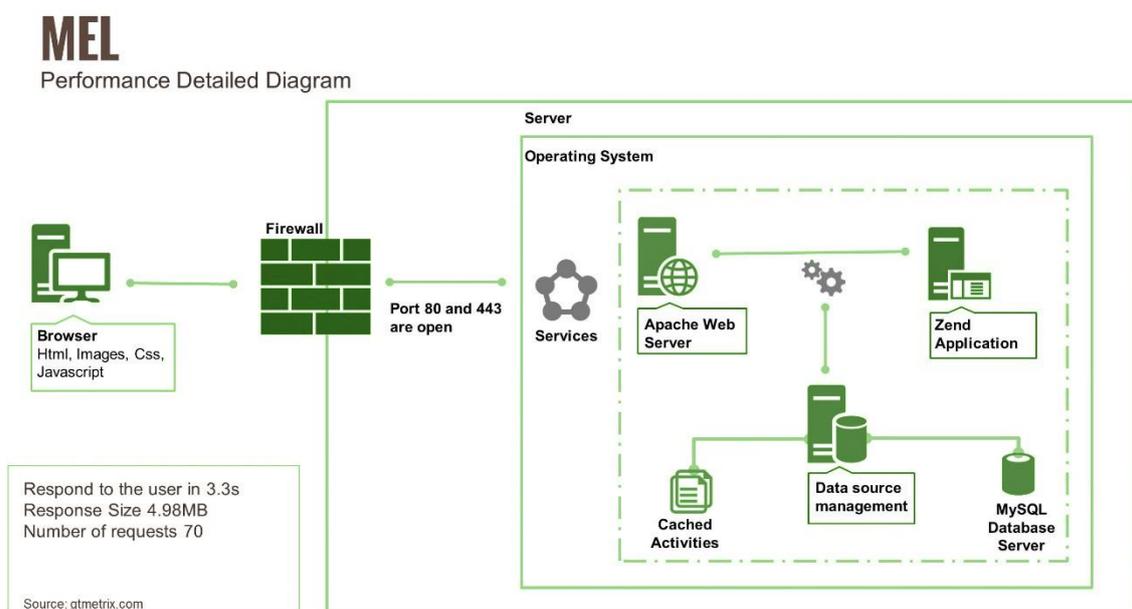
7.8 Performance Detailed Design

Instructions: Provide a graphical representation with detailed information for each of the individual performance and reliability components to include the below items:

- Capacity and volume requirements/estimates
- Performance expectations
- Availability requirements
- Performance design to meet capacity requirements
- Reliability design to meet availability requirements
- Backup, recovery, and archive design

Identify single points of failure and, if relevant, describe high availability design (e.g., clustering).

Figure 91. Performance Detailed Design





In this graph it shows the rendered pages contents (unoptimized contents html, css and javascript files) of the system, the open ports for accessing the system through firewall, operating system services which load (web server, zend framework, and database) then how the system loads data from database or from cached files.

7.9 Software Components Off the Shelf (COTS)

Instructions: For each COTS (Visualisation, version control etc.) describe:

- Its identification and version
- Its purpose
- Where it comes from: manufacturer ...
- Whether it is maintained by a third party or not
- Its interfaces and data flows

Metronic is a responsive and multipurpose admin and frontend theme powered with Twitter Bootstrap 3.1.1 Framework. Metronic can be used for any type of web applications: custom admin panels, admin dashboards, CMS, CRM, SAAS and websites: business, corporate, portfolio, one page parallax, blog. Metronic has a sleek, clean and intuitive metro & flat balanced design which makes your next project look awesome and yet user friendly. Metronic has a huge collection of plugins and UI components and works seamlessly on all major web browsers, tablets and phones.

7.10 Achievement of functional requirements

Instructions: For each main function of the system, add a description of the sequences / data flow that occur. Use sequence diagrams, collaboration diagrams. Describe the workflow/sequence of the main function, from user perspective.

The following tables (Table. 20 – Table. 26) represent MEL System workflow for each MEL process. Please note that the achievement of functional requirements for any system won't be completed unless the project technical requirements is completed, hence the following Workflows is what MEL system would achieve at its final release that has all the technical requirements done and complete.

Table 20. Workflow / Sequence 0: Project Definition

Actors	<ul style="list-style-type: none"> i. Admin User. ii. Project Manager (User). iii. Financial Administrator (User-Focal Point). iv. Guest.
Work Flow	<ul style="list-style-type: none"> i. Admin User: <ul style="list-style-type: none"> a. Select Project from Pre-Planning, Select Add new Project and fill "General Information": Add Project Name, Select Implementing Partner, Assign a Project Manager, Specify the total budget, Define the Country (ies), Define the CRP mapping, Define Period of Implementation. b. Add "Notes". ii. Project Manager (User): Complete info related to project : "Objectives and Targets", "Where we work", "Media", "Key Documents", Donor's Reports", "Center's Strategic Framework", "Notes". iii. Financial Administrator (User-Focal Point): <ul style="list-style-type: none"> a. Discuss with Project Manager the Budget allocation.

Actors	<ul style="list-style-type: none"> b. Insert the budget allocation by category for each year. c. Develop a sync with OCS in order to update the burn rate. If not available add a function for the Financial Administrator to enter the spent budget every 3-6-12 months. iv. Guest: Can register his detail (email, pwd, bio, and picture) and see all projects info in a view mode.
--------	---

Table 21. Workflow / Sequence 1: Project Planning and Monitoring

Actors	<ul style="list-style-type: none"> i. Project Manager (User). ii. Reviewers (User-Center Focal Point; CapDev Focal Point; Gender Focal Point). iii. Guest.
Work Flow	<ul style="list-style-type: none"> i. Project Manager (User): <ul style="list-style-type: none"> a. Add planned resources "Scientists and Consultants", "Partners", and expected results "Outputs and Deliverables", "Trainings and Workshops", "Outcomes", "Research Phase". b. Review Budget allocation against planned resources. ii. Reviewers (User-Center Focal Point; CapDev Focal Point; Gender Focal Point): Approve related sections as entered by the Project Manager. iii. Guest: Can review level of achievements of each expected results.

