



## Final Project (4FD3)



### Design Documentation

**Submitted By:**

**Sonam Sonam**

**Mark Villa Vidad**

**Submitted On:** 3 April 2018

## Contents

INTRODUCTION .....	3
Project Background .....	3
Functional Requirements .....	4
Non-Functional Requirements .....	5
DESIGN DOCUMENTATION .....	6
Software Design Methodology .....	6
Project Plan.....	7
Use-Case Diagram.....	7
Entity-Relationship Diagram (ERD).....	10
MVC Architecture .....	12
Sequence Diagrams .....	13
State Diagrams .....	14
User Interface Design .....	15
SOFTWARE TECHNOLOGIES IMPLEMENTED .....	16
System File Structure .....	16
Front-End Design.....	18
ReactJS .....	19
Cookies .....	22
Bootstrap .....	22
LESS.....	22
Backend Design.....	23
PHP .....	23
MySQL.....	25
JSON API .....	26
HOSTINGER: Webhost .....	27
CONCLUSION AND RECOMMENDATIONS .....	28
REFERENCES .....	29
APPENDIX .....	30
Gantt Chart.....	30
SQL/Database Schema .....	31
User Interfaces .....	32

## LIST OF FIGURES

Figure 1. Waterfall And Iterative approach .....	6
Figure 2. Initial stage Gantt Chart.....	7
Figure 3. Use-Case Diagram .....	9
Figure 4. Entity-Relationship Diagram.....	11
Figure 5. MVC Architecture .....	12
Figure 6. Sequence Diagram for adding a terminl .....	14
Figure 7. State Diagram for Logging-in .....	15
Figure 8. UI for an admin account .....	16
Figure 9. File system structure (top) .....	17
Figure 10. Content of database.php .....	17
Figure 11. Actual file system structure.....	18
Figure 12. View (Front-End) file structure .....	19
Figure 13. Content of index.html .....	20
Figure 14. Controller view file structure .....	24
Figure 15. Model view file structure .....	24
Figure 16. DEVICE class attributes and methods .....	25
Figure 17. Device object <i>delete()</i> function .....	25
Figure 18. MySQL database .....	26
Figure 19. JSON output.....	27

## INTRODUCTION

### Project Background

McMaster Hospitality Service is one of the fastest growing organization in McMaster University. The department directly provides variety of trending and healthy food options around the university. Currently, they are supporting around 20 stores inside the campus and additional of more than 10 local off-campus vendors.

Mac Express department supports the Hospitality Services business operations by overlooking the system that enables students, staff, and faculty of the university to pay using their university ID card. The current system implemented is called 1CARD from TouchNet-Heartland Systems. Heartland systems also supplies the POS desktop machines, and other payment devices. Other POS equipment such as scanners, printers, printer papers and others are through third party suppliers.

Inevitably as the business grows, more and more stores are being setup each year. Additionally, more off-campus partners are being introduced. Monitoring of POS inventory materials are starting to get problematic. Monitoring available spare parts, checking warranty periods, making updated inventory reports, and other related tasks are becoming more tedious. Currently, Mac Express have a rugged inventory wherein the data is implemented in a non-user-friendly MS Access file. A more user-friendly web-based application would be a huge help in making more the inventory process easier.

## Functional Requirements

Data gathering was done to come with the functional requirements for the system. Users of the system, information to be saved, as well as possible resources to be used were identified.

Use cases were created as shown in Figure 1. Minimum requirements for the system includes the following.

1. Enable users to add, edit, delete (CRUD) information regarding the following:
  - Organizations,
  - Locations,
  - Terminals,
  - Suppliers,
  - Devices,
  - Suppliers,
  - Inventory,
  - Maintenance History, and
  - Users.
2. Let users to be able to log-in and log-out of the system.
3. It allows users to change passwords for security.
4. It allows system administrators to restrict users on some system settings.
5. It allows users to search for specific inventory.
6. It allows users to print reports.
7. It holds records for all transactions being done in the system.
8. It sends system error reports to system admins.
9. It sends email notifications to users about stock status.

## Non-Functional Requirements

- **Portability**

- The source code of the system should be flexible that there should be minimal effort in case of changes in the back-end or front end.

- **Usability**

- The system needs to be as user-friendly as possible as users include non-technical staff. There should be minimal steps in querying user tasks.
- The application must run in major web browsers such as Google Chrome, Internet Explorer, Mozilla and others.

- **Maintenance and Security**

- Accessing critical components of the system should be limited to user type accounts.
- User passwords should be encrypted.
- The system should have daily and weekly back-up.
- Server must be continuously updated with latest software patches.

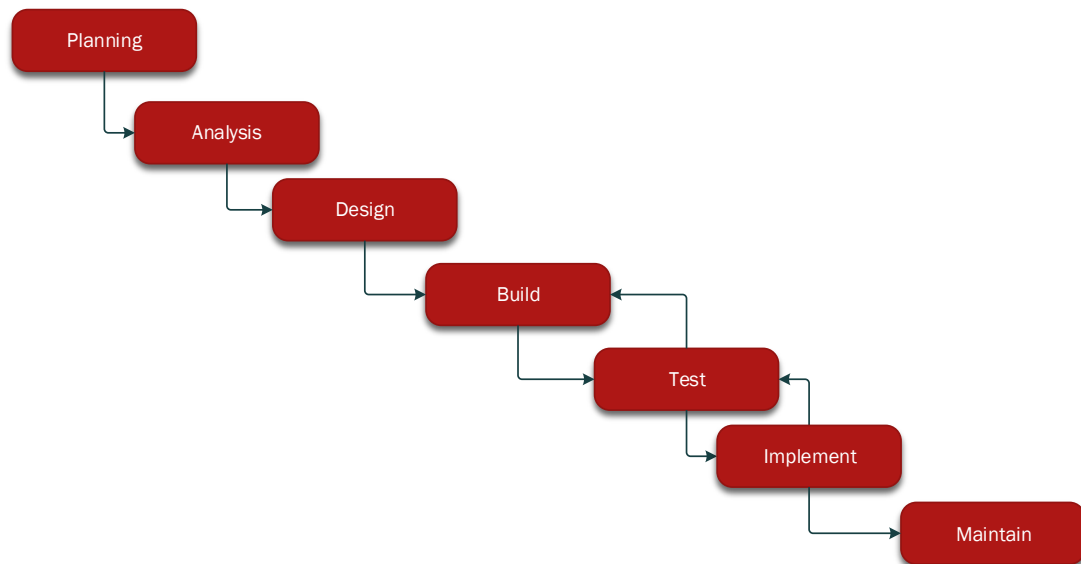
- **Performance**

- Response time of the system should be minimum as much as possible. Users of the system is expected to be less than 10 in a minute, but this might change in the future.

## DESIGN DOCUMENTATION

### Software Design Methodology

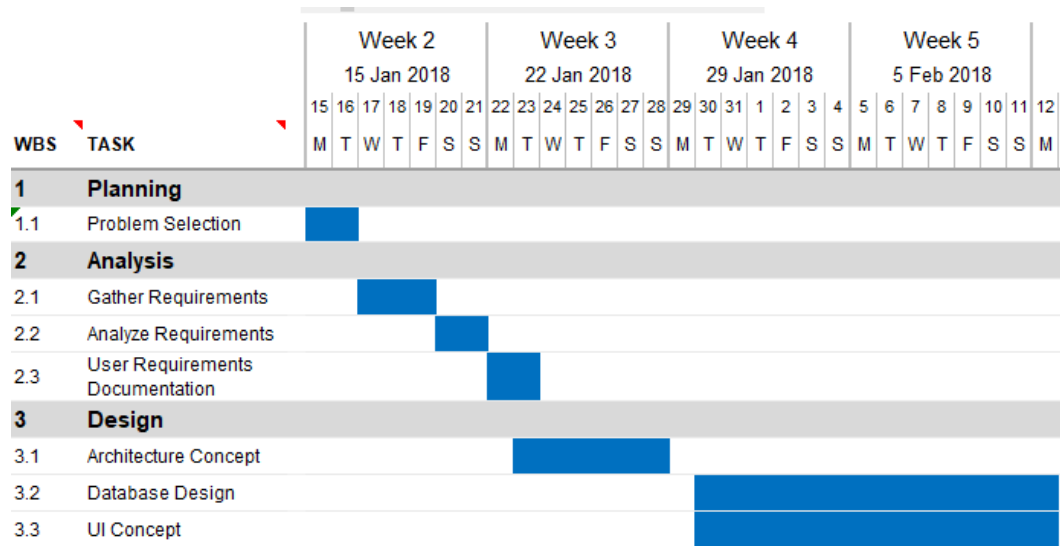
There is a clear idea on the requirements of the system and that makes us possible to use combination of a Waterfall and iterative process. Iterative approach is applicable during the building-testing-implementation phase. This method makes us possible to build the minimum requirements first and then build the other requirements after. Iterative approach during the testing and implementation ensures user functionalities are working.



