November 11, 2016

Verbat Logo

Review & Recommendations for Application Development of the Construction Project Management Application



**IEAG Digital**

Construction Project Management

**Revision** **History**

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Version | Description | Author |
| 09 Nov 2016 | 1.0 | Draft Version | Prashant Thomas |
|  |  |  |  |
|  |  |  |  |

**Document Approvals**

|  |  |  |
| --- | --- | --- |
| Name | Role | Signature |
|  |  |  |
|  |  |  |

Table of Contents

[Introduction 6](#_Toc466644041)

[Software Requirements for the Admin Module 6](#_Toc466644042)

[Missing use cases 6](#_Toc466644043)

[Non Functional Requirements 7](#_Toc466644044)

[Super User 7](#_Toc466644045)

[Background Jobs and Scripts 8](#_Toc466644046)

[Auditing & Logging 8](#_Toc466644047)

[Database Design and Application logic 8](#_Toc466644048)

[Lack of Actor Perspective 8](#_Toc466644049)

[Document tracking and versioning 8](#_Toc466644050)

[Application (Source Code) Review 9](#_Toc466644051)

[Language issues 9](#_Toc466644052)

[Weak typing 9](#_Toc466644053)

[Exceptions and error handling 10](#_Toc466644054)

[Unhelpful built-ins 11](#_Toc466644055)

[Framework issues 11](#_Toc466644056)

[URL routing 11](#_Toc466644057)

[Input handling 12](#_Toc466644058)

[Untrusted Data 14](#_Toc466644059)

[Use of $\_Request 15](#_Toc466644060)

[Never Concatenate or Interpolate Data in SQL 15](#_Toc466644061)

[Escaping is not safe 17](#_Toc466644062)

[Use Prepared Statement 17](#_Toc466644063)

[ORM 18](#_Toc466644064)

[Encoding Issues 19](#_Toc466644065)

[Shell Injection 19](#_Toc466644066)

[Code Injection 20](#_Toc466644067)

[Hashing Passwords. 20](#_Toc466644068)

[Authentication and Session Management 21](#_Toc466644069)

[Session Management 21](#_Toc466644070)

[Session Hijacking Prevention 21](#_Toc466644071)

[Invalidate Session ID 21](#_Toc466644072)

[Rolling of Session ID 21](#_Toc466644073)

[Exposed Session ID 22](#_Toc466644074)

[Session Fixation 22](#_Toc466644075)

[Session Expiration 22](#_Toc466644076)

[Cookies 23](#_Toc466644077)

[Authentication 25](#_Toc466644078)

[Helpful Tips 25](#_Toc466644079)

[Overriding Server Variables 25](#_Toc466644080)

[File inclusion vulnerabilities: 26](#_Toc466644081)

[Database Code Review 27](#_Toc466644082)

[Primary Key 27](#_Toc466644083)

[Naming convention 27](#_Toc466644084)

[Table relationships 27](#_Toc466644085)

[Auditing and logging 27](#_Toc466644086)

[Nulls in Mandatory Columns 27](#_Toc466644087)

[Restrict field size 27](#_Toc466644088)

[Column and Table Descriptions 28](#_Toc466644089)

[Index for commonly referenced fields 28](#_Toc466644090)

[Stored Procedures 28](#_Toc466644091)

[Initialize Default values appropriately 28](#_Toc466644092)

[Smoke Test Results 29](#_Toc466644093)

# Introduction

This document provides coverage of the review and recommendations suggested by Verbat Technologies towards the completion of the application development for the “Construction Project Management – Admin Module” undertaken by IEAG.

