Project: Web Application Development Final Project
Module: Web Application Development
Course: BSc in Web Technologies and Programming
Lecturer: Kevin O'Brien
Student: Leonard M Reidy
Student #: G00341288

Setup

To install and run the application, you will need to have an appropriate development environment. Prerequisites include:

* HTTP Server (Apache)
* PHP interpreter
* MySQL DB (MariaDb)
* Git
* NodeJS
* Bower

Acquire the Source Code and Install the XAMPP/WAMPServer

First clone or download the application github repository from here. Next you will need a working HTTP server, a database, and a language interpreter for PHP at a minimum. XAMPP or WAMP are suitable choices. XAMPP, much like WAMPServer, combines the Apache HTTP server, a MySQL database (MariaDB), and a selection of interpreters for scripts written in several languages, including PHP and Perl. XAMPP also comes bundled with phpMyAdmin a dashboard front-end to MySQL/MariaDB. The default XAMPP installation should be appropriate for this purpose. If you have not installed it already, you can find XAMPP here.

Once you have installed XAMPP or WAMPServer, the phoMart root directory :

```
phoMart-master
```

should be transferred to the directory used by the associated Apache install to serve files on your domain/system. On Ubuntu, for example, this is:

```
/opt/lampp/htdocs
```

Please see the documentation provided by XAMPP or WAMP for your system.