**Figure 1. Waterfall and Iterative approach**

## Project Plan

Figure 2 is the Gantt chart plan for the initial stages of the development. The complete project plan is documented in Index 1. Based on the SDLC methodology used, the major steps were breakdown into smaller tasks.



**Figure 2. Initial stage Gantt Chart**

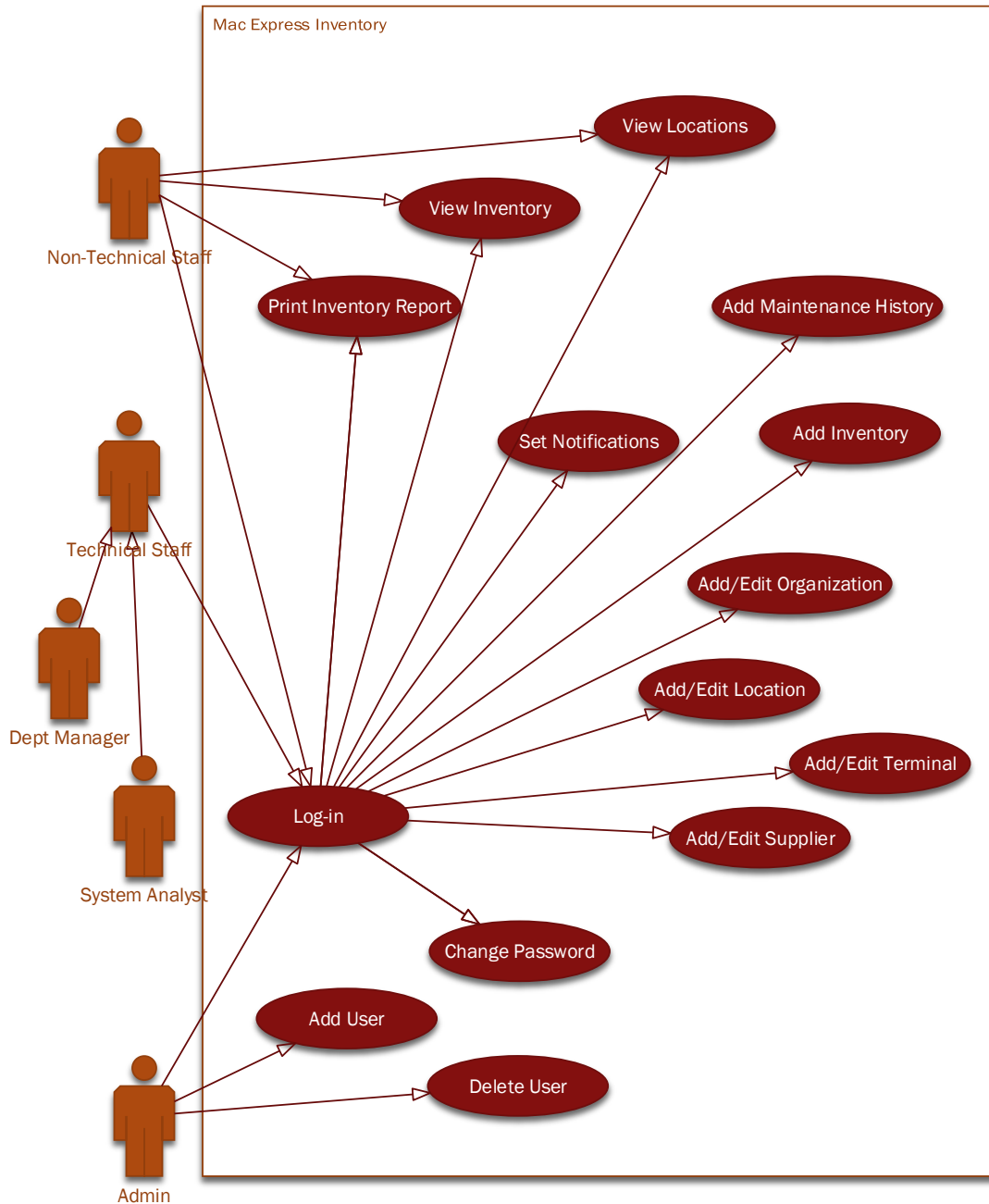
## Use-Case Diagram

Based on initial study, users of the system would include the system admin, system analyst, department manager, and some non-technical staff that may include store managers and office staff. Figure 3 shows the use-case diagram of the system based on initial system requirement analysis. Below are the user descriptions as well as a use-case diagram of the system functional requirements for each user or user-group.

- 1. Admin.** Manages primarily the SQL database. He is in charge in adding or deleting new users of the server.



- 2. System Analyst.** Manages the inventory system. This person also manages the current 1Card system of McMaster University and serve as support to any problem that arises in the system.
- 3. Department Manager.** This may include Hospitality Services Director and Mac Express Manager
- 4. Non-Technical Staff.** Includes store managers and other non-technical staff that can only view locations. Non-technical staff can only view locations and inventory as well as print reports.