About this document

This document is divided into four separate sections, each of which details a specific aspect of the application development cycle for the Admin module. These sections are

1. Review & Recommendations of Requirements documentation
2. Review & Recommendation of Application Code
3. Review & Recommendation of MSSQL Database
4. Review & Results of Smoke testing the Admin Panel

# Software Requirements for the Admin Module

## Missing use cases

* Use cases are missing along with any kind of description for the requirements.
* The prototype screens available in the Requirements are actual screen shots of the developed application.
* Requirements are not marked by a requirement ID. Requirement ID helps to track the requirements against the test cases and use cases and allow for accountability in the development process
* Requirement ID’s help in rolling out the requirements traceability matrix as well as the work Breakdown Structure (WBS)
* Use cases should be depicted either by Use-Case diagrams or use case descriptions similar to the ones below

|  |  |  |  |
| --- | --- | --- | --- |
| **REQUIREMENT ID (RID)** | **DESCRIPTION** | [**PRIORITY LEVELS**](file:///C:\Users\Verbat\Desktop\pmp\requirements\Requirements_Gathering_Guidelines_and_Checklists_2.xls#'Document Conventions'!_Toc136410958) | [**EFFORT REQUIRED**](file:///C:\Users\Verbat\Desktop\pmp\requirements\Requirements_Gathering_Guidelines_and_Checklists_2.xls#'Document Conventions'!_Toc136410958) |
| **(P1, P2, P3, P4)** | **(H,M,L)** |
| RID #1  RID #2  (Related requirements if needed) | A description of the requirement, along with mandatory and non-mandatory fields. (clearly identify the user and   * Field1 * Field2   Validation summary indicated expected actions and error conditions. | Priority | Effort required |

Typically this would be followed up with a wireframe of the screen described

## Non Functional Requirements

The requirements document has no coverage for Nonfunctional requirements

Namely

* Performance (Performance, Safety, and Reliability),
* Quality (Availability, Security, and Usability)
* Constraints on quality ( Budget, Schedule , scope)

## Super User

There is no documentation on how the super user is created.

## Background Jobs and Scripts

If the application is using background jobs for routine maintenance or running scripts for mitigating errors, these should be documented separately and duly noted in the requirements.

## Auditing & Logging

There were no requirements listed for auditing and error logs. Audit logs should tell you who did what (i.e. the logged in user) while application logs will tell you when the application issues warning or error messages.

## Database Design and Application logic

Any description related to the DB design or Application logic (not business logic) should be detailed in the Low level or high level design specifications rather than the requirement specification.

## Lack of Actor Perspective

No actors are described in the requirements. Without actors the context is unclear and does not help in creating test cases. Inclusion of actors helps in ironing out the answers to questions such as Why, Where & What about the requirement

## Document tracking and versioning

The SRS does not have Version Tracking and Author. Although the documents states that it is at revision one, It does not indicate how many revisions the document has gone through, nor does it indicate who authored and revised the document. This is important to track how the requirements got changed and who made those changes (for what reason).

# Application (Source Code) Review

An in depth analysis of the source code was not conducted because of limitations in time. Instances of application code/logic issues are cited below as an example that could be applied to the rest of the application. After reviewing the source code, we recommend that the following changes. Please not that some suggestions were made based on potential future functionality as it may not have been implemented yet.

## Language issues

### Weak typing

PHP is weakly typed, which means that it will automatically convert data of an incorrect type into the expected type. This feature very often masks errors by the developer or injections of unexpected data, leading to vulnerabilities (see “Input handling” below for an example).

Use functions and operators that do not do implicit type conversions (e.g. === and not ==). Not all operators have strict versions (for example greater than and less than), and many built-in functions (like in\_array) use weakly typed comparison functions by default, making it difficult to write correct code.

#### Example

In the source file manage-project.php

if($op == "add")

Instead do this

if($op === "add")

### Exceptions and error handling

Many PHP libraries, do not use exceptions, but instead report errors in other ways (such as via notices) that allow the faulty code to carry on running. This has the effect of masking many bugs. In many other languages, and most high level languages that compete with PHP, error conditions that are caused by developer errors, or runtime errors that the developer has failed to anticipate, will cause the program to stop running, which is the safest thing to do.