Table 22. Workflow / Sequence 2: Financial Accounting

Actors	<ul style="list-style-type: none"> i. Financial Administrator (User-Focal Point). ii. Project Manager (User). iii. Reviewers (User-Center Focal Point; CapDev Focal Point; Gender Focal Point). iv. System. v. Guest.
Work Flow	<ul style="list-style-type: none"> i. Financial Administrator (User-Focal Point): <ul style="list-style-type: none"> a. Discuss with Project Manager the Budget allocation. b. Insert the budget allocation by category for each year. c. Develop a sync with OCS in order to update the burn rate. If not available add a function for the Financial Administrator to enter the spent budget every 3-6-12 months. Is notified if resources cannot be covered by the allocation and/or if budget is not committed. ii. Project Manager (User): <ul style="list-style-type: none"> a. At project start define needed resources. Set time allocation for staff and system notifies resources of committed budget. b. Set consultant time and partners engagement.

Actors	<ul style="list-style-type: none"> c. System notify legal officer to process contractual obligations. iii. Reviewers (User-Center Focal Point; CapDev Focal Point; Gender Focal Point): <ul style="list-style-type: none"> a. Approve plan of work and budget. b. Review financial allocation for cross-cutting themes. iv. System: <ul style="list-style-type: none"> a. Notify inconsistency with error message. b. Display allocation and burn rate to Project Manager, Financial Administrator and Approval Focal Points. v. Guest: <ul style="list-style-type: none"> a. System compiles financial information from Flagship, Cluster, Project and Open Facts Section and return them in each dedicated section.
--------	--

Table 23. Workflow / Sequence 3: Human Resources Allocation

Actors	<ul style="list-style-type: none"> i. Project Manager (User). ii. Reviewers (User-Center Focal Point). iii. Guests and Users selected in the Projects.
Work Flow	<ul style="list-style-type: none"> i. Project Manager (User): <ul style="list-style-type: none"> a. Select Staff, Consultants and Partners. b. Identify staff time needed and assign expected deliverables. ii. Reviewers (User-Center Focal Point): <ul style="list-style-type: none"> a. Approve resources allocation. iii. Guests and Users selected in the Projects: <ul style="list-style-type: none"> a. User Receives notification and annual allocation across projects. b. Guest can view summary under each section (Project, Flagship, Cluster).

Table 24. Workflow / Sequence 4: Surveys

Actors	<ul style="list-style-type: none"> i. Guests and Users (incl. Project Manager and Focal Points). ii. Admin User.
Work Flow	<ul style="list-style-type: none"> i. Guests and Users (incl. Project Manager and Focal Points): <ul style="list-style-type: none"> a. User is required to specify the survey type, intended scope and audience. b. User customizes the survey as needed and select audience from Partners' contact database. c. User can add contacts to be validated by Admin. ii. Admin User: <ul style="list-style-type: none"> a. Activate function to the requiring user. b. User approves contacts and allow user to initiate survey and see reporting.

Table 25. Workflow / Sequence 5: Impact Pathway

Actors	<ul style="list-style-type: none"> i. Project Manager (User). ii. System.
--------	---

Work Flow	<ul style="list-style-type: none"> i. Project Manager (User): <ul style="list-style-type: none"> a. Project Manager in Manage section links Output with Outcomes. b. Specifies Output/Outcome characterization, risks, assumptions, and share. ii. System: <ul style="list-style-type: none"> a. System displays interactive, collapsible, impact pathway disaggregated by country. b. Aggregate different IPs under the same country, Cluster and Flagship.
-----------	--

Table 26. Workflow / Sequence 6: Knowledge Sharing

Actors	<ul style="list-style-type: none"> i. Guests and Users. ii. Reviewers (User-Center Focal Point; CapDev Focal Point; Gender Focal Point).
Work Flow	<ul style="list-style-type: none"> i. Guests and Users: <ul style="list-style-type: none"> a. Guests and Users activate a discussion forum organized with keywords with any contact in the systems. b. Recipients receive an email and should log-in to reply. ii. Reviewers (User-Center Focal Point; CapDev Focal Point; Gender Focal Point): <ul style="list-style-type: none"> a. Reviewers activate a discussion forum organized with keywords directly in the review panel with the Project Manager or Users submitting a modification (Plan of Work and Budget; Reporting; Survey; Metadata). b. Recipients receive an email and should log-in to reply.



8. System Integrity Controls

Instructions: Provide design specifications for the following levels of control and any additional controls as appropriate or necessary:

- a) Internal security to restrict access of critical data items to only those access types required by users/operators*
- b) Audit procedures to meet control, reporting, and retention period requirements for operational and management reports*
- c) Application audit trails to dynamically audit retrieval access to designated critical data*
- d) Standard tables to be used or requested for validating data fields*
- e) Verification processes for additions, deletions, or updates of critical data*
- f) Ability to identify all audit information by user identification, network terminal identification, date, time, and data accessed or changed.*

After scientists submit deliverables as verifiable indicators of outputs and validate outcome indicators, the User FP/CoA Leader will review them in order to ensure that planned outputs and outcomes have been achieved. Then approves, rejects, or initiates a discussion with the leader.