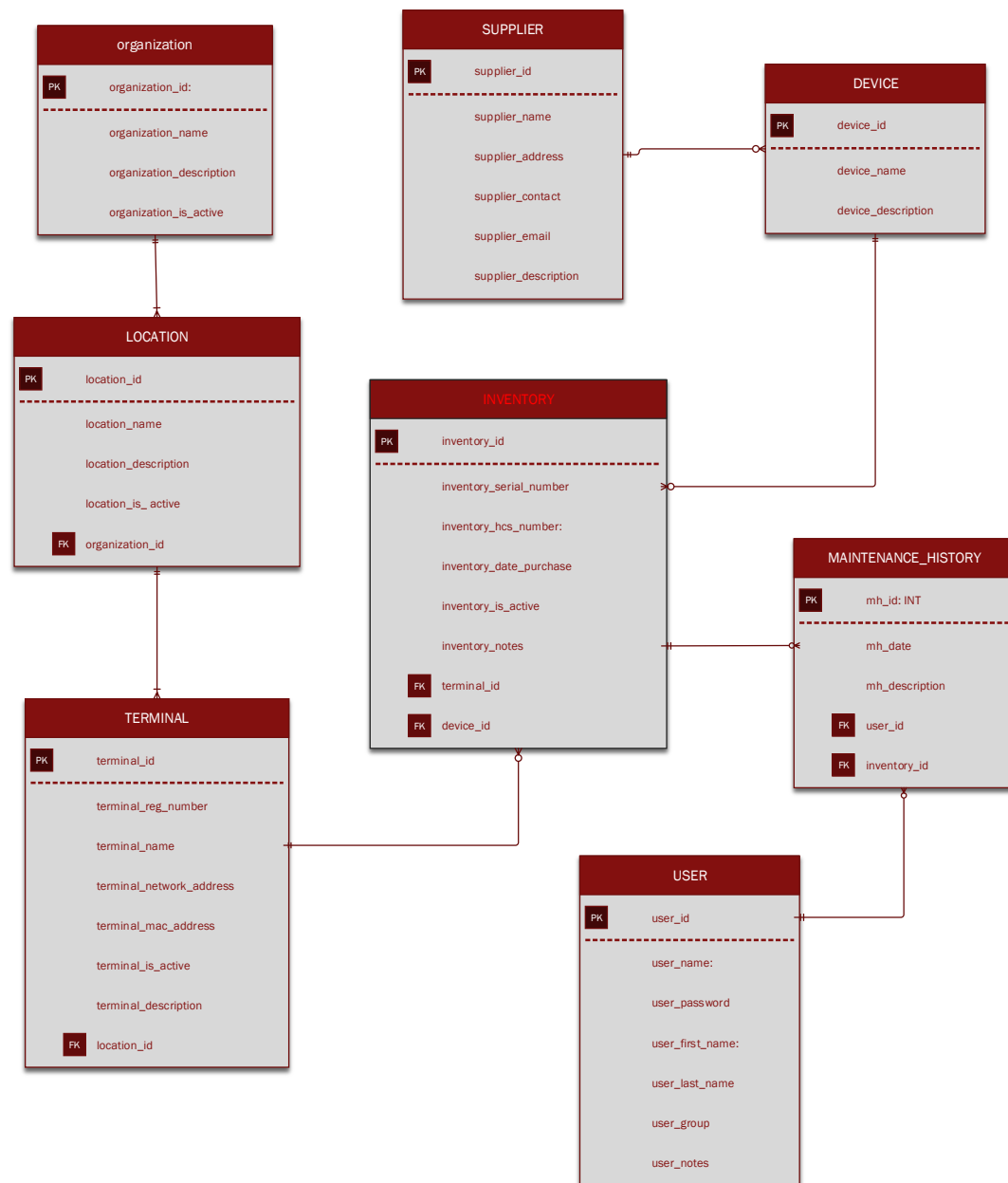


**Figure 3. Use-Case Diagram**

### Entity-Relationship Diagram (ERD)

An entity-relationship model is a way to describe the different processes of the business. It is a great basis in creating objects or database tables in system design. Based on initial requirements, the design will be focused on the inventory, where it is located, who are the suppliers as well as the maintenance history. Figure 4 shows the ERD for the system. This is also the basis of the database schema used in the system. The following are the entities added.

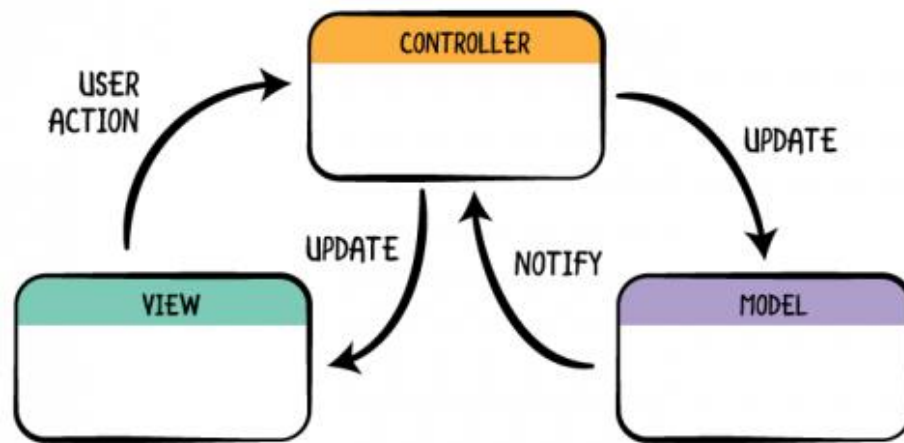
1. **Organization.** Different locations are divided into different organizations. Example is that all off-campus vendors are in the Off-Campus organization.
2. **Location.** This is the actual store. Examples are MUSC-Starbucks, Fireball Café, etc.
3. **Terminal.** This table will list all the terminals- the actual POS machine for a location. One location can have one or more terminals.
4. **Inventory.** A list of all the inventories for all locations identified by their ID.
5. **Supplier.** Every inventory has a supplier. The table will include supplier details.
6. **Device Type.** This will be the device type of an inventory. Example is a printer.
7. **Maintenance History.** Maintenance history will be added for each inventory as this would be helpful for a user to track device history.



**Figure 4. Entity-Relationship Diagram**

## MVC Architecture

Over the years, Model-View-Controller become so popular for web development. As shown in Figure 5, this pattern divides the system into three main components – Model, View, and Controller. The Model part directly handles the database. In an SQL back-end, SQL commands are in this section. It directly manages the data and logic of the operation. The View part is involved in presenting the date to the user. The Controller servers as the gateway between the two.



**Figure 5. MVC Architecture**

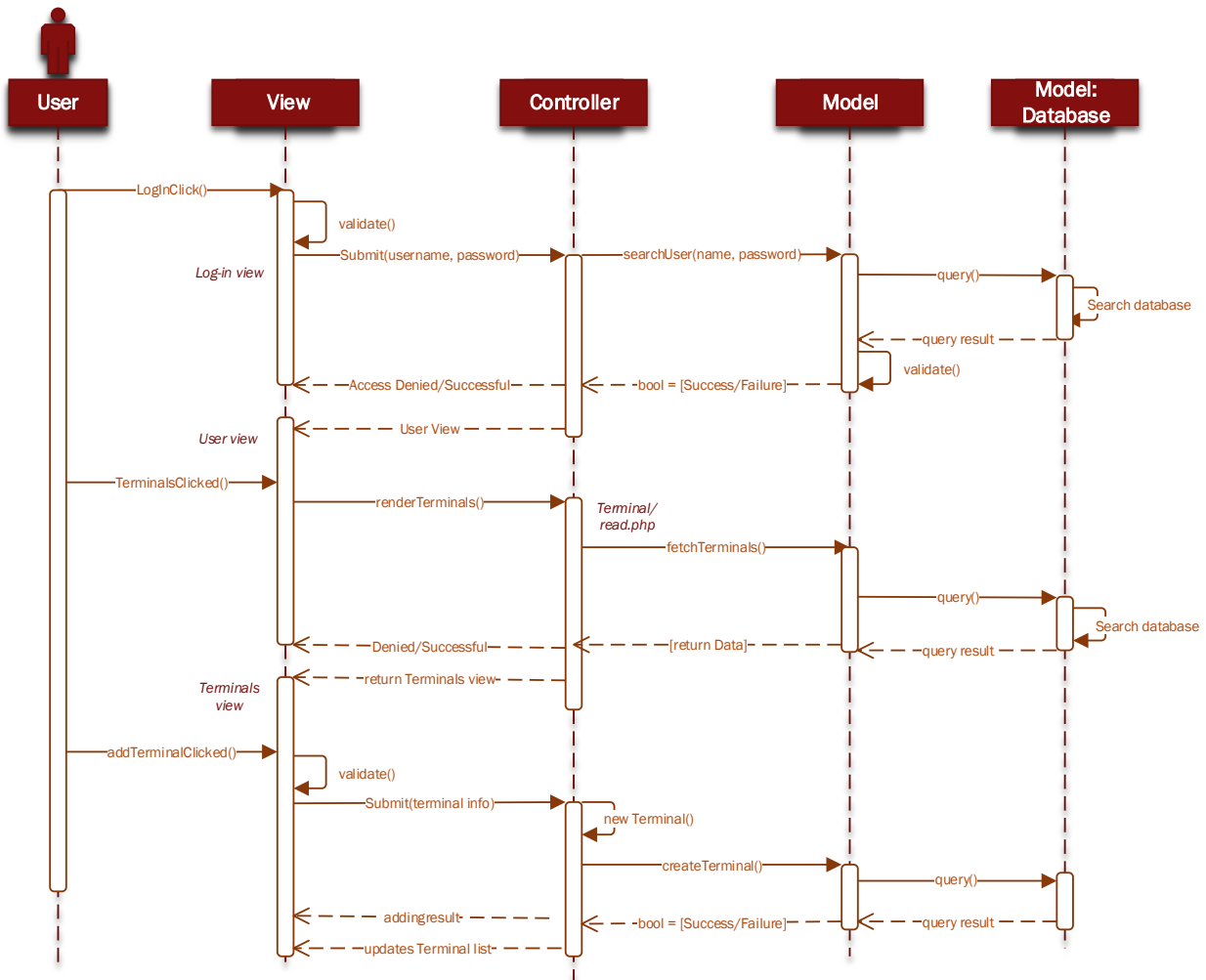
We decided to implement the system using MVC architecture knowing its following advantages.

- **Faster Development Process.** There are two developers in the group and having this architecture would enable us to work in parallel. The first developer can work in the front-end while the other one is working with the back-end.

- **Low coupling.** Having independent back-end from front-end would make the system more portable and flexible. Modification in one part does not affect the entire model.
- **High Cohesion.** One of the good programming practices is having your codes “high cohesion and low coupling.” MVC framework attains this by logically grouping related actions.
- **Multiple Views Possible.** MVC implementation is a great way to create multiple views for a model. This is important for future expansion in order to be able to use the system in different devices.