Consider the following code in include\_file.php

error\_reporting(0);

Instead do this

try {

//code

} catch (Exception $e) {

//code after exception caught

echo 'Caught exception: ', $e->getMessage(), "\n";

}

### Unhelpful built-ins

PHP comes with many built-in functions, such as addslashes, mysql\_escape\_string and mysql\_real\_escape\_string, that appear to provide security, but are often buggy and, in fact, are unhelpful ways to deal with security problems. Some of these built-ins are being deprecated and removed, but due to backwards compatibility policies this takes a long time.

For example consider the code from manage-project.php

addslashes(trim($\_REQUEST['title']))

Instead do this

Use trim($\_REQUEST['title']) and fetch the value only after proper validation.

## Framework issues

### URL routing

The application uses PHP’s built-in URL routing mechanism. This opens up several *vulnerabilities*:

* Remote execution vulnerability for every file upload feature that does not sanitize the filename. (In other words, the web server executes something instead of serving it). Ensure that when saving uploaded files, the content and filename are appropriately sanitized.
* Source code, including Config files, are stored in publicly accessible directories along with files that are meant to be downloaded (such as static assets). Misconfiguration (or lack of configuration) can mean that source code or Config files that contain secret information can be downloaded by attackers. (In other words, the web server serves a resource which should have been private or executable only). You can use .htaccess to limit access. This is not ideal, because it is insecure by default, but there is no other alternative.
* The URL routing mechanism is the same as the module system. This means it is often possible for attackers to use files as entry points which were not designed as such. This can open up vulnerabilities where authentication mechanisms are bypassed entirely - a simple refactoring that pulls code out into a separate file can open a vulnerability. This is made particularly easy in PHP because it has globally accessible request data ($\_GET etc), so file-level code can be imperative code that operates on the request, rather than needing request handling code to be within function definitions.
* The lack of a proper URL routing mechanism often leads to developers creating their own ad-hoc methods. These are often insecure and fail to apply appropriate authorization restrictions on different request handling functionality.

### Input handling

Instead of treating HTTP input as simple strings, PHP will build arrays from HTTP input, at the control of the client. This can lead to confusion about data, and can easily lead to security bugs.

For example, In the source file index.php, Input is directly applied in the query.

if ($\_REQUEST['btnSubmit']) {

$usedID = $\_REQUEST['user\_name'];

$password = $\_REQUEST['password'];

if (trim($usedID) != "" && trim($password) != "") {

$sql = "SELECT \* FROM [dbo].[admin\_user] WHERE username='" . $usedID . "' AND password = '" . $password . "' ";

$usrArr = $db->fetch\_table\_Values($sql);

if ($usrArr[0]['id']) {

$\_SESSION['ID'] = $usrArr[0]['id'];

$\_SESSION['UserName'] = $usrArr[0]['username'];

$\_SESSION['Name'] = $usrArr[0]['fname'] . " " . $usrArr[0]['lname'];

header('Location: dashboard.php');

exit();

} else {

$error\_mgs = "Invalid username or password";

}

} else {

$error\_mgs = "Please enter username or password";

}

}

Instead do this

Use the input data in the query after proper validation and filtering.

### Untrusted Data

All data that is a product, or sub product, of user input is to NOT be trusted. They have to either be validated, using the correct methodology, or filtered, before considering them untainted. This is different from data validation at the front end (i.e. input handling, as discussed before). In instances where the user was able to override the input validation maliciously; That input maybe passed directly to a database query

Consider the code from manage-project.php

$DataArr['currency'] = addslashes(trim($\_REQUEST['currency']));

$DataArr['is\_active'] = addslashes(trim($\_REQUEST['Status']));

if($op == "add"){

$DataArr['sys\_created\_by'] = $\_SESSION['ID'];

$DataArr['sys\_create\_date'] = date('Y-m-d H:i:s');

$inserted\_id = $db->insert\_tableValues('[dbo].[project]', $DataArr);

if($inserted\_id){

header("Location:manage-project.php");

exit();

}

}

Instead do this

Use the input data in the query after proper validation and filtering.

### Use of $\_Request

Using $\_REQUEST is strongly discouraged. This super global is not recommended since it includes not only POST and GET data, but also the cookies sent by the request. All of this data is combined into one array, making it almost impossible to determine the source of the data. This can lead to confusion and makes your code prone to mistakes, which could lead to security problems.