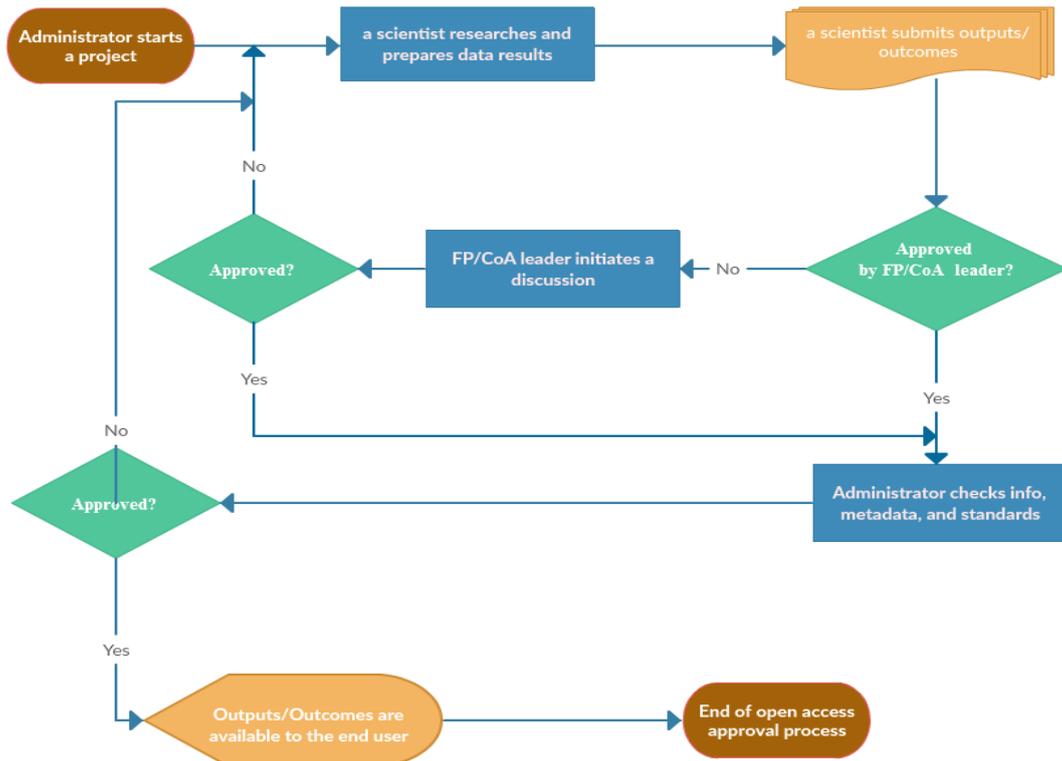
Administrators will still need to check this deliverable info, and metadata to decide on approving to give it open access approval to the DSpace Repository on <http://mel.cgiar.org/repo>.

(Fig. 50) Represents the flow of MEL integrity control system through a document submission until it get an open access approval.

Figure 92. Process of Getting an Open Access Approval for a Document

MEL

Getting an open access approval for a document deliverable





9. External Interfaces

Instructions: Describe any interfaces that exist with external systems that are not within the scope of the system, regardless whether the other systems are managed by ICARDA or another entity. Describe the electronic interface(s) between the system being designed and each of the other systems and/or subsystem(s), emphasizing the point of view of the system being designed. If there are more than one or two external systems, or if the interfaces are not simplistic, one or more separate Interface Control Documents (ICDs) should be prepared and referenced here. If applicable, identify how many ICDs exist and what they are.

MEL has its own public repository exists on <http://mel.cgiar.org/repo/>, and it is based on DSpace software.

DSpace is chosen for academic, non-profit, and commercial organizations building open digital repositories. It is free and easy to install "out of the box" and completely customizable to fit the needs of any organization.

DSpace preserves and enables easy and open access to all types of digital content including text, images, moving images, mpegs and data sets. And with an ever-growing community of developers, committed to continuously expanding and improving the software, each DSpace installation benefits from the next.

DSpace is a digital service that collects, preserves, and distributes digital material. Repositories are important tools for preserving an organization's legacy; they facilitate digital preservation and scholarly communication.

Deliverables will get public and accessible on the DSpace after both FP/CoA leader, and administrator approval.

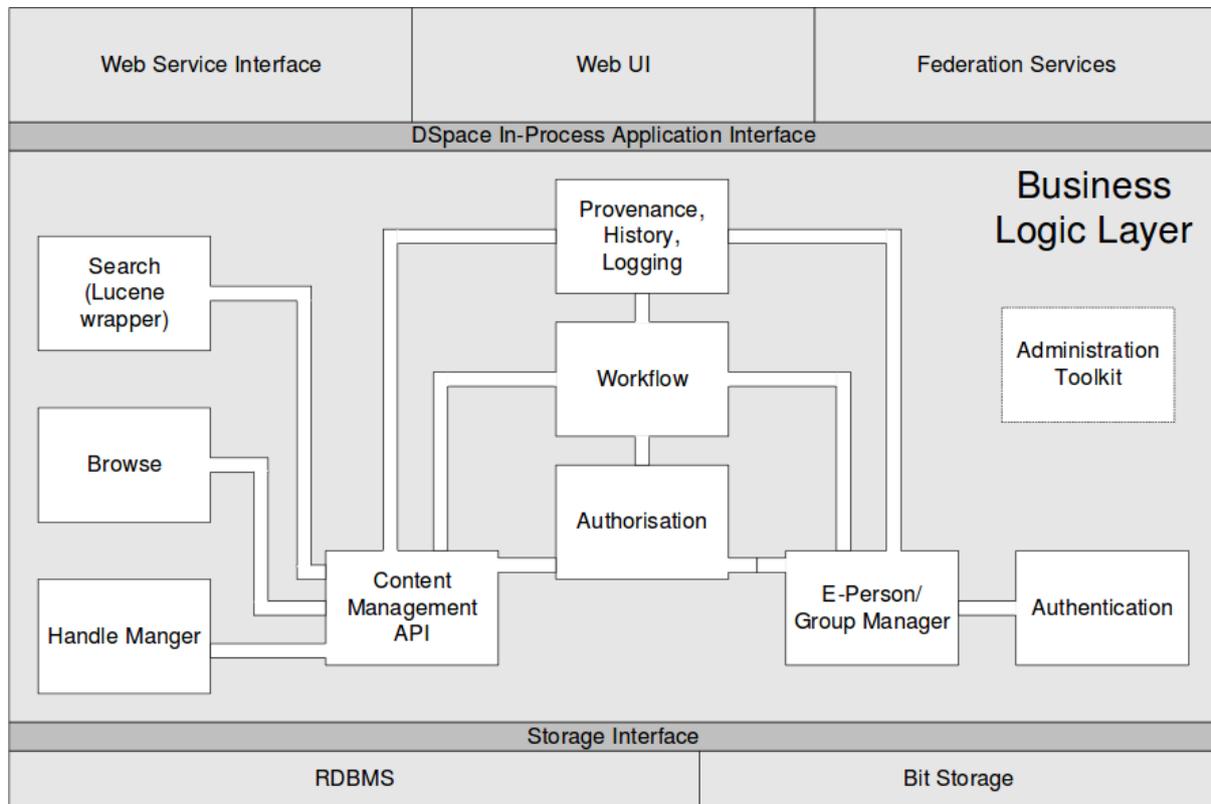
9.1 Interface Architecture

Instructions: Describe the interface(s) between the system and other systems (e.g., batch transfers, queries, etc.), indicating the location of the interfacing system. Include the interface architecture(s) being implemented (e.g., wide area networks, gateways, etc.) and the interfacing mechanisms. If remote connectivity is required, identify the method of access. Provide a diagram depicting the communications path(s) between this system and each of the other systems, which should map to the context diagram(s) provided in the Section for System Overview. The graphical representation should depict the connectivity between systems, showing the direction of data flow. Use subsections or a separate ICD(s) to address each interface independently.