### Sequence Diagrams

Implementing MVC design makes a process or request to go first in the controller before executing the actual server commands. Figure 6 shows how system implements the log-in system. When the user clicks a log-in button, a window will show wherein he will type username and password. When submit is clicked, View will transfer the request to Controller with verification request indicating username and password. Controller will then request this through the Model which has the actual server codes. If username and password are good, then model will return success to controller. Controller will then send this information to View. With this process, we can say that MVC can be complex as it adds layers of abstraction.



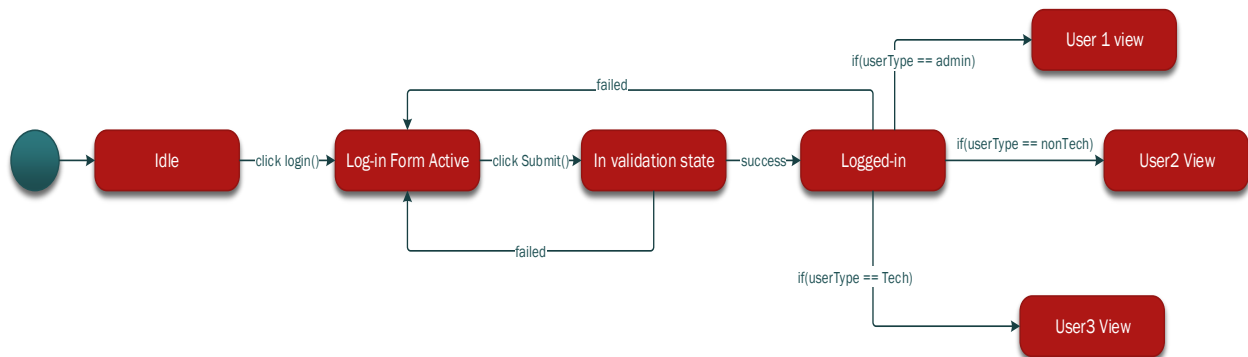
**Figure 6. Sequence Diagram for adding a terminal**

## State Diagrams

The system is an event-driven and that makes its behavior dependent on the user input.

When a user goes to the website, the default homepage will be shown. The system will be in idle mode unless a user clicked a button or a link. In Figure 7, it shows the typical state of the system when user logged-in. When user decided to log-in, system will show log-in form where user can input his username and password. When user submit, the system will go in

Validation state wherein query to the database will be done to make sure that the user exists as well as if the password is current. If it is successful, then the system will go in User Logged-in state. User view will be dependent on the type of user logged in.

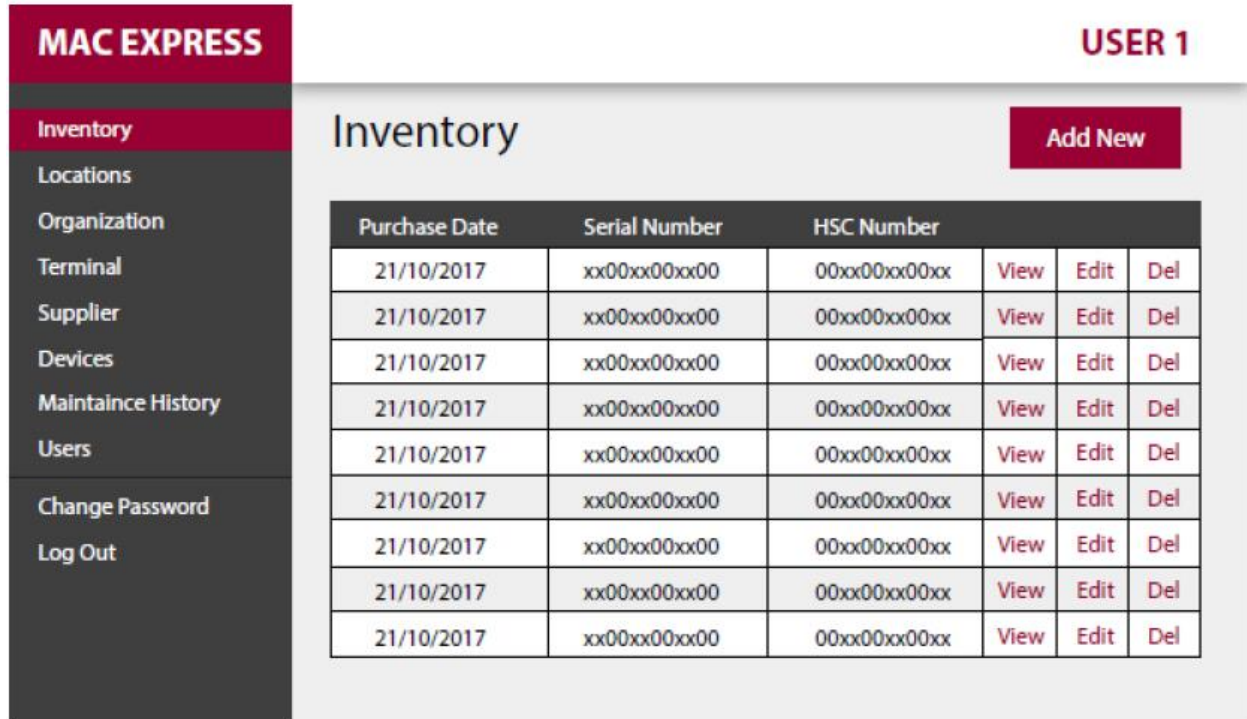


**Figure 7. State Diagram for Logging-in**

### User Interface Design

We wanted the system to be inventory-centric so when user logged-in their account, the first window to show would be then inventory list. However, page navigation for different tables would be present in the side or top of the window. Figure 8 shows the planned user interface for an admin account. Inventory view would be the default view but other functionalities for the account is available in the side bar. For the Inventory form, list of then inventories are listed including the option to View for more information about the item, Edit, Delete, and Add new item.



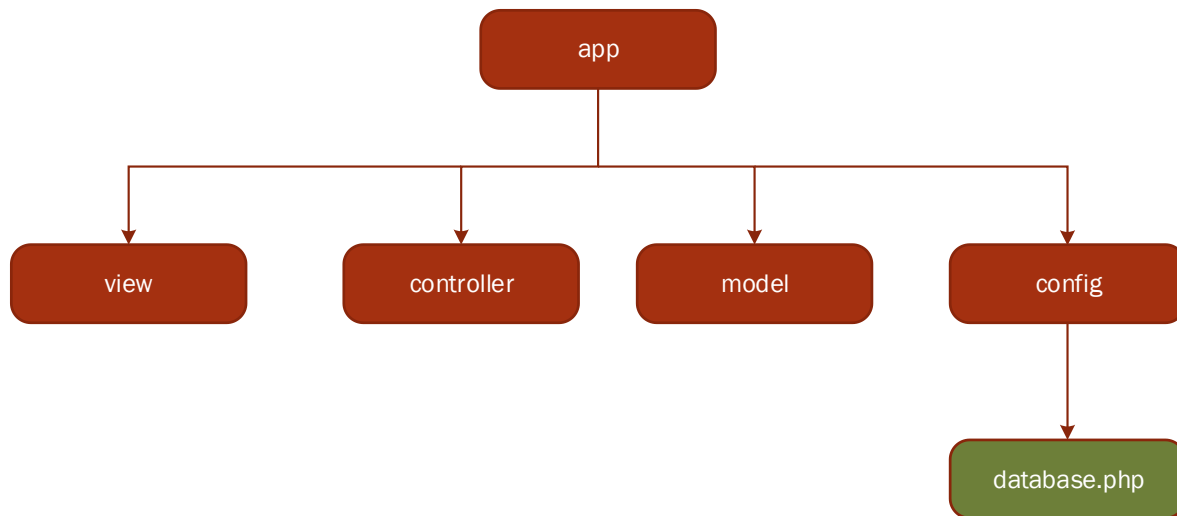


**Figure 8. UI for an admin account**

## SOFTWARE TECHNOLOGIES IMPLEMENTED

### System File Structure

The fact that the system is implemented in MVC architecture, main folders will be the View, Controller and Model. Figure 9 shows the top file structure of the system while Figure 11 shows the actual file structure for the project. Note that this is MVC architecture and that they are loosely coupled. The “config” folder will be the database setup of the server. Saved in this folder would be the *database.php*. As shown in Figure 10, *database.php* indicates the database credentials of the server to be used. Having these separate files from the model folder makes it easier for anyone to update it.