Consider the code from manage-project.php

$DataArr['title'] = addslashes(trim($\_REQUEST['title']));

Instead do this

Specify $\_GET, $\_POST or $\_COOKIE instead of $\_REQUEST. Here the data arrived as POST so use

$DataArr['title'] = addslashes(trim($\_POST['title']));

### Never Concatenate or Interpolate Data in SQL

Never build up a string of SQL that includes user data, either by concatenation:

$sql = "SELECT \* FROM users WHERE username = '" . $username . "';";

or interpolation, which is essentially the same:

$sql = "SELECT \* FROM users WHERE username = '$username';";

If '$username' has come from an untrusted source (and you must assume it has, since you cannot easily see that in source code), it could contain characters such as ' that will allow an attacker to execute very different queries than the one intended, including deleting your entire database etc. Using prepared statements and bound parameters is a much better solution.

Consider the code in index.php

$sql = "SELECT \* FROM [dbo].[admin\_user] WHERE username='" . $usedID . "' AND password = '" . $password . "' ";

Instead do this

Use prepared statement.

$conn = sqlsrv\_connect( $serverName, $connectionInfo);

if( $conn === false) {

die( print\_r( sqlsrv\_errors(), true));

}

$sql = "UPDATE Table\_1

SET OrderQty = ?

WHERE SalesOrderID = ?";

// Initialize parameters and prepare the statement.

// Variables $qty and $id are bound to the statement, $stmt.

$qty = 0; $id = 0;

$stmt = sqlsrv\_prepare( $conn, $sql, array( &$qty, &$id));

### Escaping is not safe

mysql\_real\_escape\_string is not safe. Don't rely on it for your SQL injection prevention.When you use mysql\_real\_escape\_string on every variable and then concat it to your query, *you are bound to forget that at least once*, and once is all it takes. You can't force yourself in any way to never forget. In addition, you have to ensure that you use quotes in the SQL as well, which is not a natural thing to do if you are assuming the data is numeric, for example. Instead use prepared statements, or equivalent APIs that always do the correct kind of SQL escaping for you. (Most ORMs will do this escaping, as well as creating the SQL for you).

It appears that mysql\_real\_escape\_string is not used in this application. This is mentioned here for reference purposes.

### Use Prepared Statement

Prepared statements are very secure. In a prepared statement, data is separated from the SQL command, so that everything user inputs is considered data and put into the table the way it was. Please note that sometimes prepared statements do not work: The problem is, when you need to build dynamic queries, or need to set variables not supported as a prepared variable, or your database engine does not support prepared statements.

It appears that this application does not use prepared statements.

Use prepared statement.

$conn = sqlsrv\_connect( $serverName, $connectionInfo);

if( $conn === false) {

die( print\_r( sqlsrv\_errors(), true));

}

$sql = "UPDATE Table\_1

SET OrderQty = ?

WHERE SalesOrderID = ?";

// Initialize parameters and prepare the statement.

// Variables $qty and $id are bound to the statement, $stmt.

$qty = 0; $id = 0;

$stmt = sqlsrv\_prepare( $conn, $sql, array( &$qty, &$id));

### ORM

ORM’s are not used in this application. Consider using an ORM. ORMs (Object Relational Mappers) are a good security practice. If you're using an ORM (like Doctrine) in your PHP project, you're still prone to SQL attacks. Although injecting queries in ORM's is much harder, keep in mind that concatenating ORM queries makes for the same flaws that concatenating SQL queries, so NEVER concatenate strings sent to a database. ORM's support prepared statements as well.

### Encoding Issues

Many new attack vectors rely on encoding bypassing. Use UTF-8 as your database and application charset unless you have a mandatory requirement to use another encoding.