Three Layers Architecture:

The DSpace platform is separated into three distinct layers. From the bottom up, these layers are the storage, business logic and service layers. (Fig. 92) Shows the DSpace system Architecture with its three distinct layers.

Figure 93. DSpace System Architecture (Bass, Stuve, & Tansley, 2002)

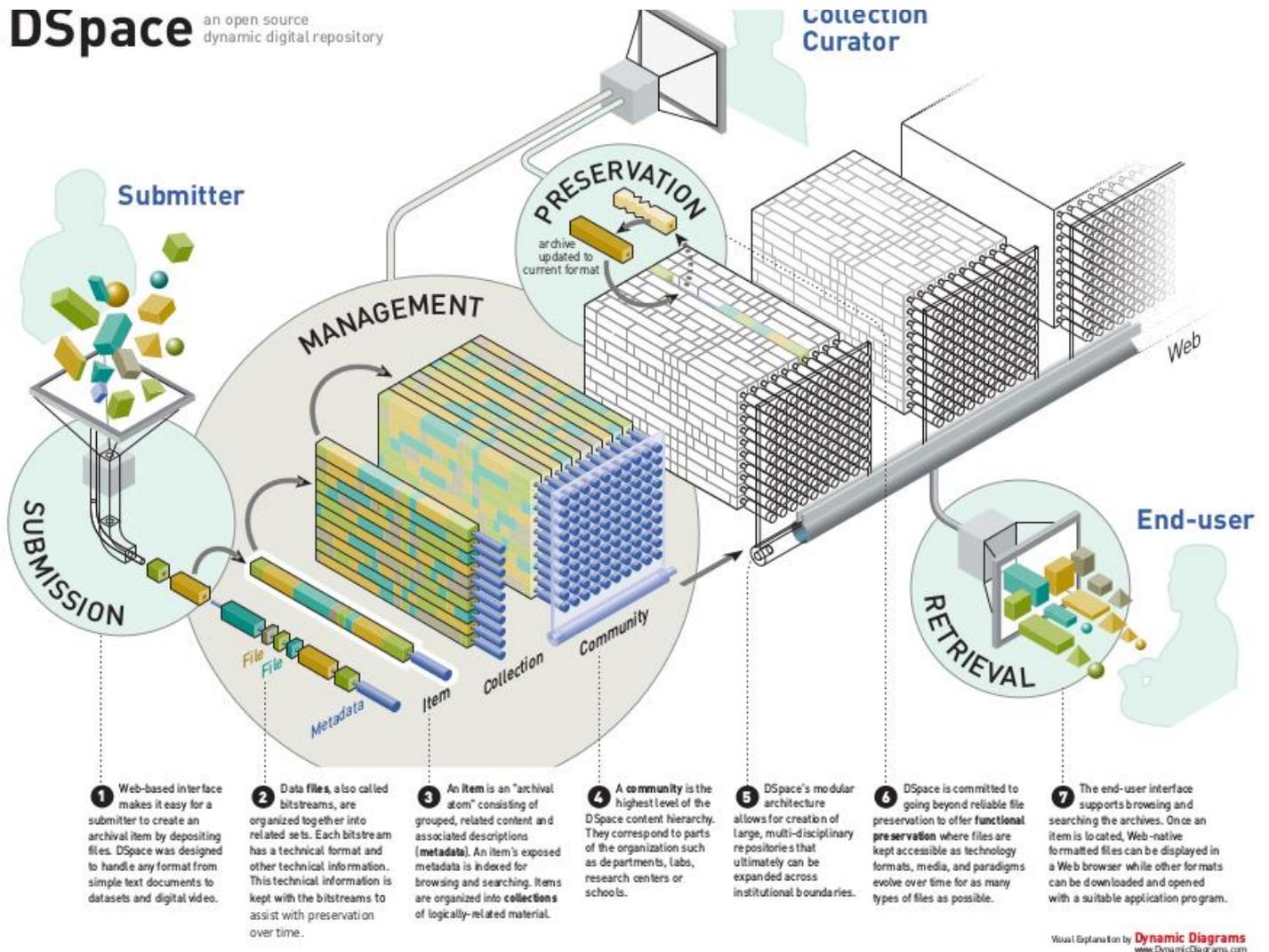


The lowest layer is the **storage layer**. This presently consists of a relational database for storing metadata and a “bitstream” storage module for storing content data. Each of these has an API accessible to the business logic layer. The union of these APIs comprises the storage interface.

The central layer contains the modules that perform the **business logic** of the system. (Fig. 92) displays the internal plumbing between these modules. Each module has a “public” API. The union of these APIs comprises the DSpace “in-process application interface.” It is on this API that services such as the Web user interface and future interoperability and federation services are built.

The top layer of the Dspace platform is the **services layer**. At present, the only implemented service is the Web user interface, though an Open Archives Initiative metadata harvesting protocol service is to be added shortly. (Fig. 93) Shows the DSpace conceptual model:

Figure 94. DSpace Conceptual Model (<http://cs.calstatela.edu>)



9.2 Interface Detailed Design

Instructions: For each external system with which the system being designed interfaces, describe the information exchange and rules governing the interface. Provide enough detailed information about the interface to correctly format, transmit, and/or receive data across the interface. Generally, this information should be documented in a separate ICD(s) that should be referenced within this section.

