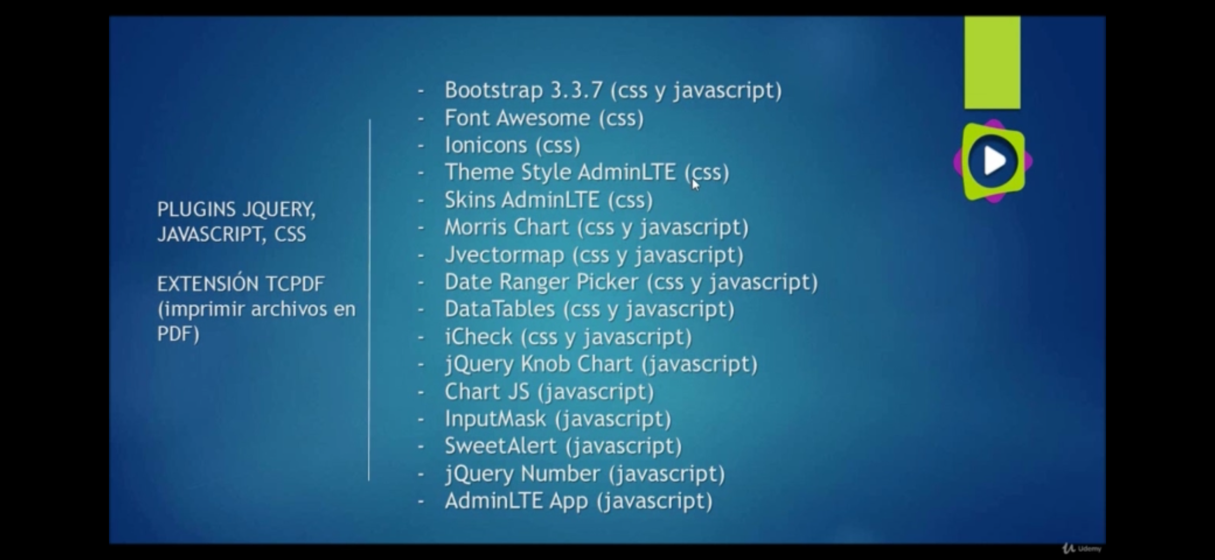
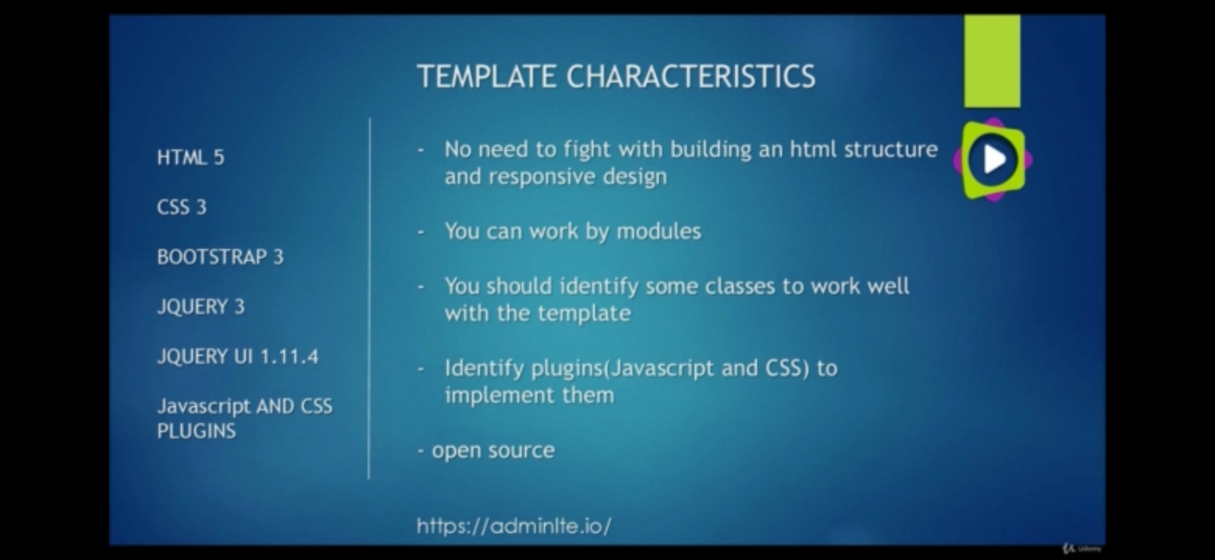
1. Download the admin LTE and paste it inside the htdocs folder
2. Local Server - <https://github.com/ColorlibHQ/AdminLTE/releases>
3. Code editor - Check the timeline, multi level
4. Sublime personal settings Download the plugins
5. Comm-html-section + tab = comment section
6. Command + alt + n = create new file
7. Cmd + ctrl + G = select and replace similar selected characters
8. Admin LTE template
9. Plugins and extensions





