

M2M Connect; SMS

→ *PHPProcessing*

Adam Hai - P14174331, Benoit Njika - P12210713,
Danish Hussain - P14181072

January 2017

Supervisor 1: Clinton Ingrams

Supervisor 2: Rafael Ktistakis

Contents

1	Introduction	3
2	Project Contract	4
3	System Specifications	5
4	System Architecture	7
4.0.1	Model	7
4.0.2	View	8
4.0.3	Controller	10
4.1	Use Case Scenarios	11
4.1.1	Case Scenario: Board Status Display	11
4.1.2	Case Scenario: New Status Updates	11
4.2	Project Structure	12
4.3	UML Diagrams	13
5	System Implementation	16
5.1	Model-View-Controller Architecture	16
5.2	Security	19
6	Test Plan	21
7	Version Control	23
7.1	SVN Commit History	23
8	Extensions	25
9	Conclusion	28

1 Introduction

The following project requires a web application that will control and manage the status of telematics circuit boards. Furthermore, the application will also contain functionality that will display and extract updates of the boards which can hold information such as, the temperature or fan statuses.

Information about the boards will be accessible via a public webpage that will allow users to view the board statues. The webpage should be easy to use, so that the users can easily find the data or information required. The system architecture for the proposed project will be to use Model-view-controller(MVC), this is so that a large quantity of data can be displayed without puzzling the user.



Figure 1.1: Telematics circuit board

2 Project Contract

Table 2.1: Table with member assigned roles

Team Member	Roles
A	Web Application Developer/Architect
B	Documenter/Author/Designer
C	Tester

Member A Adam Hai **Signature** _____ **Date** _____

Member B Danish Hussain **Signature** _____ **Date** _____

Member C Benoit Njika **Signature** _____ **Date** _____

3 System Specifications

Prior to starting the development of the application, it is important to specify the requirements for this project. This helps to certify the objectives of the application.

The short message service, also known as SMS is mostly used within mobile phones. This allows users to transfer and exchange plain-text messages to one another(Triggs,2013). However, this feature is also implemented into the circuit boards in the telematics laboratories, due to the fact that there is a limit of 160 characters on messages that can be transmitted (Plivo, 2016) it is vital to ensure that the whole status of the board can be transmitted within a single SMS.

The updates retrieved from the board will then be sent to the EE SMS Server, which means that the application will not need to listen for messages, but can connect to the EE machine-to-machine (M2M) web service. The web service allows stored messages to be downloaded, which then can be read by, connecting to the EE simple object access protocol (SOAP) server. The application must employ a SOAP client that will connect to the EE SOAP server, in order to extract the board messages. To interpret the data of the messages it is important to deploy a an XML Parser as the stored messages are in the XML format. The status of the board requires a custom format that will be used for the rest of the data, this will once again be the XML language. The required data for the board status includes the following:

1. Statuses of the four switches - Which can be either ON or OFF.
2. Status of the fan - Which can be going FORWARD or REVERSING.
3. Last known number entered on the keyboard of the device.
4. The current temperature.

Table 3.1: Keys to format the data

Name	Description
Switch1	Status of switch 1 - 0(OFF) or 1(ON)
Switch2	Status of switch 2 - 0(OFF) or 1(ON)
Switch4	Status of switch 3 - 0(OFF) or 1(ON)
Switch4	Status of switch 4 - 0(OFF) or 1(ON)
F	Status of the fan - 0(FORWARD or 1 (REVERSE)
T	Current temperature
K	Last entered digit

Once the data has been downloaded and parsed it has to be stored, so that it is accessible if required. The way of supporting this idea is to construct a database to store this information, this will also be available to view for the users through the application. The database will ideally hold two tables, one with the information about the board (e.g the name) and the other holding the recent stored statuses (e.g temperature). Finally, once the data has been parsed and stored it can be displayed to the user. This will be deployed via a webpage dynamically created through the PHP language. The web pages will all be validated and finally styled using a cascading style sheet (CSS).

4 System Architecture

The application should be designed effectively and clearly. For the architecture of this product the model-view-controller(MVC) pattern has been chosen during implementation. The MVC architectural pattern is used to develop user interfaces, mostly for desktop applications, but can also be used for web applications such as this one. The purpose of it is to allow the developer to manage a complex system efficiently. The pattern can be broken down into the following:

- **Model** - The model highlights the data within the product/system, for example log-in information used in a system.
- **View** - The view, simply portrays the display of the data held within the model and presents it to the user. If the model data changes, so does the view.
- **Controller** - The controller is basically used for the logic of an application, in other words it handles the interactions. For example clicking a button performs a specific action.

Using the MVC effectively will help aid the overall management of this application.

4.0.1 Model

The model within this project will include the tables in the database that hold the information and statuses of the boards. This will be fulfilled with a relational database management system to allow safe interactions with a database. The open source RDBMS for this project will be MYSQL. The board information and statuses will be implemented with the following:

Table 4.1: Information Table

Name	Type	Key
msisdn	varchar(15)	Primary & Foreign
date	timestamp	

Table 4.2: Statuses table

Name	Type	Key
msisdn	varchar(15)	Primary & Foreign
date	timestamp	
switch1	enum('OFF', 'ON')	
switch2	enum('OFF', 'ON')	
switch3	enum('OFF', 'ON')	
switch4	enum('OFF', 'ON')	
fan	enum('FORWARD', 'REVERSE')	
temperature	int(3)	
keypad	int(1)	

The msysdn in each table will create the relation between the two, which maintains the idea of keeping only one status per board.