MEL system is synchronizing approved deliverables to the DSpace repository via application/modules/default/controllers/ReportingController.php. Another specification MEL DSpace is using that it replaces the Zend_Rest_Client with Pest; a PHP client library for RESTful web services. Pest supports the four REST verbs (GET/POST/PUT/DELETE) and pays attention to HTTP response status codes, it also has PestXML an XML-centric version of Pest, and PestJSON a JSON-centric version of Pest.

The storage layer is responsible for physical storage of metadata and content. The business logic layer deals with managing the content of the archive, users of the archive (e-people),



Monitoring, Evaluation and Learning Platform: System Design & Architecture

authorization, and workflow. The application layer contains components that communicate with the world outside of the individual DSpace installation, for example the Web user interface and the Open Archives Initiative protocol for metadata harvesting service.

Each layer only invokes the layer below it; the application layer may not use the storage layer directly, for example. Each component in the storage and business logic layers has a defined public API. The union of the APIs of those components are referred to as the Storage API (in the case of the storage layer) and the DSpace Public API (in the case of the business logic layer). These APIs are in-process Java classes, objects and methods.

It is important to note that each layer is *trusted*. Although the logic for *authorising actions* is in the business logic layer, the system relies on individual applications in the application layer to correctly and securely *authenticate* e-people. If a 'hostile' or insecure application were allowed to invoke the Public API directly, it could very easily perform actions as any e-person in the system.

The reason for this design choice is that authentication methods will vary widely between different applications, so it makes sense to leave the logic and responsibility for that in these applications. The source code is organized to cohere very strictly to this three-layer architecture. Also, only methods in a component's public API are given the *public* access level. This means that the Java compiler helps ensure that the source code conforms to the architecture.

A detailed Interface design can be found on the DSpace Resources Wiki Page:
<https://wiki.duraspace.org/display/DSPACE/DSpaceResources>

Appendix A: Acronyms

Instructions: Provide a list of acronyms and associated literal translations used within the document. List the acronyms in alphabetical order using a tabular format as depicted below.

Table 27. Acronyms

Acronym	Literal Translation
AJAX	Asynchronous JavaScript and XML.
API	Application Program Interface.
CDM	Conceptual Data Model.
CDN	Content Data Network.
COTS	Commercial Off-the-Shelf.
CSS	Cascading Style Sheet.
CapDev	Capacity Development.
CRPs	CGIAR Research Programs.
CRUD	Create, Read, Update and Delete.
DAO	Data Access Object.
DC	Developing Countries
DDD	Database Design Document.
DS	Dryland Systems.
GL	Grain Legumes.
GUI	Graphical User Interface.
HTML	Hyper Text Markup Language.
ICD	Interface Control Document.
IE	Internet Explorer.
IDO	Intermediate Development Outcomes.
LAN	Local Area Network.
LDM	Logical Data Model.
MEL	Monitoring, Evaluation & Learning framework.
MVC	Model-View-Controller.
PDM	Physical Data Model.
PHP	PHP: Hypertext Preprocessor.
OCS	One Corporate System.
RDBMS	Relational Database Management System.
RTP	Research program on Roots, Tubers, and Bananas.
SCP	Strategy and Corporate Plan.
SDD	System Design Document.
SLOC	Source Lines of Code.
SLO	System-Level Outcome.
SRF	Strategy and Results Framework.
UTF	Unicode Transformation Format.
WAN	Wide Area Network.
W1-W2	Window 1 - Window 2.
W3/bilateral	Window 3 / bilateral.
XML	Extensible Markup Language.
ZF1	Zend Framework 1.

Appendix B: Glossary

Instructions: Provide clear and concise definitions for terms used in this document that may be unfamiliar to readers of the document. Terms are to be listed in alphabetical order.

Table 28. Glossary

Term	Definition
AJAX	Client-side script that communicates to and from a server/database without the need for a postback or a complete page refresh.
API	Set of routines, protocols, and tools for building software and applications.
CDM	Map of concepts and their relationships used for databases.
CDN	Globally distributed network of proxy servers deployed in multiple data centers to deliver webpages and other Web content to a user based on the geographic locations of the user.
COTS	Products that are commercially available and can be bought.
CSS	Style sheet language used for describing the presentation of a document written in a markup language.
CapDev	The process through which individuals, organizations and societies obtain, strengthen and maintain the capabilities to set and achieve their own development objectives over time.
CRUD	The four basic functions of persistent storage.
DAO	Object that provides an abstract interface to some type of database or other persistence mechanism.
Deliverable	This is intended to be a sub-component of an output that a scientist can use to show the contribution to a specific output. An integrated set of deliverable constitute the output. The deliverable can be a document (publication) a dataset, a training/workshop report/material.
DS	Partnership of several dozen actors, including national research systems from 28 countries, universities, extension agents, civil society organizations, advanced research centers, CGIAR partners, and other development partners.
GIT	Is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency
GUI	Type of interface that allows users to interact with electronic devices through graphical icons and visual indicators such as secondary notation, as opposed to text-based interfaces, typed command labels or text navigation.
HTML	Standardized system for tagging text files to achieve font, colour, graphic, and hyperlink effects on World Wide Web pages.
ICD	The interface or interfaces between subsystems or to a system or subsystem.
Impact Pathway	Impact pathways describe these result chains, showing the linkages between the sequence of steps in getting to impact.

Term	Definition
Indicator	Indicators are the measures that help define the change at the appropriate level – whether output, outcome or impact. They are precise statements that should be SMART (Specific, Measurable, Accurate, Reliable and Time-bound).
LAN	Group of computers and associated devices that share a common communications line or wireless link to a server.
MVC	Software architectural pattern for implementing user interfaces on computers.
OCS	Software to help staff manage projects, human resources, and finances, and perform other administrative functions.
Output	Output is the products and services resulting directly and attributable from the activities undertaken. Outputs can be 'bought' in the sense that all costs associated with them should be clear and associated with the program. An output is an integrated set of deliverables for which it is possible to define a budget. The output type falls into categories defined by the CO (Technologies, Policies, Tools, Framework/Concept, Value Chain and Agro-Ecosystems assessments, and Innovation Platform).
Outcome	Outcomes are the changes (intended or realized) in the target individuals, institutions and/or systems. These changes result from the range of outputs, working together – typically mixing product-related outputs such as tools and databases, and service-related outputs around training, workshops and other fora to raise awareness, build interest and demand.
PDM	Representation of a data design which takes into account the facilities and constraints of a given database management system.
PHP	Server scripting language, and a powerful tool for making dynamic and interactive Web pages.
RDBMS	Database management system (DBMS) that is based on the relational model.
SCP	Organization's process of defining its strategy, or direction, and making decisions on allocating its resources to pursue this strategy.
SDD	Document includes the design of system components, modules, interfaces, and data for a system to satisfy specified requirements.
SLOC	Software metric used to measure the size of a computer program by counting the number of lines in the text of the program's source code.
Theory of change	A theory of change adds to an impact pathway by describing the causal assumptions behind the links in the pathways— what has to happen for the causal linkages to be realized.
UTF	Character encoding capable of encoding all possible characters, or code points, defined by Unicode.
WAN	Computer network that spans a relatively large geographical area.