* Morris chart >> <https://morrisjs.github.io/morris.js/>
* Date ranger picker >>> <https://www.daterangepicker.com/>
* Data tables >>> <https://datatables.net/>
* jVector map >>> <http://jvectormap.com/>
* jQuery knob >>> <https://github.com/aterrien/jQuery-Knob> and <http://anthonyterrien.com/demo/knob/>
* sweet alert >>> <https://lipis.github.io/bootstrap-sweetalert/>
* jQuery number >>> <https://plugins.jquery.com/df-number-format/>
* chart js >>> <https://www.chartjs.org/>
* tcpdf >>> <https://tcpdf.org/>

1. Rename the source code and sql to pos-copy and sql-copy

access phpMyAdmin, set up the collation to UTIF-8 name it as pos-copy, then create

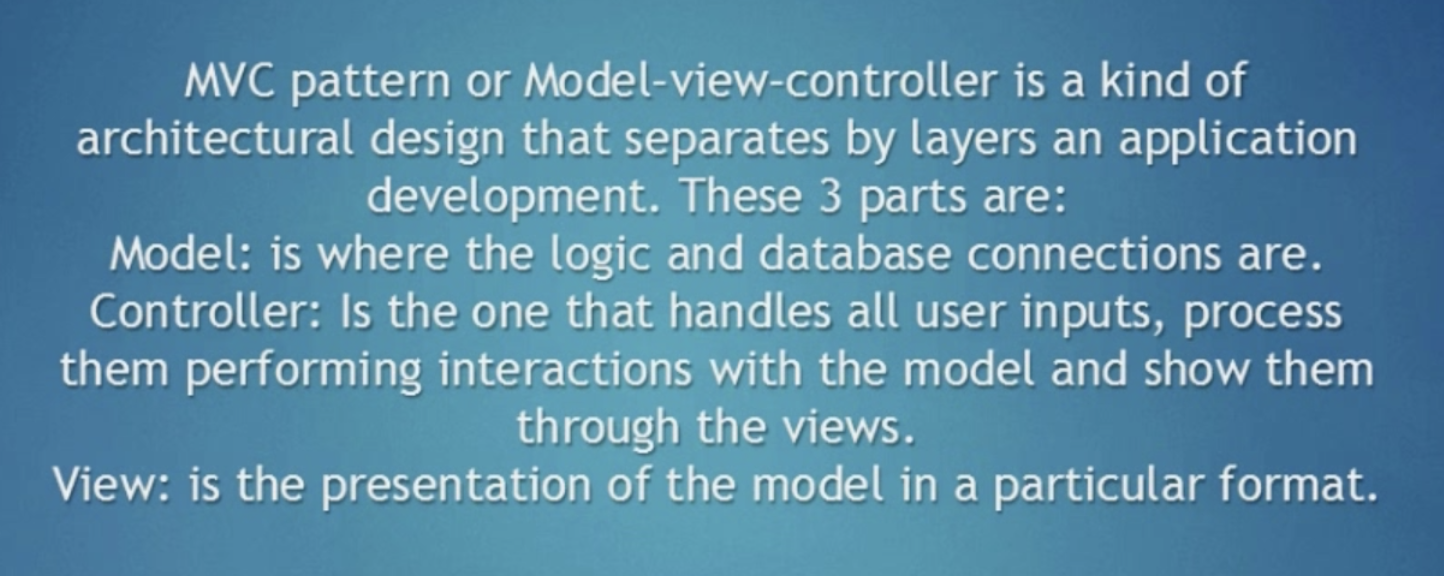
import the sql-copy. Rename the database in /models/connection.php and use sql-copy