$DB = new mysqli($Host, $Username, $Password, $DatabaseName);

if (mysqli\_connect\_errno())

trigger\_error("Unable to connect to MySQLi database.");

$DB->set\_charset('utf8');

Please note that the issues listed below has not been identified as issues, because functionality related to these has not been implemented in the application. Hence issues listed below should be used for reference in future development

### Shell Injection

A few PHP functions namely shell\_exec, exec, passthru, system , [backtick operator](http://no2.php.net/manual/en/language.operators.execution.php) ( ` ) run a string as shell scripts and commands. Input provided to these functions (specially backtick operator that is not like a function). Depending on your configuration, shell script injection can cause your application settings and configuration to leak, or your whole server to be hijacked. This is a very dangerous injection and is somehow considered the haven of an attacker.

Never pass tainted input to these functions - that is input somehow manipulated by the user - unless you're absolutely sure there's no way for it to be dangerous (which you never are without whitelisting). Escaping and any other countermeasures are ineffective, there are plenty of vectors for bypassing each and every one of them; don't believe what novice developers tell you.

### Code Injection

All interpreted languages such as PHP, have some function that accepts a string and runs that in that language. In PHP this function is named eval(). Using eval is a very bad practice, not just for security. If you're absolutely sure you have no other way but eval, use it without any tainted input. Eval is usually also slower.

Function preg\_replace() should not be used with unsanitised user input, because the payload will be [eval()'ed](http://stackoverflow.com/a/4292439).

preg\_replace("/.\*/e","system('echo /etc/passwd')");

Reflection also could have code injection flaws. Refer to the appropriate reflection documentations, since it is an advanced topic

### Hashing Passwords.

No hashing is used in this application. Password is directly checked without encoding.

See code from index.php

$sql = "SELECT \* FROM [dbo].[admin\_user] WHERE username='" . $usedID . "' AND password = '" . $password . "' ";

Instead of this

$sql = "SELECT \* FROM [dbo].[admin\_user] WHERE username='" . $usedID . "' AND password = '" . md5($password). "' ";

## Authentication and Session Management

### Session Management

The default session management using PHP session is implemented in the application. Sessions can be managed better by implementing frameworks.

### Session Hijacking Prevention

The application has not made use of storing the client IP to prevent session hijacking. To implement Session Hijacking Prevention, simply store the client IP in the session first time it is created, and enforce it to be the same afterwards. The code snippet below returns client IP address:

$IP = getenv ( "REMOTE\_ADDR" );

### Invalidate Session ID

In case of session hijack attempt, there is no implementation to invalidate the session nor is there any mechanism to log the events.

In practice you should invalidate (unset cookie, unset session storage, remove traces) of a session whenever a violation occurs (e.g 2 IP addresses are observed). A log event would prove useful. Many applications also notify the logged in user via their email address (Eg Gmail)

### Rolling of Session ID

Rolling of session ID is not implemented. You should roll session ID whenever elevation occurs, e.g when a user logs in, the session ID of the session should be changed, since its importance is changed.

### Exposed Session ID

No exposing session ID in this application. Which is good. Transfer session ID over TLS whenever session holds confidential information, otherwise a passive attacker would be able to perform session hijacking.

### Session Fixation

[session\_regenerate\_id()](http://www.php.net/session_regenerate_id) is not implemented. Invalidate the Session id after user login (or even after each request) with [session\_regenerate\_id()](http://www.php.net/session_regenerate_id).

### Session Expiration

The application appears to not expire sessions. Session available until user closes the browser.

Ideally session should expire after a certain amount of inactivity, and after a certain time of activity as well. The expiration process means invalidating and removing a session, and creating a new one when another request is met.

Also keep the log out button close, and unset all traces of the session on log out.

* Inactivity Timeout
  + Expire a session if current request is X seconds later than the last request. For this you should update session data with time of the request each time a request is made. The common practice time is 30 minutes, but highly depends on application criteria.
  + This expiration helps when a user is logged in on a publicly accessible machine, but forgets to log out. It also helps with session hijacking.
* General Timeout
  + Expire a session if current session has been active for a certain amount of time, even if active. This helps keeping track of things. The amount differs but something between a day and a week is usually good. To implement this you need to store start time of a session.