Monitoring, Evaluation and Learning Platform: System Design & Architecture

Term	Definition
XML	Markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.
ZF1	Open source, object oriented web application framework for PHP 5.



Appendix C: Approvals

Instructions: List the individuals whose signatures are desired. Examples of such individuals are Business Owner, Project Manager (if identified), and any appropriate stakeholders. Add additional lines for signature as necessary.

The undersigned acknowledge that they have reviewed the System Design Document and agree with the information presented within this document. Changes to this Document will be coordinated with, and approved by, the undersigned, or their designated representatives.

Signature:	_____	Date:	_____
Print Name:	Enrico Bonaiuti		
Title:	Research Program Coordinator (DS)		
Role:	Business Owner		

Signature:	_____	Date:	_____
Print Name:	Dagmar Wittine		
Title:	Program Management Officer (RTB)		
Role:	Business Owner		

Signature:	_____	Date:	_____
Print Name:	Hashem Abed		
Title:	ITU Head (ICARDA)		
Role:	Software and Infrastructure		

Signature:	_____	Date:	_____
Print Name:	Percy Cabello		
Title:	Applications and PMO Manager (CIP)		
Role:	Software and Infrastructure		



Appendix D: Security Architecture

Instructions: Insert any related security architecture documents, including integrity controls, or provide a reference to where they are stored.

As software applications are developed with minimal security in mind, security architecture must be implemented to make sure the system is secure, there are many reasons behind that:

- Application analysts/architects mostly concentrate on the problem domain.
- Designers/developers concentrate on implementation details.
- Development teams lack security expertise.
- It is difficult to hire a fulltime security professional on the team.

When deployment dates arrive, everyone realizes the need for protecting the application by applying the security layer to it. The notion of providing a logon screen to protect the application is immature by the current security standards, because:

- Threats have multiplied and grown far more sophisticated, ranging from cyber-terrorists to industrial espionage.
- Application architectures have moved from centralized mainframes to distributed technologies.
- Security technology has improved over time.

MEL system is using a web-based technology, which make the administrators who are working on the system decentralized, because the data and services are dynamically available over the internet, this make the system vulnerable for fraud and online attacks. These threats cannot be prevented by using network security products like firewalls, routers and intrusion prevention systems, as the firewalls usually have port 80 open for use by web applications - the very port that a large number of application and system vulnerabilities take advantage of.

It is a definite need to define architecture for application security. The architecture should work as a guideline for developing security in the system. Overall, the security architecture should help ICARDA to:

- Apply the security solutions to any application, no matter what technology it uses.
- Have proper security controls in place for MEL system.
- Protect the application from threats.
- Ease the process of security administrating.
- Easily adopt to change security infrastructure.

This document will define Security Architecture for MEL system, also it will help securing ICARDA's developed applications.

MEL system users are classified into various types:

- By project: administrators, scientists, partners, registered user, and guests.
- By network: external and internal users.

There can be a mix of this classification, where an employee can be an external user if he connects from different places. The same way user can be internal if he/she is on the premises of the organization.



Various mechanisms are utilized for a user to communicate with MEL system as well as application layers to communicate with each other. Operating system inter-process communication mechanisms, like shared memory and semaphores to network protocol based means like TCP/IP, sockets, remote procedure calls, and distributed objects are being used.

D.1 Security Policy

The security policy needs to be thoroughly applied to the system. Traditionally, it has always been applied to the network to protect the resources (servers, access points, printers, etc.). Since MEL application is a resource, there is more risk at this level and more need to protect it thoroughly.

D.1.1 Security Development

As MEL system designed and developed using Zend object-oriented methodology the security life cycle for the application will have several steps to be followed as shown in the following graph:

D.1.1.1 Security Analysis

Is data and business repository for security analysis purpose available to user over communication channel. The key security concept that revolve around user is the identification, Zend framework provide that by Zend_ACL class for authorization and Zend_Auth for authentication, where **authentication** is the process that presents an identifier to the system so that the system can recognize system entities and distinguish them from other entities, **authorization** is a right or a permission that is granted to a system entity to access a system resource.

The system implements these two aspects of security through User Email (registered in the system as the USERID) and a password. To make this available for users a registration process need to be implemented to assign unique identity to every registered user to the system, once the administrators add users they give them different role according to their position. This way of assigning privileges to a user is called access control list (ACL) method.

Once a user is authenticated and is authorized to perform a business function, there may be data/control transfer between multiple module/objects which perform different logical operations to accomplish a complete task. In addition to protecting data over communication channels, protection of static data is required to provide integrity and confidentiality. For example, if the configuration files containing the connection and initialization attributes (these configurations for Zend application are set in *application.ini* file) of the system were compromised, integrity and confidentiality of the application resources could be severely compromised.

The following tasks must be accomplished in order to grant best security layer for the system:

- Establish secure connections with users by installing secure socket layer (SSL) on the web server to encrypt the transferred data between user and MEL system, especially user dashboard which contain users' information.
- Validate all values sent from the users by client-side and server-side validation to filter the entered data by users.
- Authenticate user on different roles types, to prevent normal users from accessing sensitive data.
- Separate public and restricted areas by specifying allowed sections for users.
- Use account lockout policy for End-User accounts.
- Support password expiration periods.