4.0.2 View

The view is quite simply the visual representation of the model. The view will be styled using a cascading style sheet, this will then allow users to see the relevant information in a neat and clear manner. The user interface should be easy to use and navigation should allow one to switch between pages such as board information and updates. The visual representation of the page might be as the following:

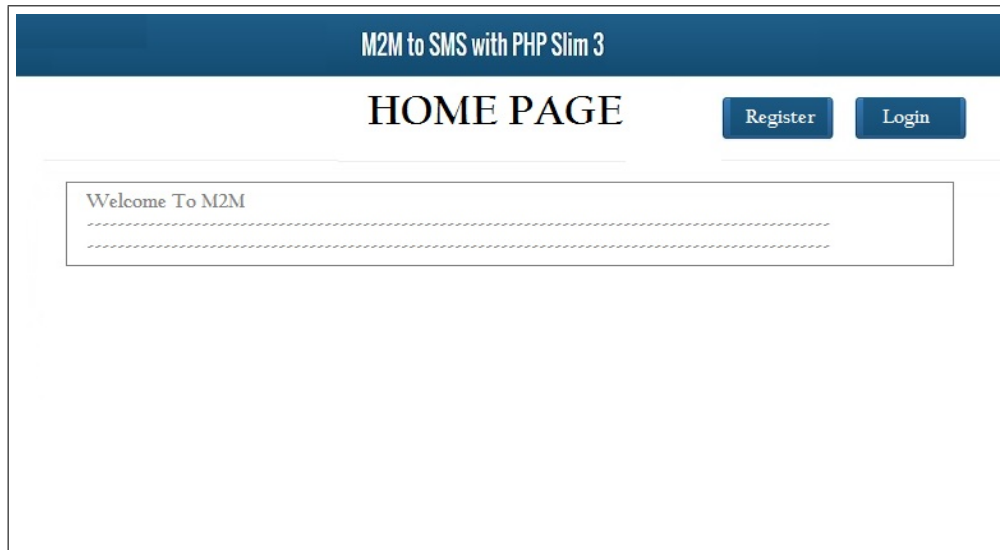


Figure 4.1: Possible representation of the home page

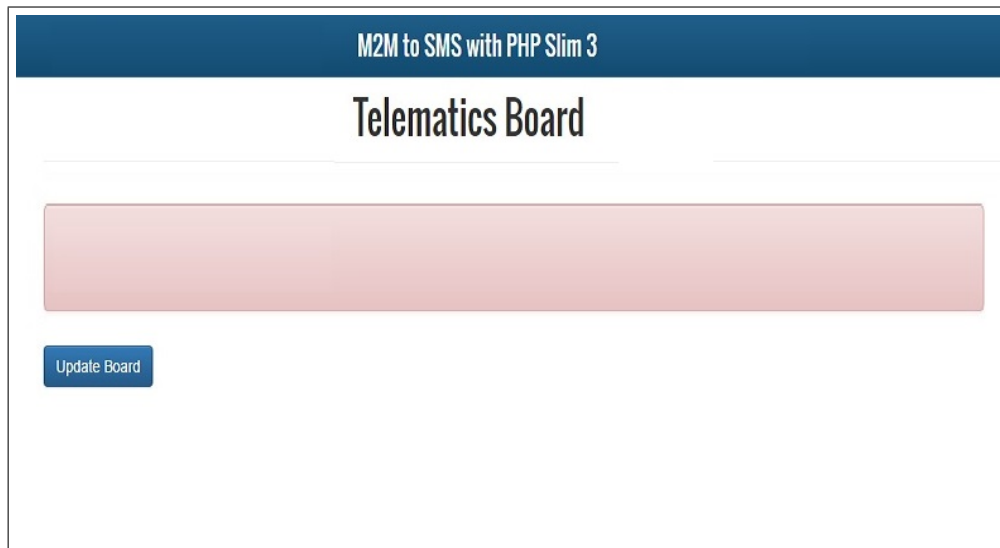


Figure 4.2: Possible representation of the telematics page

These figures should only be taken as a possible representation of the final application as changes are more than likely to be made throughout the development.

4.0.3 Controller

The controller is used for the logical aspect of the application. It will handle the interactions such as button clicks and other events (if applicable). The controller safeguards the model from security vulnerabilities as the user cannot directly manipulate the model. One of the possible security issues within the interface might be a user sending a status to server that will change or possibly delete the database.

The techniques of sanitisation and validation will be used to eliminate the possibilities of data poisoning. Sanitisation will ensure that the data only contains valid characters and validation will limit the data to a specific range.

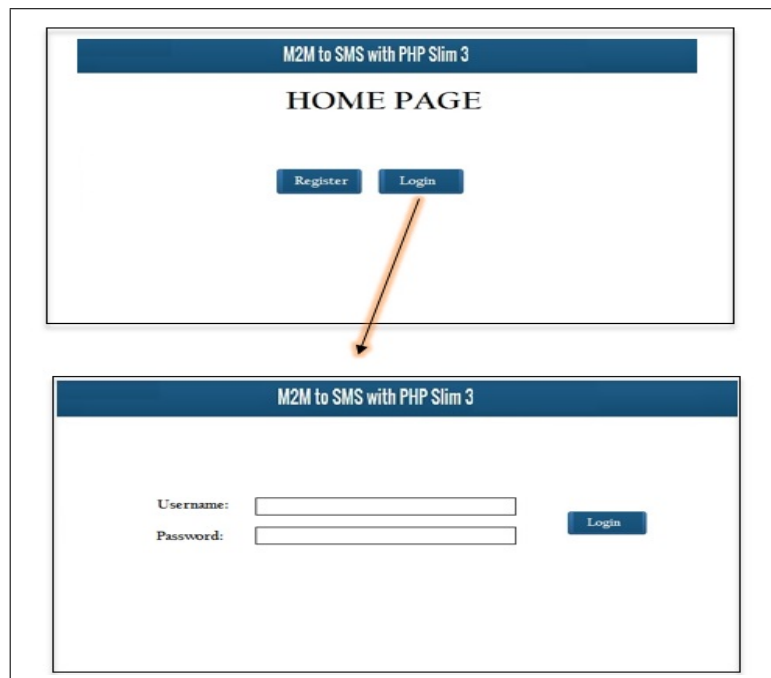


Figure 4.3: Typical controller event-action scenario

4.1 Use Case Scenarios

The application should contain functions to display the board statues and to check for updates. The following case scenarios give an idea of how the application should typically react:

4.1.1 Case Scenario: Board Status Display

Success Scenario:

1. User requests board status
2. Application - connects to and authenticates the database
3. Application - queries the database for board information and status
4. Application - combines the tables to get the statuses of the boards
5. Application - updates the model, which then displays the statuses.

4.1.2 Case Scenario: New Status Updates

Success Scenario:

1. User requests status updates
2. Application - connects to and authenticates the database
3. Application - queries the database for board information
4. Application - connects to the SOAP server
5. Application - downloads/parses the messages from the server
6. Application - updates the database
7. Application - updates the model, which then displays the statuses

4.2 Project Structure

The IDE for this project will be JetBrains PhpStorm 2016. This is the latest version of the IDE, it is quick and easy to use. The MVC architectural pattern will also be easily identifiable.

The following highlights the typical product structure in the chosen IDE, although the files might possibly change, the structure shall remain the same:

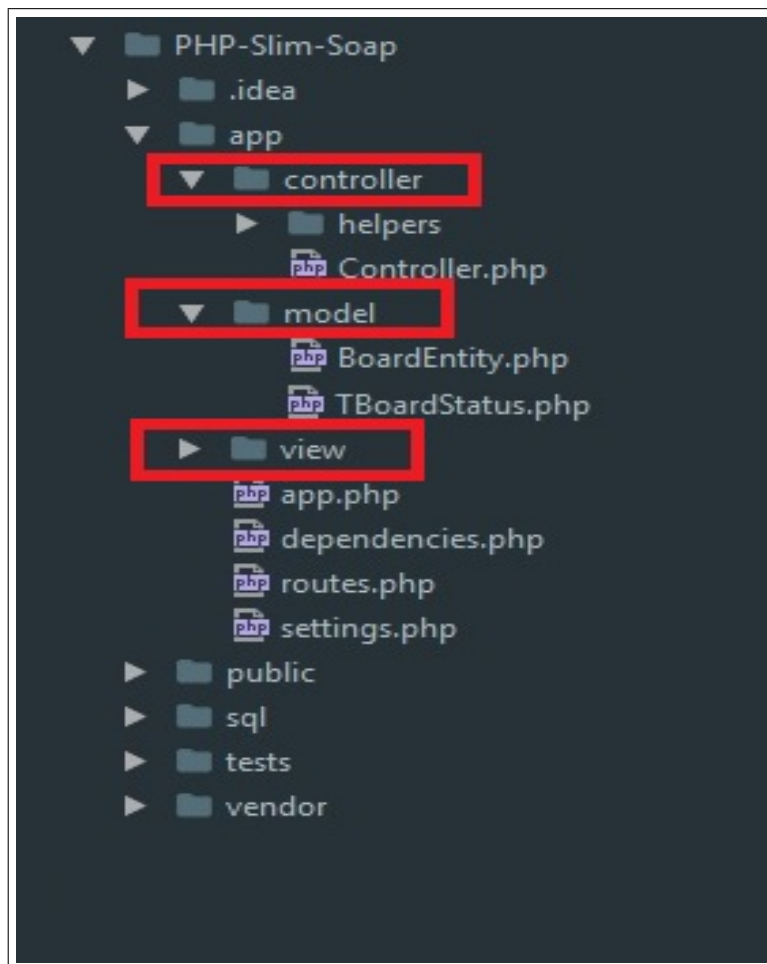


Figure 4.4: MVC Tree Structure of the 'current' project

4.3 UML Diagrams

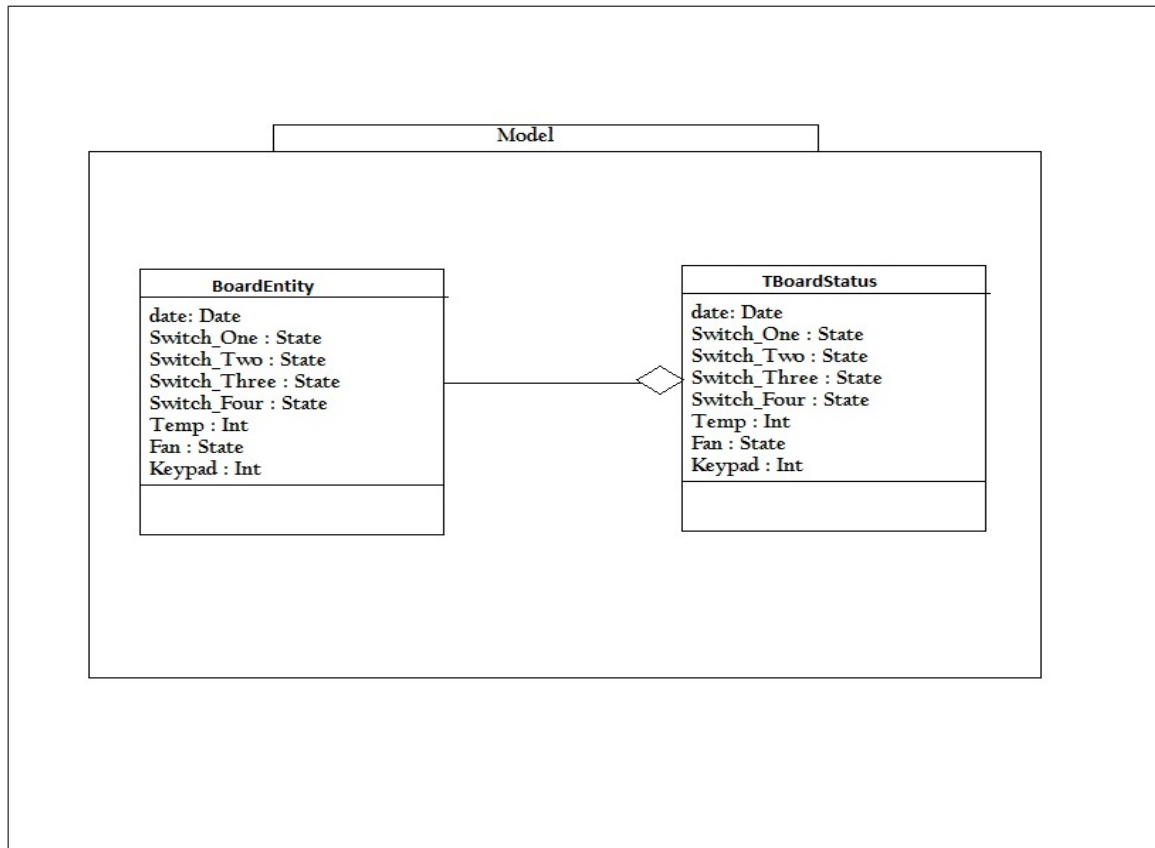


Figure 4.5: Model UML Diagram

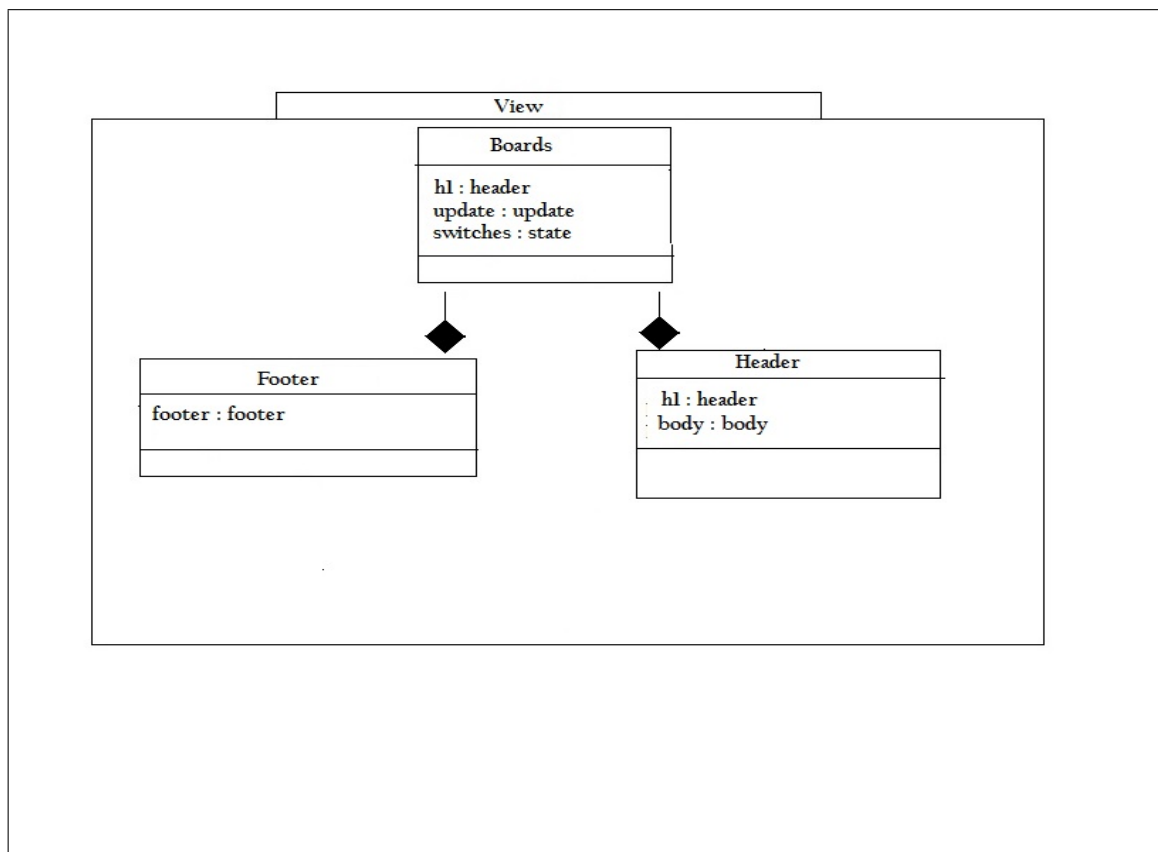


Figure 4.6: View UML Diagram

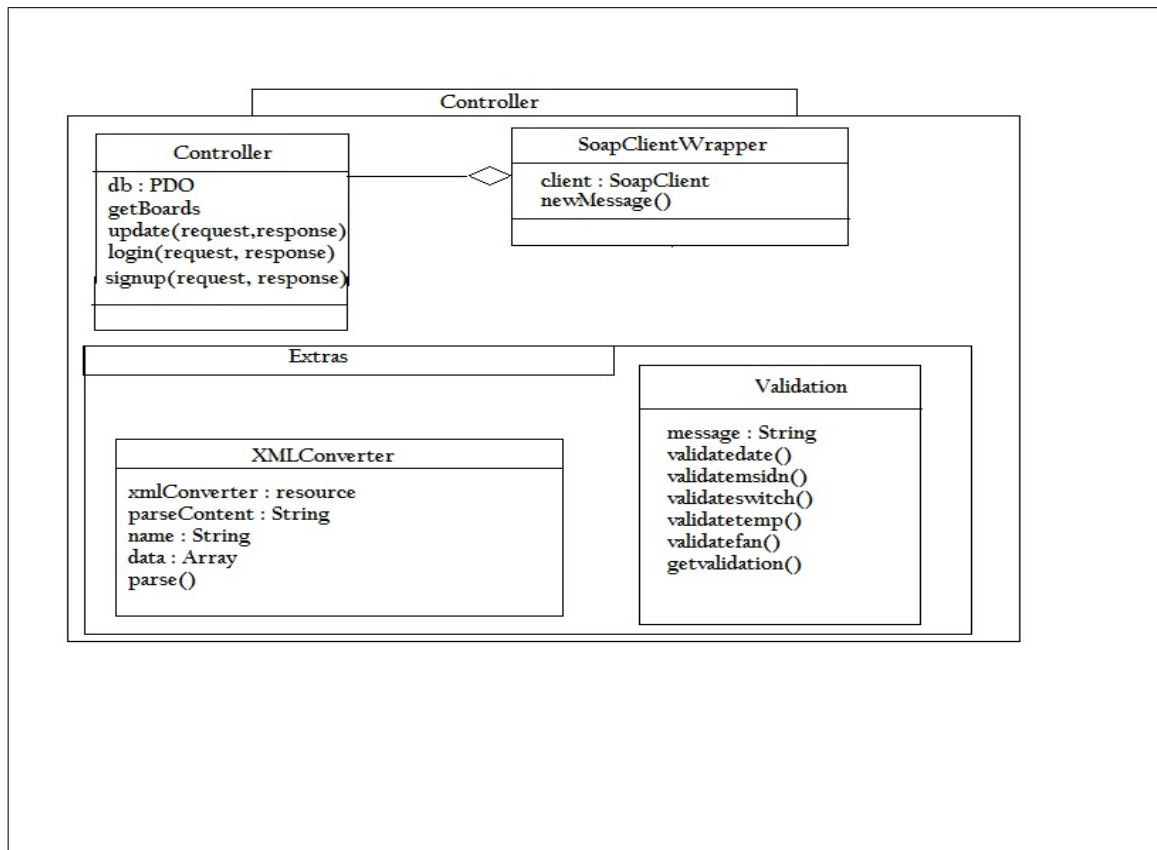


Figure 4.7: Controller UML Diagram

5 System Implementation

The implementation of the application is used to highlight how the design choices have been applied throughout. The framework for the following application is the SLIM micro-framework, as this helps to write simple but powerful applications (Lockhart & Smith, 2017).

5.1 Model-View-Controller Architecture