### Cookies

It appears that cookies are not used in this application. Use cookies if necessary. Cookies are preferred when you need to store long-term information/values, such as user's account (so that even when they shut down the computer for 2 days, their account will still be logged in). If the application is accessed generally from the intranet, cookies would be more ideal to sessions. So It depends on the context.

Sample code for initializing and using cookies.

Setting new cookie

=============================

<?php

setcookie("name","value",time()+$int);

/\*name is your cookie's name

value is cookie's value

$int is time of cookie expires\*/

?>

Getting Cookie

=============================

<?php

echo $\_COOKIE["your cookie name"];

?>

Updating Cookie

=============================

<?php

setcookie("color","red");

echo $\_COOKIE["color"];

/\*color is red\*/

/\* your codes and functions\*/

setcookie("color","blue");

echo $\_COOKIE["color"];

/\*new color is blue\*/

?>

Deleting Cookie

==============================

<?php

unset($\_COOKIE["yourcookie"]);

/\*Or\*/

setcookie("yourcookie","yourvalue",time()-1);

/\*it expired so it's deleted\*/

?>

### Authentication

Remember me functionality is not implemented.

If you add remember me functionality it will be better. You should use this strategy.

1. When the user successfully logs in with Remember Me checked, a login cookie is issued in addition to the standard session management cookie.
2. The login cookie contains a series identifier and a token. The series and token are un-guessable random numbers from a suitably large space. Both are stored together in a database table, the token is hashed (sha256 is fine).
3. When a non-logged-in user visits the site and presents a login cookie, the series identifier is looked up in the database.
   * If the series identifier is present and the hash of the token matches the hash for that series identifier, the user is considered authenticated. A new token is generated, a new hash for the token is stored over the old record, and a new login cookie is issued to the user (it's okay to re-use the series identifier).
   * If the series is present but the token does not match, a theft is assumed. The user receives a strongly worded warning and all of the user's remembered sessions are deleted.
   * If the username and series are not present, the login cookie is ignored.

## Helpful Tips

### Overriding Server Variables

Once variables are created, they can be referenced from anywhere in the PHP web page. This is so convenient that most PHP programmers do not bother to initialize them. PHP web pages were built primarily to process inputs from the web browser. Any user input passed as GET/POST/cookie parameter automatically becomes a PHP script variable at the server. This means, a remote attacker can create or override any PHP script variable at the server. This effectively gives an attacker control over server-side processing.

### File inclusion vulnerabilities:

PHP being a function-rich scripting language, allowed functions like include(), require(), fopen(), etc., to parse PHP code libraries when required.

A common vulnerability found in many PHP web applications occurs, when PHP programmers tend to pass the PHP script page or library as user input. In the following URL www.domain.com/somepage.php?page=header.php the programmer is trying to use a common header across all pages in the website. He has modularly coded it in another PHP script header.php. The security implications of this can be understood when we realize that URL parameter “page” can be controlled by a remote attacker, who may execute malicious PHP codes by “including” a PHP file hosted on his remote server.

Performing security code reviews of PHP applications and ensuring that secure coding practices are followed helps organizations develop secure and reliable web applications.

# Database Code Review

Based on the initial survey there are 10 tables in the admin module. The tables were evaluated for

## Primary Key

All the tables are given primary keys. Tables with a parent child relationship are associated with foreign keys. Furthermore all tables have been defined with indexes (clustered and non-clustered)

## Naming convention

All the database objects adhered to a common naming convention. Tabled were well named as were there columns.

## Table relationships

The relationship between the different tables were human friendly and easy to comprehend. The relationships between the tables were fairly regular and well established.

## Auditing and logging

All of the tables have representative columns to hold audit and log information.

## Nulls in Mandatory Columns

It is recommended not to allow nulls for mandatory fields in the database (eg:- user\_name in User table). User Name may be a mandatory field in the application but the database field allow null values. It is suggested that in cases where null values exist, that it should default to a predetermined value.