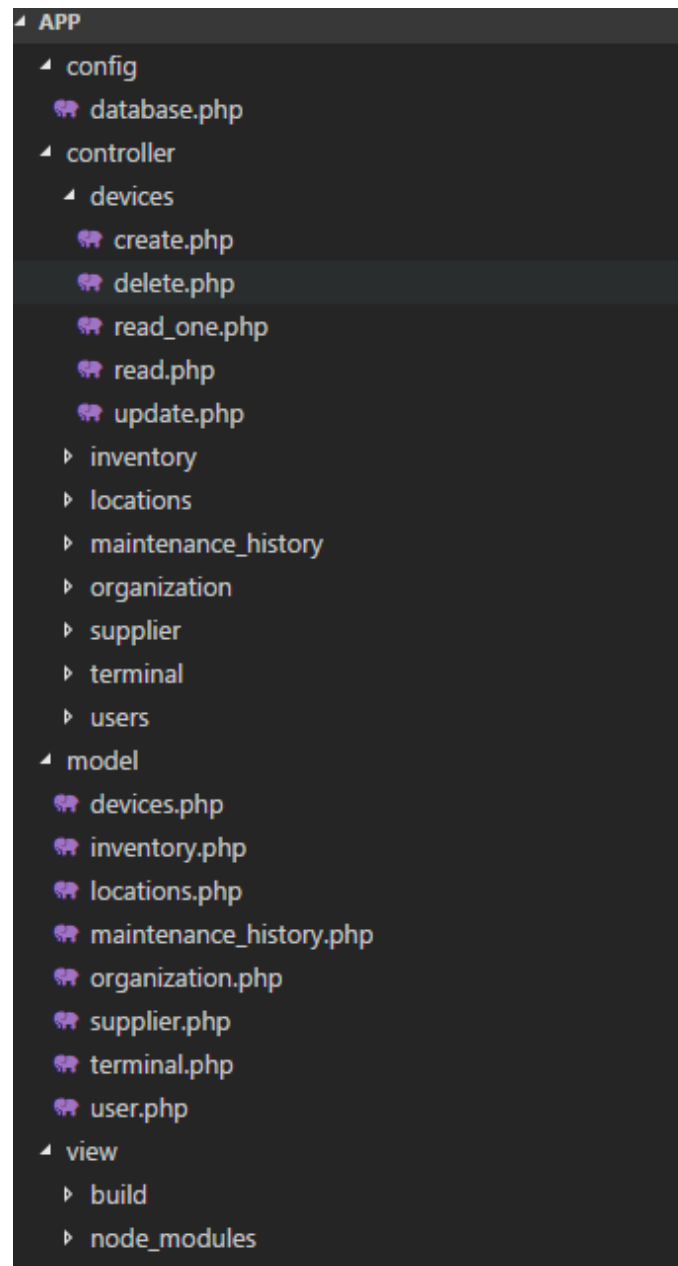


**Figure 9. File system structure (top)**

```

1  <?php
2  session_start();
3  class Database{
4
5      // specify your own database credentials
6      // private $host = "localhost";
7      // private $db_name = "mac_express_inv";
8      // private $username = "root";
9      // private $password = "mysql";
10     private $host= "mysql.hostinger.com";
11     private $db_name = "u707314810_mac";
12     private $username = "u707314810_root";
13     private $password = "mysqlmac";
14     public $conn;
15
16     // get the database connection
17     public function getConnection(){
18
19         $this->conn = null;
20
21         try{
22             $this->conn = new PDO("mysql:host=" . $this->host . ";dbname=" . $this->db_name, $this->username, $this->password);
23             $this->conn->exec("set names utf8");
24         }catch(PDOException $exception){
25             echo "Connection error: " . $exception->getMessage();
26         }
27         return $this->conn;
28     }
29 }
30
  
```

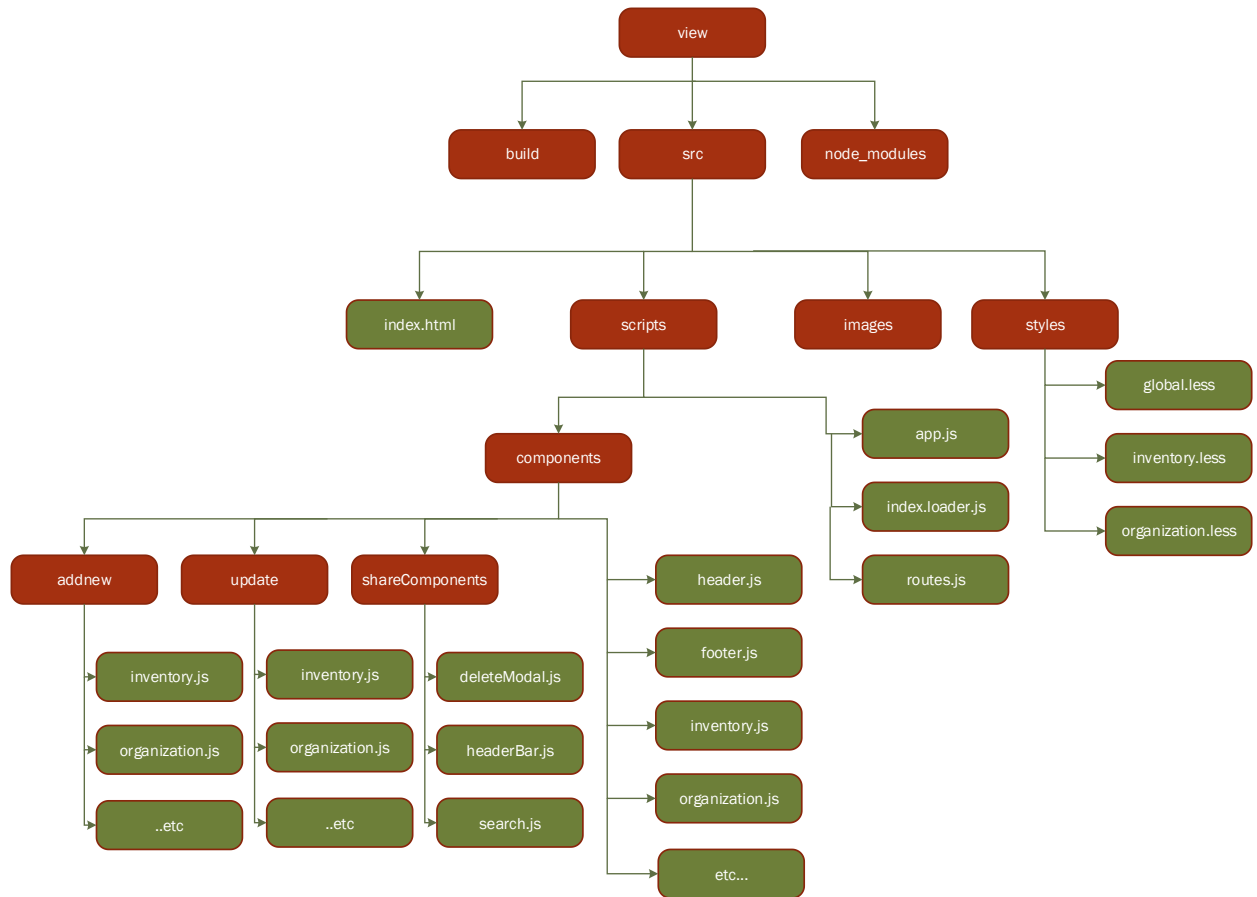
**Figure 10. Content of database.php**



**Figure 11. Actual file system structure**

### Front-End Design

The system is implemented using ReactJS, Bootstrap, HTML and Less for front-end. Also, Webpack is used to compile ReactJS and LESS for the browser. Figure 12 shows the file structure of the View model.



**Figure 12. View (Front-End) file structure**

### ReactJS

We used ReactJS to build the user interfaces. ReactJS is a JavaScript library which is known to provide speed, simplicity, and scalability. Facebook, Instagram and a community of developers and corporations are actively maintaining the framework. React is based on View model. When the user clicked a link or button, it just changes the view of the page without refreshing the page. ReactJS is popular for single-page applications as it handles required DOM updates efficiently. It is possible to create different components and can switch the changes based on the URL.

```

<!DOCTYPE html>
<html>
<head>
  <link rel="icon" href="" type="image/x-icon">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>MAC Express</title>
</head>
<body>
  <div id="app"></div>
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
</body>
</html>

```

**Figure 13. Content of index.html**

Figure 13 shows the content of the file `index.html`. This is the only content that is showing to a user wherever part of the system he is browsing. The `div` with `id="app"` will serve as a container for rendering the different pages. Added in the bottom is the jQuery library that supports the Bootstrap.