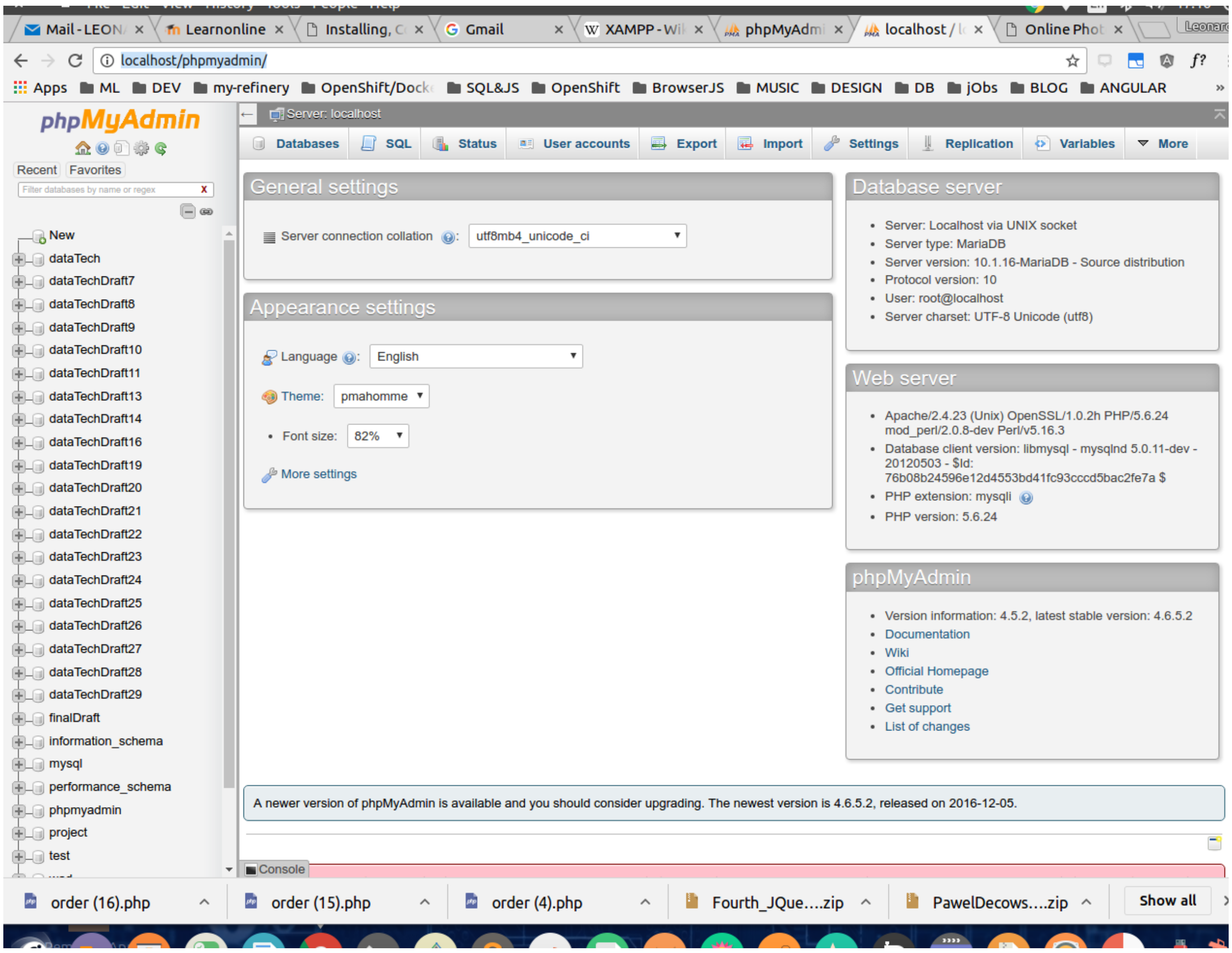
Prepare the Database

Once you have installed XAMPP/WAMP, you will need to fire it up and take steps to create a new database. To do this:

1. Navigate to the phpMyAdmin dashboard in your browser at:

```
localhost/phpmyadmin
```

You should now see a screen that looks like the one below. In that screen, click on the ‘New’ link in the sidebar to create a new database.



General settings

Server connection collation: utf8mb4_unicode_ci

Appearance settings

Language: English

Theme: pmahomme

Font size: 82%

More settings

Database server

- Server: Localhost via UNIX socket
- Server type: MariaDB
- Server version: 10.1.16-MariaDB - Source distribution
- Protocol version: 10
- User: root@localhost
- Server charset: UTF-8 Unicode (utf8)

Web server

- Apache/2.4.23 (Unix) OpenSSL/1.0.2h PHP/5.6.24 mod_perl/2.0.8-dev Perl/v5.16.3
- Database client version: libmysql - mysqlnd 5.0.11-dev - 20120503 - $Id: 76b08b24596e12d4553bd41fc93cccd5bac2fe7a $
- PHP extension: mysqli
- PHP version: 5.6.24