- Make the ability to disable user accounts.
- Store user passwords in different table than user table.
- Require strong passwords.
- Protect authentication cookies using encrypted and secure communication layer.
- Limit http session lifetime to mitigate the risk of session hijacking and replay attacks.

D.1.1.2 Cryptography

Is the art or science encompassing the principles and methods of transforming an intelligible message into one that is unintelligible and then retransforming that message back to its original form, this implemented in MEL system after getting the credentials for user when being registered and add salt (which is a generated random) string being added to the password and encrypt the both string using SHA-1 then stored in the database under password field and the salt string stored under salt field. When the user is logging in the password will be combined with the salt field value and encoded then Zend_auth class will compare the two encoded values if they are equal the user will be allowed to access the system.

The best practices to use cryptography in the best way as follow:

- Use cryptographic algorithms and routines used and tested before, like cryptographic services provided by the platform. This includes the Zend Framework and the underlying operating system. Do not develop custom implementations because these frequently result in weak protection.
- Use the correct algorithm and correct key size, in the following list it shows algorithm types and its key size, the larger key size the more security:
 - Data Encryption Standard (DES) 64-bit key (8 bytes).
 - TripleDES 128-bit key or 192-bit key (16 or 24 bytes).
 - TripleDES 128-bit key or 192-bit key (16 or 24 bytes).
 - TripleDES 128-bit key or 192-bit key (16 or 24 bytes).
- Secure encryption keys which are secret numbers used as input to the encryption and decryption processes, by protecting the storage that contains them.
- Log detailed error messages, send detailed error messages to the error log and send minimal information to the user.
- Secure log files by using Windows ACLs and restrict access to the log files. Authorize access only to highly trusted accounts such as administrators.

D.1.1.3 Risk Assessment

System need to be assessed at the business level to ascertain the risk based on information compromise, unauthorized access and availability for determining the security level that needs to be assigned to it. After developing the system and functionally tested and before deploying it to production environment, a security risk assessment need to be performed with assurance test.

This test will help ensure the total system is in compliance based on the security level assigned to it. These tests need to be mandated on all the developing lifecycle, newly developed modules or changing to the system after deployment. Risk checklists need to be developed to assure that proper security controls have been placed at the appropriate locations within the system. The



checklist must be updated at a regular interval to accommodate newer technologies and threats. The checklist should contain all aspects of logical access for various security levels including:

- User identification (registration process).
- Authentication (level, password strength, sign-on attempts, account lockout policies, processes to forgot password, session tracking) authorization.
- Authorization.
- Encrypting and hashing.

Apart from the system tests any system software such as operating system, web application server (XAMPP) and RDBMS (MySQL) need to be assessed and patched to the latest security compliant level.

System contingency plans should be reviewed to make all backup and recovery plans are up to date so that there is no disruption of system availability.

D.1.2 Security Infrastructure

The infrastructure needs to be interoperable with the system and be maintained by a team that can keep pace with the latest standards. Accomplishing the above and offering security as a centralized component can be a tedious and painful task. But the results for that deserve it and cause cost savings and controlled environment. This centralized infrastructure should:

- Authentication, authorization, integrity and audit components.
- Stick to industry standards.
- Scalability.
- Easily manageable.

D.2 Security Analysis

Table 29. MEL System Analysis

Characteristic	Value	Description
Name	MEL (Monitoring, Evaluation and Learning)	Platform
Number of Users	256	Number of registered users
Type	Client/Server	Web based
Software Used	PHP, MySQL, XAMPP	Software Server, Database
OS	Windows	Windows 2012 r2
Identity Source	Database	Adding users
Identity Used	User email and password	User credentials
Network	Internet	Type of communication
Data Classification	Restricted access	Restricted, Non-restricted
User Access	Custom Client	IE, Chrome, Firefox

Attacks that the System can be vulnerable to:

- **Cross-Site Scripting (XSS):** which enables attackers to inject client-side scripts into the system and it may be used by attackers to bypass access controls such as the same-origin policy.



- **Injection Flaws:** allow attackers to relay malicious code through the system to another system. These attacks include calls to the operating system via system calls, the use of external programs via shell commands, as well as calls to backend databases via SQL (like SQL injection).
- **File Inclusion Vulnerabilities:** is a vulnerability that occurs due to user input or uploads to the system not being properly handled or poor data validation by the system.
- **Cross-site request forgery:** is a type of malicious exploit of the system where unauthorized commands are transmitted from a user that the system trusts through login cookies.
- **Insecure Cryptographic Storage:** is a common vulnerability that occurs when sensitive data is not stored securely. Protecting sensitive data by encrypting it should be a highly considered when storing user data.

Appendix E: Performance

Instructions: Insert any performance documents or provide a reference to where they are stored.

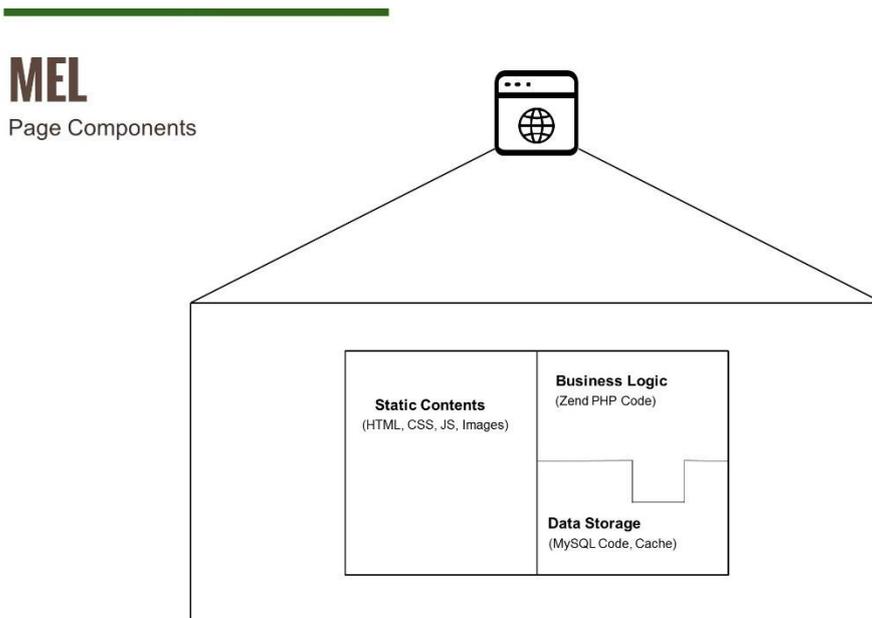
Performance and scalability are key principles that affect the system after increase in users, which in turn increases the amount of data, processing power and other resources needed to keep MEL system running smoothly.