In React, it is possible to create different standalone components and use wherever it is needed. Instead of HTML files, different JavaScript files are created. HTML is part of the JS files created. This is shown in Figure 12 where there is only one HTML file – `index.html` and the different tables are rendered through JS files under the ‘components’ folder. The following are some of the JS files created for the system.

- **index.js.** It adds router content to “app” component.
- **app.js.** It holds all the components like home, login, Forget Password, Inventory and more. Template for different pages’ path and their component were created. Inside the `render` function are the Header, Switch, and Footer components. Header and

Footer will be constant for all the pages and the Switch will be dependent on the requested page content. The program will loop through the template using JavaScript *map* function to add the key, path and component. Added also is the authorization access of the user – Admin, Tech, or NonTech.

- **routes.js.** It adds the app components in the router. The BrowserRouter from ReactJS is also imported to support switching between the different components.
- **header.js.** Basic HTML to create the top bar and user information if user is logged-in. It contains basic HTML header for all components.
- **footer.js.** It contains basic HTML to be added in the bottom bar of all pages.
- **login.js.** Contains Login page built through Bootstrap HTML form. On form submit, it makes a POST ajax call to *login.php*. On success, REACT redirect it to inventory page. If it fails, then an error message will show.
- **forgotpassword.js.** It contains the form that allows a user to enter an email and request for an temporary password. Front end makes a request for *forgot-password.php* and the PHP will generate a random password and send an email to a valid user email with a password. If the email is invalid it will show an error on the page.
- **inventory.js.** React's *componentDidMount* function is used on load to make a GET Ajax request to backend *inventory/read.php* to get all the inventory data and populate the front-end's table. The page also contains the search input form. Search works by making a call to *read.php*. The "s" parameter is passed to the backend and filter results based on this parameter and sends it back to the front-end. In front-end, *this.state.inventories* stores the data which updates depending on the content of *this.setState*.

In React, it is possible to create micro components and pass the properties based on the page. Example of this are the HeaderBar, Search and Delete Modal located in shareComponent folder (Figure 12).

### Cookies

Cookies-JS were used in the project set username and the user-role. Based on the User-role cookies, different user view was made for Admin, Tech and NoTech. Current prototype has unsecured cookies though. To make Cookies secure, it need to pass secure true in *cookies.set* and that works for HTTPS. Currently, the domain the system implemented doesn't have SSL certificate, it is only HTTP. Cookies were set when user is logged in and remove when user is logged out. It is set to expire to 1 which means the cookies will be valid for 1 day.

### Bootstrap

Bootstrap is a front-end framework developed at Twitter to encourage consistency across internal tools. Adding the Bootstrap classes helps to design the site faster as there are already default design for different HTML components. ReactJS (<https://getbootstrap.com/docs/4>) provides the support to different functionality like create the build package, compile less and more.

### LESS

LESS is just a processor of CSS. Using LESS makes styling more flexible as it allows to variables to be defined. In the system, LESS files are under the style folder (as shown in Figure 12) which includes some of the following:

- **variable.less.** It holds all the variables like font colors, font size.
- **main.less.** It includes all the less file and webpack will compile this file for the final product.

- **global.less.** It has the style that will be used globally in different pages or different components.

The rest of the LESS files holds the style for the different pages. In the end, all these different less files will be compiled by webpack as one big CSS file.

## Backend Design

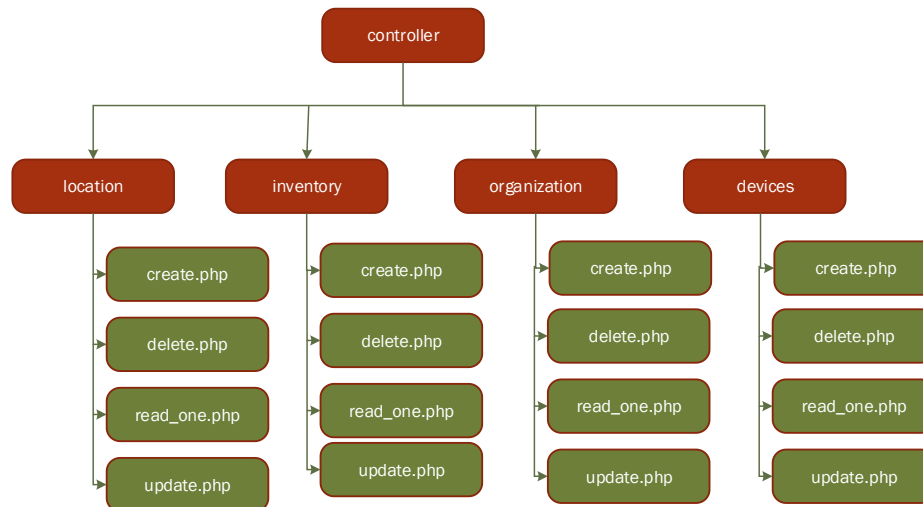
To simulate the server-side, we use PHP, Apache for the webserver, and MySQL as the database. Figure 14 shows the Controller view structure which contains the different php files.

### PHP

Most Content Management Systems (CMS) such as WordPress, Joomla, and Drupal as well as major websites like Facebook still use PHP to power their back-end. Currently, 83.1% of all websites are still using PHP. This is the reason why we go with PHP in implementing the Controller and Model views.

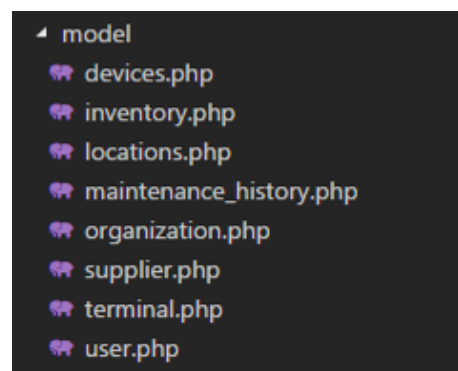
As shown in Figure 14, Controller view is all implemented in PHP. Each of the objects or tables have their own php files for *create*, *delete*, *read\_one*, and *update*. These files handle the GET and POST requests from the front-end or View. Corresponding php file will then call the function from the model to perform the needed query to the database.





**Figure 14. Controller view file structure**

Figure 15 shows the files inside the Model. A class of each of the tables are created containing the different the different attributes and methods of each. Figure 16 shows the UML class diagram of the object Device. Methods contains the actual SQL querying codes for the database. Figure 17 shows how a *delete()* function is implemented for the device object.



**Figure 15. Model view file structure**



Figure 16. DEVICE class attributes and methods

```

// delete the location
function delete(){
    // delete query
    $query = "DELETE FROM " . $this->table_name . " WHERE device_id = ?";

    // prepare query
    $stmt = $this->conn->prepare($query);

    // sanitize
    $this->id=htmlspecialchars(strip_tags($this->id));

    // bind id of record to delete
    $stmt->bindParam(1, $this->id);

    // execute query
    if($stmt->execute()){
        return true;
    }

    return false;
}
  
```

Figure 17. Device object *delete* function

## MySQL

System's database is implemented through MySQL, the most popular Open Source SQL database management system. It is known to be fast, efficient, and functions with standard

SQL syntax. It is used by many large, high-profile web apps including Facebook, Google Twitter, and YouTube. During the development process, database is simulated using AAMPS package (<https://www.ampps.com/>). Figure 18 shows how the different tables in the database *mac\_express\_inv*. Manipulation of the database is done through the model php files. In Figure 17, it can be noticed that delete *function()* query using SQL syntax..