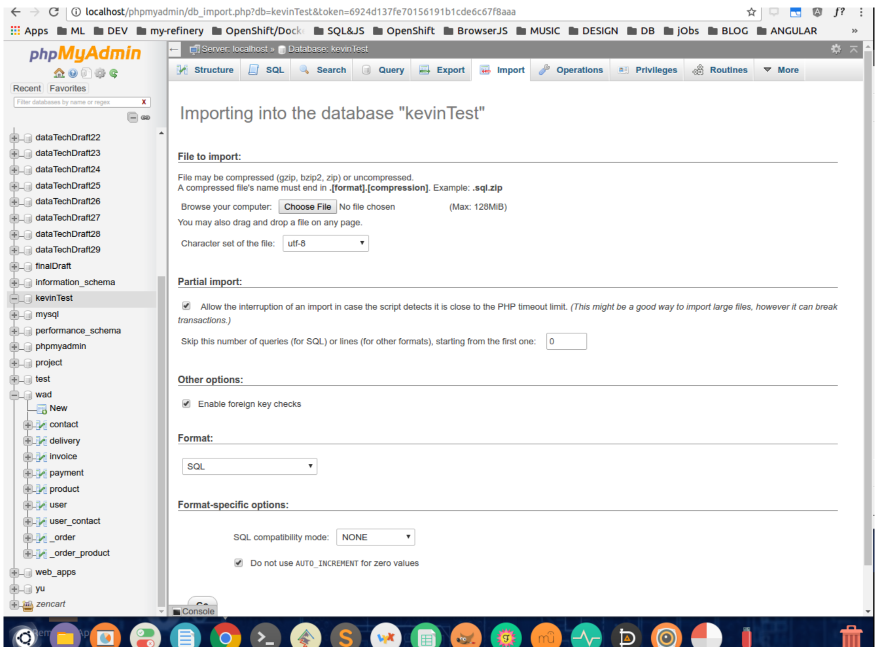
phpMyAdmin

- Version information: 4.5.2, latest stable version: 4.6.5.2
- Documentation
- Wiki
- Official Homepage
- Contribute
- Get support
- List of changes

A newer version of phpMyAdmin is available and you should consider upgrading. The newest version is 4.6.5.2, released on 2016-12-05.

Give the new db an appropriate name and choose a collation if necessary. The default collation is acceptable. Then, click ‘Create’.

On the next screen, look for the ‘Import’ link and click it. Once you have clicked it, the main pane will update and it will now look like the image below.



Importing into the database "kevinTest"

File to import:

File may be compressed (gzip, bzip2, zip) or uncompressed.
A compressed file's name must end in **.[format].[compression]**. Example: **.sql.zip**

Browse your computer: Choose File No file chosen (Max: 128MiB)

You may also drag and drop a file on any page.

Character set of the file: utf-8

Partial import:

☑ Allow the interruption of an import in case the script detects it is close to the PHP timeout limit. *(This might be a good way to import large files, however it can break transactions.)*

Skip this number of queries (for SQL) or lines (for other formats), starting from the first one: 0

Other options:

☑ Enable foreign key checks

Format:

SQL

Format-specific options:

SQL compatibility mode: NONE

☑ Do not use `AUTO_INCREMENT` for zero values

Click ‘Choose File’ and navigate to the recently downloaded `phoMart-master` directory, and choose the database schema SQL file (`wad.sql`). This is located in the following directory:

```
phoMart/server/db/wad.sql
```

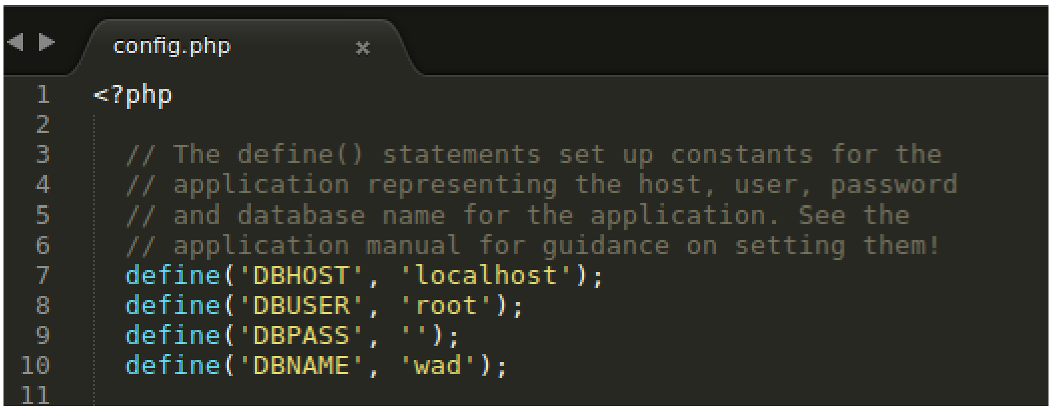
4. Click ‘Ok’ to select the file, then, when the dialogue closes, find the ‘Go’ button towards the end of the pane and click it to upload the schema. PhpMyAdmin should report that the import was successful. If you examine the database in the phpMyAdmin interface you will notice that all but one of the tables are empty. The product table has been prepared with a small selection of products because it is used to dynamically populate the main store page **(`index.php`)** in the project web application.