## Restrict field size

Use appropriate size for the fields to match the data length (eg:- Phone\_no in the user table is 200 characters long. Typically phone\_no does not require more than 15 characters.)The database size can be reduced by using actual type and size for the fields. As the database grows having right sized fields will improve the performance of the database.

## Column and Table Descriptions

While this is not a mandatory requirement, it is good to give description for each fields in the table's column properties so that developers will be able to understand the purpose of the fields. (eg:- In the “Form” table there are two fields “name” and “form\_name” respectively. The purpose of these fields seems ambiguous therefore it would be better to provide a description for these fields so that their meaning is not ambiguous.) The description field can be used for giving a brief explanation of the fields and the purpose of the same in the application

## Index for commonly referenced fields

While indexes have been provided for foreign keys and primary keys, it is not very clear if the database has indexes associated to commonly referenced or searched columns within a given table. Give index for fields which are frequently referenced in the select-queries to improve performance. (eg:- name field in the form table could be searched on hence it needs to be indexed.) The query performance can be improved by giving suitable indexing (clustered and non-clustered index) for tables. Avoid giving index for tables which does not store large number of records. These are suggestions to be used depending on the context.

## Stored Procedures

The use of stored procedures is a hotly debated topic. However stored procedures have their use when you have to process large chunks of data in an iterative manner. It is also a good deterrent against SQL injection attacks. Again context is key. Use it, if it makes sense.

## Initialize Default values appropriately

Some date time fields shows the value “*1900-01-01 00:00:000*”. Use only appropriate default data in the fields. If we store unwanted data in the field it will return unexpected results.

# Smoke Test Results

Smoke testing was done for the admin website. The following modules were covered

1. Administrator main page menu (1)
2. Administrator dashboard (3)
3. Administrator User (1)
4. Add/Edit Country Form (7)
5. Add/Edit Company Type (1)
6. Add/Edit Company (3)
7. Add/Edit Project (5)
8. User Management (1)
9. Access Management (1)
10. Main Page (1)

| **Module** | **Defect Description** |
| --- | --- |
| Administrator main page menu | Profile link not working. Must not have nonfunctioning/broken links like that. |
| Administrator dashboard | Search field is not functioning as expected. |
| Administrator dashboard | Dashboard looks blank. If there is no data to display it will be better to show a message like "No data to display". |
| Administrator dashboard | Dashboard console shows 4 errors. |
| Administrator user | Security measures like multiple login prevention for same user and session logout is not implemented. |
| Add/Edit country Form | Able to duplicate same country name with country code. Must prevent duplications. |
| Add/Edit country Form | No Save/Edit confirmation messages. |
| Add/Edit country Form | Basic Field level validations not set. Even a single space is accepted as a valid input in country name and code fields. |
| Add/Edit country Form | Country name field must not accept special characters and numbers. Also the field must have a limit. |
| Add/Edit country Form | Error message is appeared while pasting codes/scripts to the country field and save. Must not allow user to paste codes/scripts. |
| Add/Edit country Form | Country form console shows 4 errors. |
| Add/Edit country Form | Country code field must not accept characters, special characters and blank spaces. |
| Add/Edit Company type | Field level validations like blank spaces, field limits are not set |
| Add/Edit Company | Company form console shows 5 errors. |
| Add/Edit Company | No Save/Edit confirmation messages. |
| Add/Edit Company | Field level validations like blank spaces, field limits are not set |
| Add/Edit Project | It is better to keep the currency field just below the planned budget field. |
| Add/Edit Project | None of the fields in application is marked with mandatory fields. It will be better to the user if mandatory fields are marked with a star mark. |
| Add/Edit Project | Boundary value analysis Failed for all fields in the project page |
| Add/Edit Project | Unable to save data if maximum limits in each field is entered. Error messages are prevented. |
| Add/Edit Project | No Save/Edit confirmation messages. |
| User Management | Role and Users can be multiple with same name. Doesn’t make sense. |
| Access Management | User Access> Design need improvement |
| Main Page | Customize option in the page is not working as expected and shows plenty of console issues. |