Table	Action
<input type="checkbox"/> device	★ Browse Structure Search Insert Empty Drop
<input type="checkbox"/> inventory	★ Browse Structure Search Insert Empty Drop
<input type="checkbox"/> location	★ Browse Structure Search Insert Empty Drop
<input type="checkbox"/> maintenance_history	★ Browse Structure Search Insert Empty Drop
<input type="checkbox"/> organization	★ Browse Structure Search Insert Empty Drop
<input type="checkbox"/> supplier	★ Browse Structure Search Insert Empty Drop
<input type="checkbox"/> terminal	★ Browse Structure Search Insert Empty Drop
<input type="checkbox"/> user	★ Browse Structure Search Insert Empty Drop
8 tables	Sum

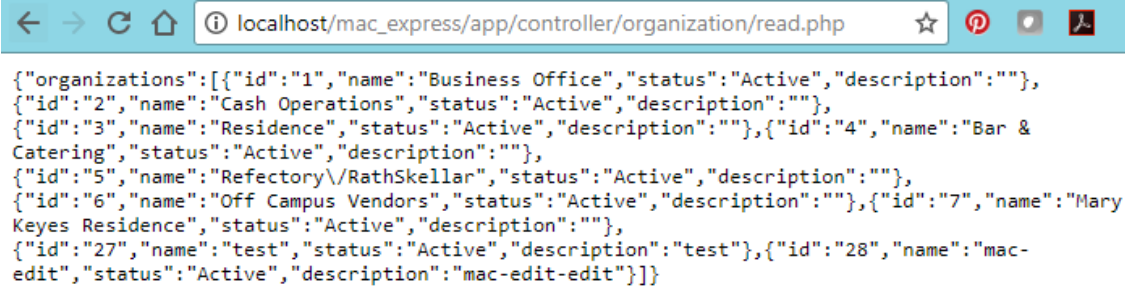
**Figure 18. MySQL database**

## JSON API

When the front-end send GET request to the back-end, controller send data using JSON files. Back-end is independent by itself and that front-end user HTTP requests to communicate. This shows the loose coupling between back-end and front end. In Figure 19,

this is the output of controller when front-end requests the list of organizations saved in the database. Notice that the link is

[http://localhost/mac\\_express/app/controller/organization/read.php](http://localhost/mac_express/app/controller/organization/read.php) and that front-end just convert this JSON file into a more readable format for the user.



The screenshot shows a web browser window with the address bar displaying `localhost/mac_express/app/controller/organization/read.php`. The main content area displays a JSON array of organization objects. Each object contains fields for `id`, `name`, `status`, and `description`. The organizations listed include Business Office, Cash Operations, Residence, Bar & Catering, Refectory/RathSkellar, Off Campus Vendors, Mary Keyes Residence, a test entry, and mac-edit.

```
{
  "organizations": [
    {
      "id": "1",
      "name": "Business Office",
      "status": "Active",
      "description": ""
    },
    {
      "id": "2",
      "name": "Cash Operations",
      "status": "Active",
      "description": ""
    },
    {
      "id": "3",
      "name": "Residence",
      "status": "Active",
      "description": ""
    },
    {
      "id": "4",
      "name": "Bar & Catering",
      "status": "Active",
      "description": ""
    },
    {
      "id": "5",
      "name": "Refectory\\RathSkellar",
      "status": "Active",
      "description": ""
    },
    {
      "id": "6",
      "name": "Off Campus Vendors",
      "status": "Active",
      "description": ""
    },
    {
      "id": "7",
      "name": "Mary Keyes Residence",
      "status": "Active",
      "description": ""
    },
    {
      "id": "27",
      "name": "test",
      "status": "Active",
      "description": "test"
    },
    {
      "id": "28",
      "name": "mac-edit",
      "status": "Active",
      "description": "mac-edit-edit"
    }
  ]
}
```

**Figure 19. JSON output**

#### **HOSTINGER: Webhost**

Prototype is currently hosted using Hostinger, a popular hosting provider. Application can be accessed through web URL at <http://sonamk.com/>. Actual implementation can be done using these available hosting providers or can be hosted internally by the university through University's Technology Services (UTS).

## CONCLUSION AND RECOMMENDATIONS

The user requirements are clearly defined which makes development easier. Challenged encountered were implementing the design in an MVC architecture as well as choosing the right technologies for the project. Overall, designing the system went smoothly and a working prototype is delivered successfully.

However, due to the time constraint there are some functionalities that are still undone, and it is expected that there would be no problem adding it in the existing system. These functionalities include the following:

1. **Password Encryption.** Though front-end shows masked password when the user is typing it, passwords are still saved without encryption in the database.
2. **Cookies Security.** Currently, cookies are not secure as it is only implemented in HTTPS.
3. **Email Notifications.** One of the functional requirements specified is to be able for the system to send notifications in case an item is running low.
4. **Better printable reports.** There is a default print options in every page but still there no custom options for print like printing only inventories in a specific location.
5. **Sorting options.** Showing items are still in default view as ordered in the database.
6. **Better User Interface.** There are still a lot of improvements that can be done in the pages for a better user experience. One would be showing a map with markers of the different locations.

## REFERENCES

Hospitality Services. (n.d.). Retrieved March 20, 2018, from <http://hospitality.mcmaster.ca/index.html>

My Meal Account. (n.d.). Retrieved March 21, 2018, from <http://mealcard.mcmaster.ca/acctinfo.htm>

Model-View-Controller Explained in C. (n.d.). Retrieved March 22, 2018, from <https://helloacm.com/model-view-controller-explained-in-c/>

Model-View-Controller (MVC) in iOS: A Modern Approach. (n.d.). Retrieved March 22, 2018, from <https://www.raywenderlich.com/132662/mvc-in-ios-a-modern-approach>

MySQL 5.7 Reference Manual :: 1.3.1 What is MySQL? (n.d.). Retrieved March 29, 2018, from <https://dev.mysql.com/doc/refman/5.7/en/what-is-mysql.html>

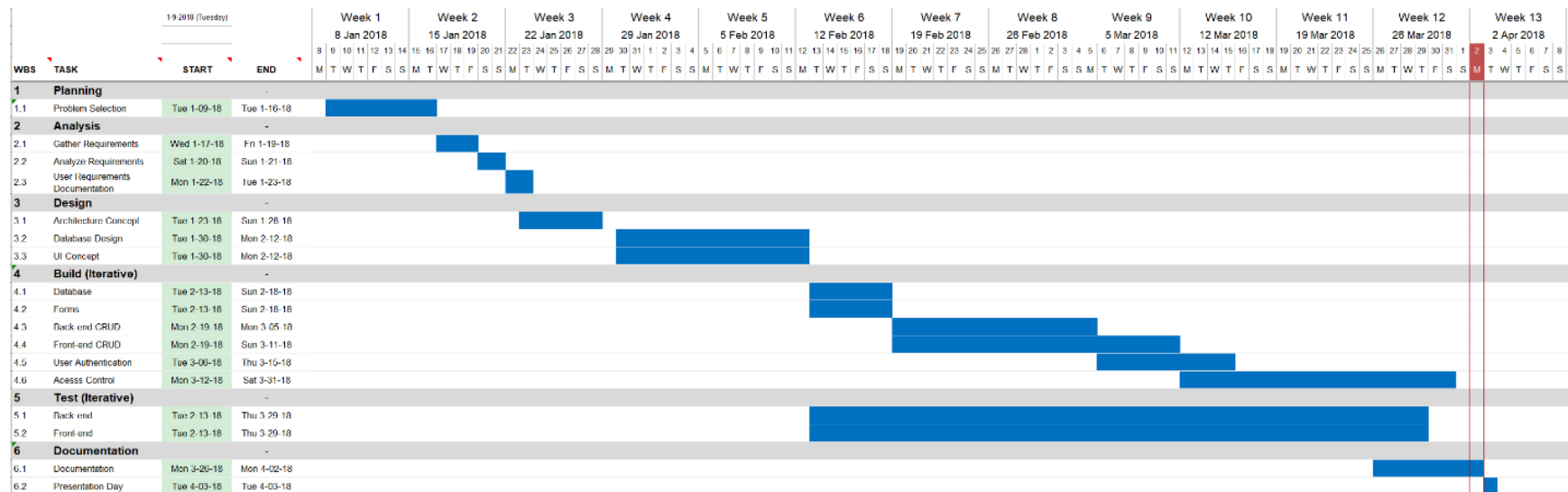
Otto, M., & Thornton, J. (n.d.). Bootstrap. Retrieved March 22, 2018, from <https://getbootstrap.com/>