Configure the application

1. To configure the application before running, open

```
phoMart-master/server/db/config.php
```

in a suitable text editor (ideally a developer’s editor or IDE) and look for the following lines:



```php
<?php

    // The define() statements set up constants for the
    // application representing the host, user, password
    // and database name for the application. See the
    // application manual for guidance on setting them!
    define('DBHOST', 'localhost');
    define('DBUSER', 'root');
    define('DBPASS', '');
    define('DBNAME', 'wad');

```

The `define()` statement is used here to set up application constants representing the host, user, password and database name associated with the application database. To ensure the application is configured correctly, add the appropriate values as the second argument to each call to the `define()` function. If you are developing locally, it is not necessary to change the first three lines. In a default XAMPP installation, localhost and root will already be set up with all privileges. If you made changes to the default installation, for example to secure it, then you will need to use the user and password values that correspond to your environment.

2. Once you are confident that these settings are correct, and assuming that you did not shutdown XAMPP or any of the associated servers (Apache, MySQL), open a web browser and navigate to the following url:

```
localhost/phoMart/server/
```

This will default to index.php, the home page for the application and if you followed the procedure above it should be working correctly and visible in the browser.

Client-side Dependencies

The application exploits AngularJS and a number of third-party Angular modules, as well as jQuery. These can be installed painlessly with the bower package manager. The easiest way to install bower is to first install git, and NodeJS. The NodeJS installers come bundled with npm, the Node package manager. This can be used to install bower. To install bower with npm, execute the following command at the command line:

```
npm install -g bower
```

The phoMart root directory includes a bower manifest file: bower.json. This collects information about the application's dependenices and is used by the package manager to identify, download and install them. To install the application's client-side dependencies execute the following at the command line in the root directory:

```
bower install
```

Note that this may not be necessary. The `bower_components` directory is under version control and so the application should work out of the box. But if not, this is a good place to start troubleshooting!

Application Architecture

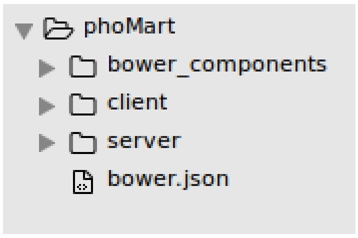
The general structure of the application broadly observes the client/server architectural design pattern. Application directory structure reflects this. Client code is organised under a ‘client’ directory, and server code is organised under a corresponding ‘server’ directory at the root of the application. However, responsibilites are shared somewhat unevenly between client and server in this application. This is because some of the client modules utilise AngularJS to bear key front-end responsibilities, while others were written in vanilla JavaScript or jQuery to satisfy key project specification requirements more transparently. AngularJS applications typically provide client-side functionality that might previously have belonged to server-side technologies. For example, a products page in a more traditional PHP application might be populated dynamically on the server side before being served to the client. In this application, the products are retrieved from the database by an AJAX request and the web page is updated on the client side using AngularJS’s built-in `ng-repeat` directive to iterate over the returned data representation and populate the page dynamically.

Broadly speaking, the client portion of the application exploits Angular’s MV* design strategy. The current iteration of AngularJS 1 (as opposed to AngularJS 2) may be described as a combination of the Model-View-Controller (MVC) and the Model-View-ViewModel (MVVM) architectural design patterns. But a lack of consensus in the community about AngularJS encourages many to describe it as model-view-whatever, MV*, or, model-view-viewmodel (MVVM). The essential idea is that these patterns promote design decisions that make the application more modular, more scalable, more extensible, more maintainable, and easier to understand. In keeping with this strategy, the application is divided out into modules and submodules, services, and controllers, in a way that favours loose coupling and the separation of concerns, although it would probably take another few iterations to get to a point where I was fully satisfied with the result.

Note that all source files are commented in detail.

Folder Structure

The application architecture is reflected in the directory structure. The `phoMart-master` application root directory contains three directories: (i) `bower_components`; (ii) `client`; and (iii) `server`.