The topics of performance and scalability are multidimensional, as they need to be followed in different levels of the system (from the operating system, programming language, web server, database or other parts). And can also be addressed in various ways, from simple changes made to source code or configuration files, to deep design choices.

E.1 Performance

When the system renders a page at the end user browser all the above components will collaborate together to generate the view.

Figure 95. Page Components



To optimize the system for best performance each of these levels need to be handled separately the following table illustrates the implied technologies for each level:

Table 30. Web Application Levels, Core Technologies, and Performance and Scalability Techniques

Level	Static Content	Business Logic	Storage
Used Technologies	<ol style="list-style-type: none"> Client-side languages (HTML, JavaScript, CSS) Web servers Browsers 	<ol style="list-style-type: none"> Server-side programming/web frameworks/scripting languages (PHP and Zend Framework) Web servers / Application servers Operating systems 	<ol style="list-style-type: none"> Data storage engines (RDBMS, Column orientated) Data access mechanisms (SQL, ORM) Operating systems
Performance Techniques	<ol style="list-style-type: none"> Compression (HTML, JavaScript, CSS) Tuning/profiling (JavaScript) HTTP headers Client-side (Browser) caching Web server tuning 	<ol style="list-style-type: none"> Tuning/profiling (PHP) Refactoring (PHP) Server-side caching Web server tuning Operating system tuning 	<ol style="list-style-type: none"> Tuning/profiling (RDBMS) Indexes Operating system tuning

E.1.1 Static Content

This is the top most level of the system, which displays information related to the system and user interactions. In this level the system needs to render pages in a high speed and load time. This can be accomplished by the following techniques, which are related to this level and optimize Zend view script code.

- Compress and combine related files with the same type together (JS, CSS).
- Compress html output reduces 10%-20% of the loading time for the page.
- As the system is using apache two of its modules, indicated in the following paragraph, can help improve the performance:
 - mod_deflate and Mod_gzip when these modules are enabled and add the following directive to the .htaccess file they will compress the response for any request sent to MEL system with gzip compression:

```

<IfModule mod_gzip.c>
    mod_gzip_on          Yes
    mod_gzip_dechunk    Yes
    mod_gzip_item_include file      \.(html?|txt|css|js|php|pl)$
    mod_gzip_item_include handler  ^cgi-script$
    mod_gzip_item_include mime     ^text/*
    mod_gzip_item_include mime     ^application/x-javascript.*
    mod_gzip_item_exclude mime     ^image/*
    mod_gzip_item_exclude rspheder ^Content-Encoding:.*gzip.*
</IfModule>

```

- Web Caching to store the contents from previous requests this increase the experience of the system. The benefits of caching:
 - Decrease network cost.
 - Improve system responsiveness.
 - Increase performance on the server.
 - Availability of content during network interruptions.

To enable web caching enable mod_expires on the apache server and add the following directive to the .htaccess file:

```

<IfModule mod_expires.c>
    ExpiresActive On
    ExpiresByType image/jpg "access plus 1 year"
    ExpiresByType image/jpeg "access plus 1 year"
    ExpiresByType image/gif "access plus 1 year"
    ExpiresByType image/png "access plus 1 year"
    ExpiresByType text/css "access plus 1 month"
    ExpiresByType application/pdf "access plus 1 month"
    ExpiresByType text/x-javascript "access plus 1 month"
    ExpiresByType application/x-shockwave-flash "access plus 1 month"
    ExpiresByType image/x-icon "access plus 1 year"
    ExpiresDefault "access plus 2 days"
</IfModule>

```

- Specify displayed images dimension, this technique allows for faster rendering by eliminating the need for unnecessary reflows and repaints.
- Host static contents on content delivery network server (a server cdn uses to store content in many locations so content is geographically/physically closer to users, resulting in faster performance).



E.1.2 Business Logic

This level controls the system functionality by performing dynamic content processing, also it controls the flow of the data.

The following steps describe how to increase performance for this level:

- Follow the best practices of Zend framework coding:
 - Reduce the usage of `include_path` and use absolute path instead.
 - Define zend framework path at early stage.
 - Speed up components plugins loading by using plugin loader.
 - Use `zend_db_table` and metadata cache as `zend_db_table` optionally utilize `zend_cache` to cache table metadata.
 - Speed up resolution of view helpers by defining custom helper methods in separate classes and calling them as if they were direct methods of `zend_view`.
- Server-side caching: by use file caching for none modified data, which renders the output of requests to the database in a template, and store it in a separate cache folder, this folder will be cleaned when data is modified in the backend so the user can see the latest data stored in the system.

E.1.3 Storage

Data storage, comprising both data sets and the database management system that manages and provides access to the data.

Below are techniques to optimize storage level:

- Indexing data store: Using an index to access the stored data quickly, this is the most well-known performance technique when it comes to database.
- Fine tune database server: increasing buffer size, concurrent user access, increase query cache size so the long queries will result in less time to read queries from database.

E.2 Scalability

Scalability can refer to many different parameters of the system:

- How much additional traffic can it handle?
- How easy to add more storage capacity? or even how many more transactions can be processed?

To make the system scalable, there are two types of scaling the hardware:

- Scaling up: this can be achieved by increasing the number of cpu, ram, or hard disk storage.
- Scaling out: this can be achieved by increasing the number of server hardware.



CGIAR



RESEARCH
PROGRAM ON
Dryland Systems



RESEARCH
PROGRAM ON
Dryland Cereals



RESEARCH
PROGRAM ON
Grain Legumes



RESEARCH
PROGRAM ON
Roots, Tubers
and Bananas



ICARDA

Science for Better Livelihoods in Dry Areas



Code**bia**



powered by
amazon
web services