The product has been split into 3 main categories that imply to the MVC architecture. The directories are as follows:

- Application - This directory holds the MVC files:

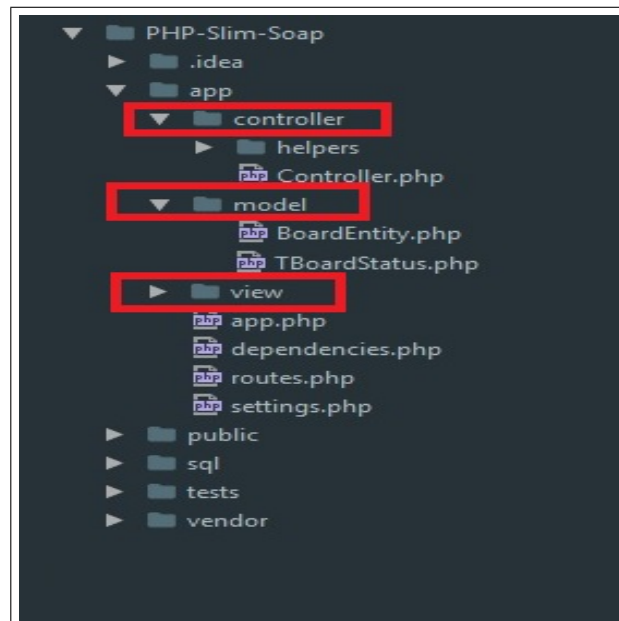
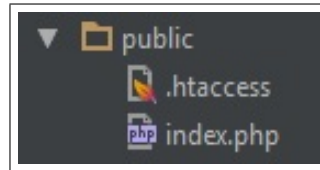


Figure 5.1: Application MVC structure

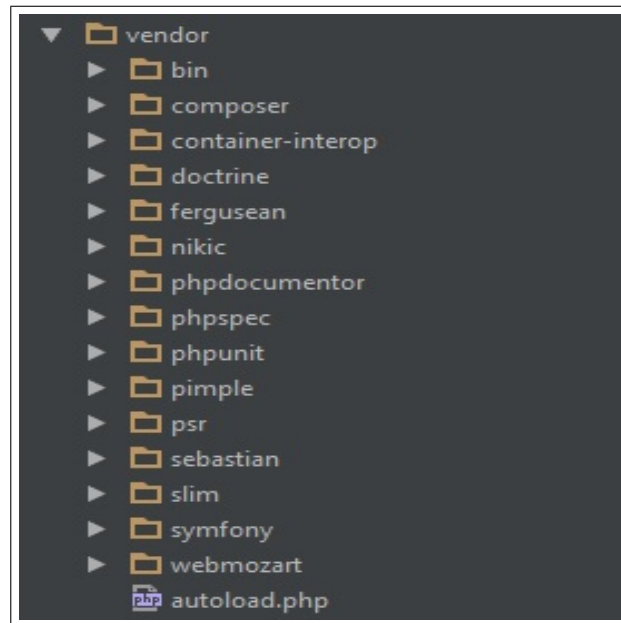
The controllers are used to control the SOAP client and the validation. The XML parser is also contained within a sub-directory in this directory. The Controller class is essentially the main part of the controller, it is used to retrieve the information from the database and sends it to the boards.phtml file (shown in the view section of Figure 5.1). The model directory is used to hold the data, this includes the information about the board and the statuses. The final directory is the view, which is essentially the user interface. The view is coded using phtml which is the extension for an HTML page that contains some PHP script.