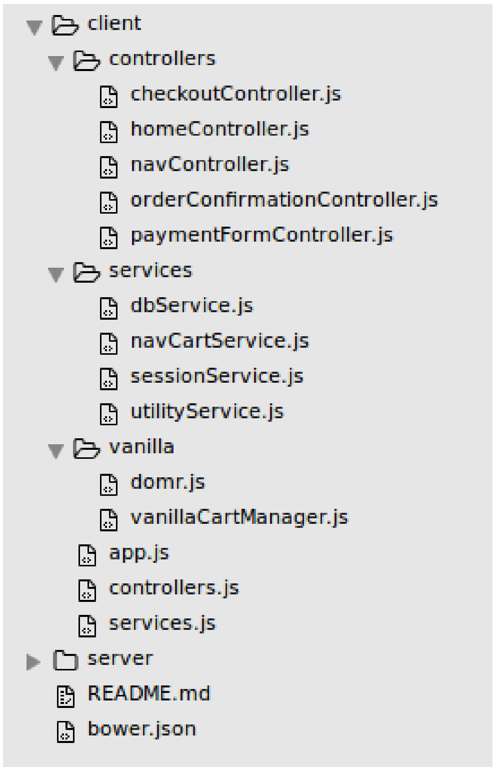
```
▼ phoMart
  ▶ bower_components
  ▶ client
  ▶ server
    bower.json
```

`bower_components` contains application JavaScript dependencies added to the project using bower, a web package manager.

`client` contains code associated with the client-side components of the application.

`server` contains code associated with the server-side components of the directory.

The phoMart-master directory also contains a file called `bower.json`. This is the bower manifest file and is used to install client-side dependencies.



```
▼ client
  ▼ controllers
      checkoutController.js
      homeController.js
      navController.js
      orderConfirmationController.js
      paymentFormController.js
  ▼ services
      dbService.js
      navCartService.js
      sessionService.js
      utilityService.js
  ▼ vanilla
      domr.js
      vanillaCartManager.js
    app.js
    controllers.js
    services.js
▶ server
  README.md
  bower.json
```

The `client` directory contains three subdirectories: (i) `controllers`; (ii) `services`; and (iii) `vanilla`.

`controllers` contains AngularJS application controller JavaScript files.

`services` contains AngularJS custom services.

`vanilla` contains client-side application components created using vanilla JavaScript and jQuery. Application views are mostly implemented in AngularJS and PHP. But the cart view is implemented using jQuery and PHP.

Controller and service files define application submodules which are registered with the AngularJS phoMart module in `client/controllers.js` and `client/services.js`. The Angular phoMart module is defined in `app.js`

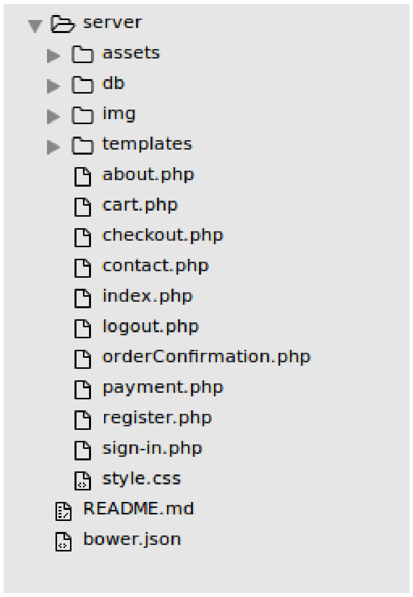
The server directory contains four subdirectories: (i) assets; (ii) db; (iii) img; and (iv) templates, alongside 10 PHP files and 1 CSS file.

`assets` contains third party styles, JavaScript (Bootstrap), associated fonts, jQuery, and a directory of images of products for sale on the site.

`db` contains PHP files for server-side interactions with the MySQL db, the SQL schema file for the wad db (`wad.sql`).

`img` contains the `phones` subdirectory which in turn contains images of the phones on sale on the site.