React. (n.d.). Retrieved March 29, 2018, from <https://reactjs.org/>

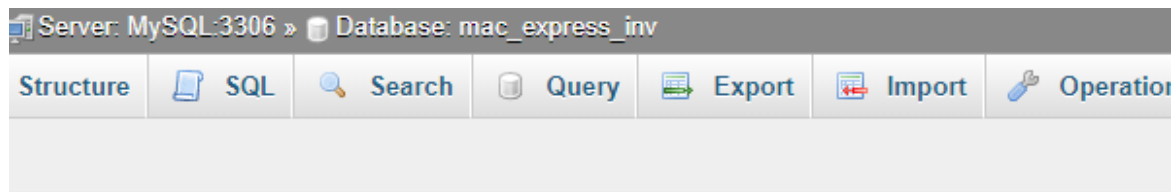
Usage statistics and market share of PHP for websites. (n.d.). Retrieved March 25, 2018, from <https://w3techs.com/technologies/details/pl-php/all/all>

## APPENDIX

### Gantt Chart



## SQL/Database Schema



mac_express_inv organization
organization_id : int(11)
organization_name : varchar(50)
organization_description : varchar(255)
organization_is_active : tinyint(1)

mac_express_inv location
location_id : int(11)
organization_id : int(11)
location_name : varchar(45)
location_is_active : tinyint(1)
location_description : varchar(255)

mac_express_inv terminal
terminal_id : int(11) unsigned
terminal_reg_number : int(11) unsigned
terminal_name : varchar(45)
terminal_network_address : varchar(45)
terminal_network_name : varchar(45)
terminal_mac_address : varchar(45)
terminal_is_active : tinyint(1)
terminal_description : varchar(255)
location_id : int(11) unsigned

mac_express_inv user
user_id : int(11)
user_name : varchar(45)
user_password : varchar(45)
user_first_name : varchar(45)
user_last_name : varchar(45)
user_group : varchar(45)
user_notes : varchar(255)

mac_express_inv device
device_id : int(11)
device_name : varchar(45)
device_description : varchar(255)

mac_express_inv inventory
inventory_id : int(11)
inventory_serial_number : varchar(45)
inventory_hcs_number : varchar(45)
inventory_date_purchase : date
inventory_is_active : tinyint(1)
inventory_notes : varchar(255)
terminal_id : int(11)
device_id : int(11)

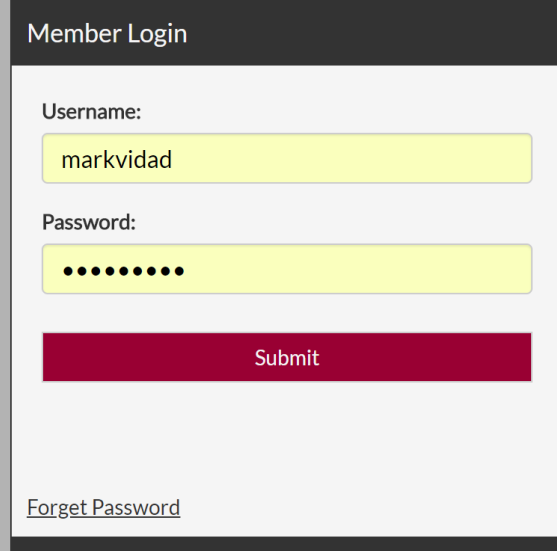
mac_express_inv supplier
supplier_id : int(11)
supplier_name : varchar(45)
supplier_address : varchar(255)
supplier_contact : varchar(45)
supplier_email : varchar(45)
supplier_description : varchar(255)

mac_express_inv maintenance_history
mh_id : int(11)
mh_date : date
mh_description : varchar(255)
user_id : int(11)
inventory_id : int(11)



## User Interfaces

### a. Log-in



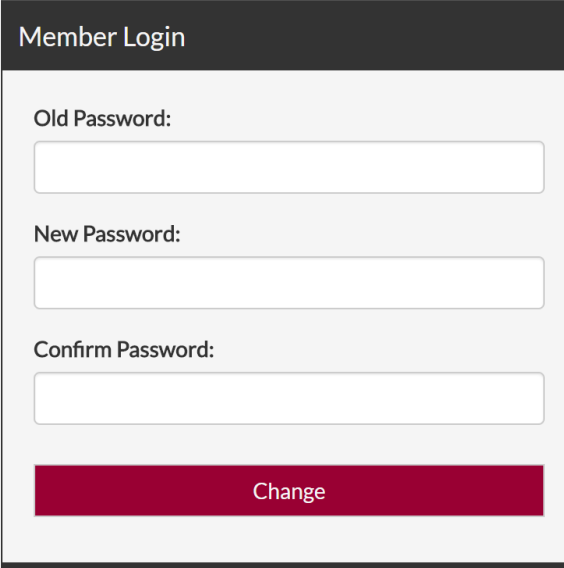
Member Login

Username:

Password:

[Forget Password](#)

### b. Change Password Form

































Member Login

Old Password:






















New Password:

Confirm Password:

### c. Admin View

MAC EXPRESS	Sonam Sonam						
Inventory	Inventories <span>Add New</span>						
Locations	Search <input type="text"/>						
Organizations	SerialNo	Device	HCSNo	Purchase Date	Status	Terminal	
Terminals	WILL-HS-ADMIN	CPU Admin		2009-09-01	Active	WILLIAMS-HSCADMIN	  
Suppliers	W1450421081817-0013	Cash Drawer		2017-09-06	Active	Spare Parts	  
Devices	W1446251081817-0001	Cash Drawer		2017-09-06	Active	Spare Parts	  
Maintenance History	W1437231071417-0176	Cash Drawer		2017-09-06	Active	Spare Parts	  
Users	W1437231071417-0138	Cash Drawer		2017-09-06	Active	Spare Parts	  
	W1437231071417-0137	Cash Drawer		2017-09-06	Active	Spare Parts	  
Change Password	T320-48-2	Cash Drawer		2009-09-01	Active	LAPIAZZA-4	  
Log Out	T320-48-1	Cash Drawer		2009-09-01	Active	LAPIAZZA-4	  
	T320-22-2	Cash Drawer		2000-01-01	Active	LAPIAZZA-2	  
	T320-22-1	Cash Drawer		2000-01-01	Active	LAPIAZZA-2	  

### d. Technical Staff View

MAC EXPRESS	Test Test						
Inventory	Inventories <span>Add New</span>						
Locations	Search <input type="text"/>						
Organizations	SerialNo	Device	HCSNo	Purchase Date	Status	Terminal	
Terminals	WILL-HS-ADMIN	CPU Admin		2009-09-01	Active	WILLIAMS-HSC ADMIN	  
Suppliers	W1450421081817-0013	Cash Drawer		2017-09-06	Active	Spare Parts	  
Devices	W1446251081817-0001	Cash Drawer		2017-09-06	Active	Spare Parts	  
Maintenance History	W1437231071417-0176	Cash Drawer		2017-09-06	Active	Spare Parts	  
Change Password	W1437231071417-0138	Cash Drawer		2017-09-06	Active	Spare Parts	  
Log Out	W1437231071417-0137	Cash Drawer		2017-09-06	Active	Spare Parts	  
	T320-48-2	Cash Drawer		2009-09-01	Active	LAPIAZZA-4	  

### e. Non-Technical Staff View

MAC EXPRESS

Inventory

Locations

Change Password

Log Out

Test2 Test2

Inventories

Search

SerialNo	Device	HCSNo	Purchase Date	Status	Terminal	
WILL-HS-ADMIN	CPU Admin		2009-09-01	Active	WILLIAMS-HSC ADMIN	
W1450421081817-0013	Cash Drawer		2017-09-06	Active	Spare Parts	
W1446251081817-0001	Cash Drawer		2017-09-06	Active	Spare Parts	
W1437231071417-0176	Cash Drawer		2017-09-06	Active	Spare Parts	
W1437231071417-0138	Cash Drawer		2017-09-06	Active	Spare Parts	
W1437231071417-0137	Cash Drawer		2017-09-06	Active	Spare Parts	
T320-48-2	Cash Drawer		2009-09-01	Active	LAPIAZZA-4	
T320-48-1	Cash Drawer		2009-09-01	Active	LAPIAZZA-4	

### f. Inventory Item View Form

MAC EXPRESS

Inventory

Locations

Organizations

Terminals

Suppliers

Devices

Maintenance History

Users

Change Password

Log Out

Mark Vidad

Inventory Info

Back

MAC Express Inventory

SN: WILL-HS-ADMIN

Date: 2009-09-01

HCS Number:

Terminal: WILLIAMS-HSC ADMIN

Device: CPU Admin

Supplier: Master Distributors Inc.

Location: William HSC

Organization: Cash Operations

Print