- Public - This directory holds the public accessible files:



The index file loads the composer and the application and then finally runs it to display to a public interface. This is the single point of entry to the application and the significance of this is that it increases security because users can not execute other PHP scripts.

- Vendor - This directory holds the required libraries:



The vendor directory is composed of the libraries that are used throughout the application. Some of these libraries include the following: SLIM, phpunit and composer.

5.2 Security

The application security was an important aspect of the implementation and has been achieved through the use of validation techniques etc. An example of this could be the SMSValidator class which holds a variety of methods to certify sanitisation and validation that are applied to the data within the messages. The PHP filter_var function was used to allow various filters to sanitise the data and to remove unwanted chars.

A screenshot of a code editor showing the implementation of the validateTemp method in PHP. The code is written in a dark-themed editor with syntax highlighting. It defines a public function validateTemp() that takes no arguments. Inside the function, it sets a key to 'T', retrieves the value from \$this->getValue(\$key), and then applies two filters: FILTER_SANITIZE_NUMBER_INT and FILTER_VALIDATE_INT with options for min and max values. If the validation fails, it throws a FilterException. Finally, it returns the temperature as an integer.

```
public function validateTemp()
{
    $key = 'T';
    $temperature = $this->getValue($key);

    $temperature = filter_var($temperature, FILTER_SANITIZE_NUMBER_INT);
    $temperature = filter_var($temperature, FILTER_VALIDATE_INT,
        array('options' => array('min' => -99, 'max' => 999)));

    if ($temperature === false) {
        throw new FilterException($key);
    }

    return (int)$temperature;
}
```

Figure 5.2: Screenshot of method to validate the temperature

The method first checks the key to ensure that it has been set. The value must match the one that was specified earlier. The filter_var ensures that the variable is an integer, so any other format would throw an error. The first filter_var sanitises the data and removes unwanted chars that do not relate to the integer. The second filter_var validates that the returned value is an integer.

The class Controller, which relates to the values and manipulation of the database. Contains security measures within the methods to validate the data. For example if someone tries to inject malicious data into the database it will recognise it through this method. Furthermore, the database contains a table for users, each user is given a privilege that specifies what they can or cannot do to the database.

6 Test Plan

The application should be tested to meet the performance expectations. The testing of the application would ensure that any bugs are fixed before the development. Throughout the development of the application, basic testing was added to ensure the application met the standards.

```
public function runApp($requestMethod, $requestUri, $requestData = null)
{
    // Create a mock environment for the test
    $environment = Environment::mock(
        [
            'REQUEST_METHOD' => $requestMethod,
            'REQUEST_URI' => $requestUri
        ]
    );

    // Set up a request object depending on the environment
    $request = Request::createFromEnvironment($environment);

    // If we have request data, add it
    if (isset($requestData)) {
        $request = $request->withParsedBody($requestData);
    }

    // New response object
    $response = new Response();

    // Load settings
    $settings = require __DIR__ . '/../app/settings.php';

    // Instantiate the application
    $app = new App($settings);

    // Load dependencies
    require __DIR__ . '/../app/dependencies.php';

    // Load routes
    require __DIR__ . '/../app/routes.php';

    // Process the application
    $response = $app->process($request, $response);

    // Return the response
    return $response;
}
```

Figure 6.1: Testing Code

```
// Basic test for listing all boards at the home route
public function testHomeRoute()
{
    $response = $this->runApp('GET', '/');

    $this->assertEquals(200, $response->getStatusCode());
    $this->assertContains('All Boards', (string)$response->getBody());
    $this->assertNotContains('Page Not Found', (string)$response->getBody());
}

// Basic test to get the project update
public function testInfo()
{
    $response = $this->runApp('GET', '/board/update/');

    $this->assertEquals(200, $response->getStatusCode());
    $this->assertContains('M2M->SMS PHP Slim', (string)$response->getBody());
    $this->assertNotContains('Page Not Found', (string)$response->getBody());
}
```

Figure 6.2: Further Testing Code

7 Version Control

One of the main aspects of this project was teamwork and collaboration. To ensure all the members fulfilling the assigned tasks it was important to keep a history of the development of the application. For this reason the version control system was used. The chosen VCS was SVN, as this was the easiest to use to track the changes throughout the project. Furthermore, it was also important to use SVN to make the development apparent to the assessor.