`templates` contains PHP files for page partials including the footer, head, nav, scripts and payment form related partials. All the site page PHP files are contained in the server directory also.



```
▼ server
    ▶ assets
    ▶ db
    ▶ img
    ▶ templates
      about.php
      cart.php
      checkout.php
      contact.php
      index.php
      logout.php
      orderConfirmation.php
      payment.php
      register.php
      sign-in.php
      style.css
  README.md
  bower.json
```

Mapping the Project Requirements

The following table indicates where in the project directory to find the files that satisfy the project requirements.

Core Requirements

#	**Files**
1	`server/sign-in.php, server/logout.php, server/register.php, server/db/config.php`
2	`server/index.php, client/controllers/homeController.js`
3	`server/cart.php, client/vanilla/domr.js, client/vanilla/vanillaCartManager.js`
4	`server/cart.php,server/checkout.php, client/controllers/checkoutController.js`
5	`client/vanilla/vanillaCartManager.js`
6	`client/vanilla/vanillaCartManager.js`
7	`server/checkout.php,client/controllers/checkoutController.js, server/payment.php, server/templates/paymentForm.php, server/templates/paymentFormWarning.php, client/controllers/paymentFormController.js, server/db/openOrder.php`
8	`See above...`
9	`server/templates/paymentForm.php, server/templates/paymentFormWarning.php, client/controllers/paymentFormController.js, db/updateUserPaymentDetails.php, server/db/config.php, server/db/closeOrder.php, server/orderConfirmation.php, client/controllers/orderConfirmationController.js`

Please note that the application design is highly modularised and each module bears some part of the applications responsibilities so the code that satisfies each requirement is quite spread out. But here I will identify the file and line numbers for code that satisfies key project criteria:

1. Hash encrypted password (`server/register.php [line 105]`), database persisted credentials (`server/register.php [line 111]`), registration (`server/register.php [all]`) and validation (`server/register.php [lines 42ff]`).
2. When logged in/registered on website, user should be taken to a home page (`server/index.php, client/controllers/homeController.js`) where the product details can be viewed as above.
3. Must include an option to add products to a cart (`server/cart.php, client/vanilla/domr.js, client/vanilla/vanillaCartManager.js`), using localStorage to persist products across page boundaries (`client/controllers/homeController.js,  .`
4. Must have a separate page where products added to cart can be viewed with price details and total costs (`server/cart.php,server/checkout.php)`, `client/controllers/checkoutController.js.`
5. The structure on the web page must include an event listener at parent level (e.g., if a table, then put event listener on table element and not sub elements such as tr, td, th. See (`client/vanilla/vanillaCartManager.js [lines 232ff])`
6. The event listener must be implemented using JavaScript, and is invoked when the user clicks on an order item. The event flow must be bubble. The event will trigger a function that deletes the item clicked on. Total costs must be recalcuated based on the revised list of products now in the cart; update localStorage object to remove the deleted item. (`client/vanilla/vanillaCartManager.js`).
7. The web page which includes the event, must also include a proceed button which will take the user to a new page to enter payment details (`client/vanilla/vanillaCartManager.js, server/cart.php, client/controllers/checkoutController, server/checkout.php)`. Payment page dependencies: `(server/payment.php, server/templates/paymentForm.php,server/templates/paymentFormWarning.php,client/controllers/paymentFormController.js`).
8. Payment details web page must include credit card validation. `(server/payment.php, server/templates/paymentForm.php,server/templates/paymentFormWarning.php,client/controllers/paymentFormController.js`).
9. Once payment is validated, the order must be written to a database table – use PHP script for this purpose. (`db/updateUserPaymentDetails.php, server/db/config.php, server/db/openOrder.php, server/db/closeOrder.php`) The order id then must be displayed on a webpage confirming the order was successfully placed `server/orderConfirmation.php [line 69], client/controllers/orderConfirmationController.js.`

Additional Requirements

* A brief user guide (this, a copy of which is included in the application root directory)
* Styled with an appropriate theme (`server/style.css`)

Additional Requirements

* Framework (Bootstrap, AngularJS)