7.1 SVN Commit History

The following is the full history of the commits throughout the development of the project:

(Following Page)




























Revision	Actions	Author	Date	Message
27		p14181072@LEC-ADMIN.DMU.AC.UK	26 January 2017 12:16:36	"Ben's Testing Added"
26		P14174331@LEC-ADMIN.DMU.AC.UK	26 January 2017 11:47:45	Updating file names and structures and refining code
25		p14181072@LEC-ADMIN.DMU.AC.UK	26 January 2017 00:29:00	History Of Commits Added.
24		p14181072@LEC-ADMIN.DMU.AC.UK	26 January 2017 00:23:25	"Extensions section completed + Final Product(Conclusion)"
23		p14181072@LEC-ADMIN.DMU.AC.UK	24 January 2017 15:50:27	Updating database details
22		p14181072@LEC-ADMIN.DMU.AC.UK	24 January 2017 15:44:31	Updating database(sql) statements.
21		p14181072@LEC-ADMIN.DMU.AC.UK	24 January 2017 15:43:39	Updating comments in controller.
20		p12210713@LEC-ADMIN.DMU.AC.UK	24 January 2017 15:26:40	Testing section & Extensions section partially added
19		p12210713@LEC-ADMIN.DMU.AC.UK	24 January 2017 15:23:50	Basic testing added
18		P14174331@LEC-ADMIN.DMU.AC.UK	24 January 2017 11:03:06	Adding full functionality and commenting the code
17		P14174331@LEC-ADMIN.DMU.AC.UK	23 January 2017 15:50:23	Updated all phtml pages and added most of the controller functionality
16		p14181072@LEC-ADMIN.DMU.AC.UK	17 January 2017 14:40:44	Added the version control section (Incomplete) and also the References
15		P14174331@LEC-ADMIN.DMU.AC.UK	17 January 2017 10:41:03	Adding entire view system and implementing the routes and overall functioning of the project including the app dependencies routes and settings files
14		P14174331@LEC-ADMIN.DMU.AC.UK	17 January 2017 10:22:36	Starting implementation of controller
13		p14174331@LEC-ADMIN.DMU.AC.UK	16 January 2017 17:37:39	Model files developed to hold the information of the telematics board from the database and the m2m server
12		p14174331@LEC-ADMIN.DMU.AC.UK	16 January 2017 17:30:14	Updating composer with relevant libraries and information & further changes to file structure
11		p14174331@LEC-ADMIN.DMU.AC.UK	16 January 2017 17:22:36	Cleaning up file structure and beginning MVC implementation
10		p14181072@LEC-ADMIN.DMU.AC.UK	16 January 2017 17:17:08	Grammatical changes + implementation section completed and added.
9		p14174331@LEC-ADMIN.DMU.AC.UK	16 January 2017 17:15:33	Editing index.phtml to reflect project
8		p14181072@LEC-ADMIN.DMU.AC.UK	11 January 2017 11:16:48	Contract, Specification & System Architecture added.
7		P14174331@LEC-ADMIN.DMU.AC.UK	09 January 2017 17:24:27	Slim 3 Skeleton application
6		P14174331@LEC-ADMIN.DMU.AC.UK	09 January 2017 17:21:00	
5		P14181072@LEC-ADMIN.DMU.AC.UK	09 January 2017 17:17:54	Introduction section of the documentation completed.
4		P14181072@LEC-ADMIN.DMU.AC.UK	09 January 2017 17:02:42	Initial Commit (Soap Client)
3		P14181072@LEC-ADMIN.DMU.AC.UK	18 November 2016 11:56:33	Initial start to the project, adding relevant directories
2		P14181072@LEC-ADMIN.DMU.AC.UK	18 November 2016 11:51:21	
1		P14181072@LEC-ADMIN.DMU.AC.UK	18 November 2016 11:49:52	

Figure 7.1: History Of Commits

8 Extensions

Once the application had been completed to meet all the mandatory requirements, there was time to implement extra functionalities to make the application further successful.

The first extension was used for **logging into the application**. The HTTP authentication uses middleware (manipulate request and response objects) built into the slim framework and also hashed passwords, the following is an example of the code:

```
/**
 * Adds HTTP Authentication with hashed passwords
 */
$app->add(new \Slim\Middleware\HttpBasicAuthentication([
    "realm" => "Protected",
    "users" => [
        "root" => "$2y$10$1lwCIlqktFZwEBIppL4ak.I1AHxjoKy9stLnbedwVMrt92aGz82.0",
        "somebody" => "$2y$10$6/vGXuMUoRLJUeDN.bUWduge4GhQbgPkm6pfyGxwgEWT0vEkHKBuW"
    ],
    "error" => function ($request, $response, $arguments) {
        $data = [];
        $data["status"] = "error";
        $data["message"] = $arguments["message"];
        return $response->write(json_encode($data, JSON_UNESCAPED_SLASHES));
    }
]));
```

Figure 8.1: Code for implementing the basic authentication prompt

Another extension implemented into the program is a **graph to represent the board temperatures**. The Fahrenheit is calculated and then added to the graph which then simultaneously updates the table as required. The graphs are implemented using the java-script charts from canvasJS in real-time from the information extracted from the database. The following is an example of the graph:

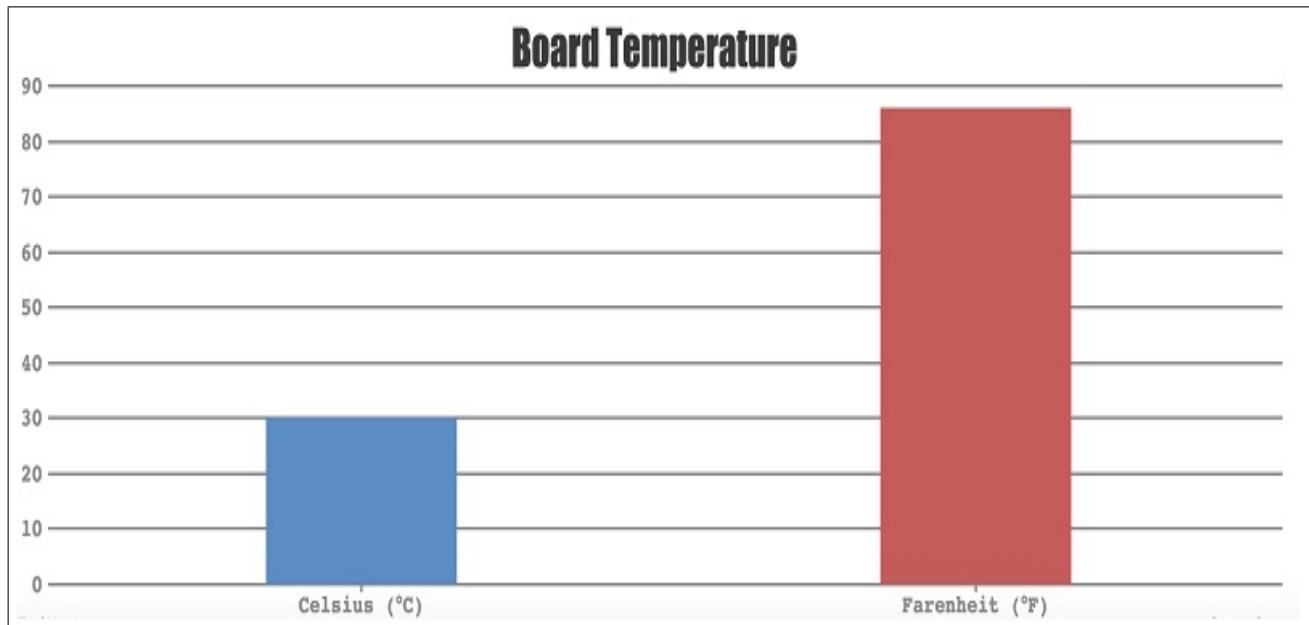


Figure 8.2: Graph showing board temperatures

Further functionalities also include using HTML5 for the style and design of the website. The website provides basic validation in the sign-up and login forms. The routes for the pages are also implemented.

9 Conclusion

An overall view of the project helps identify how successful the final application is. The functionalities and ideas that was specified throughout have been implemented successfully. Furthermore, the application reacts as it should be without any bugs or any other issues and the user interface is easy to use and navigate around to find the specific information required.

The final product is as follows:

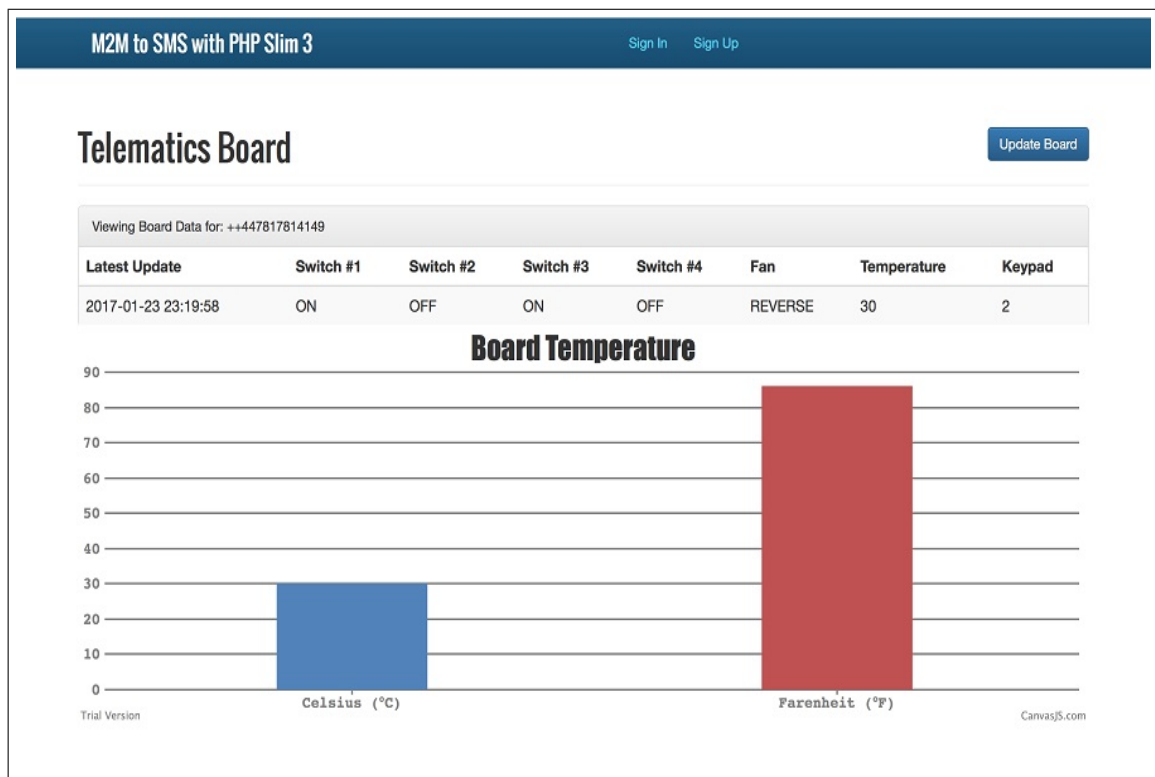


Figure 9.1: Completed Final Product

References

- [1] Lockhart, J & Smith, A. (2017). Slim A Micro Framework. Available: <https://www.slimframework.com/>. [Accessed: 26th Dec 2016]
- [2] Plivo. (2016). What is the character limit for SMS text messages?. Available: <https://www.plivo.com/faq/sms/what-is-the-character-limit-for-sms-text-messages/>. [Accessed: 21st Dec 2016]
- [3] Triggs, R. (2013). What is SMS and how does it work?. Available: <http://www.androidauthority.com/what-is-sms-280988/>. [Accessed: 21st Dec 2016]
- [4] TSS. (2000). Why Prepared Statements are important and how to use them "properly". Available: <http://www.theserverside.com/news/1365244/Why-Prepared-Statements-are-important-and-how-to-use-them-properly>. [Accessed: 22nd Dec 2016]