<http://localhost:8080/phpmyadmin/>

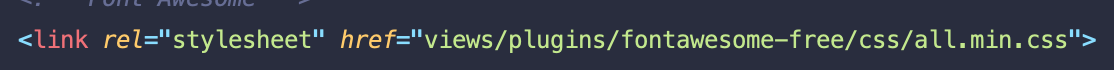
* Syblime text shortcuts = <https://gist.github.com/mrliptontea/4c793ebdf72ed145bcbf>

**Section 2 – AdminLTE and Model-View-Controller**

1. **Adapting adminLTE to MVC pattern**



1. Setup the MVC folders inside the htdocs:
2. Main folders

* model
* views >>> create a template.php and models subfolder. Copy the dist, bower components and plugin folders inside the views folders.
  + Copy the content of blank.html to template.php as a sample. Revise the link of the scripts and links to point to views
  + 
* controller

1. Create the ajax folder, this is the location of the js which will allow to interact to database using js.
2. Create the extensions folder, this will be the location for the plugins
3. Create the .htaccess
4. **Configure the controller**
5. at index.php,

“require\_once “controllers/template.controllers.php”

do not close the php tag so the succeeding syntax are all written in php

$template = new ControllersTemplate();

$template -> ctrTemplate();

1. Create the template.controllers.php in controllers folder and create the ControllersTemplate class

Create a method under ControllerTemplate class in tempate.controller.php



1. Architecture:

Create the CRUD Manager methods for Users, clients, categories, products and sales using the CRUD inquiry to database and utilize the views -> controller -> models -> views pattern

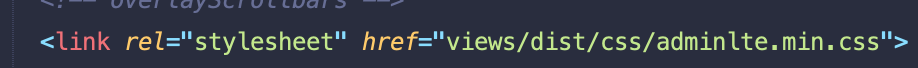
1. **Adding module to POS system**
2. Advisable screen size to configure for web app

Mobile 320px

Tablet portrait 768px

Table landscape 1024 px

Desktop 1366 px

1. Customize the template.php
2. 



Do not use the min version.

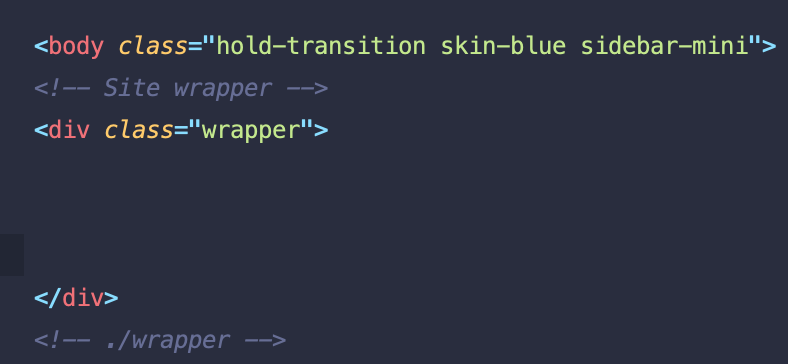
1. Transfer the scripts on top before the head close
2. Create the template.js inside js folder. This is customize and will make the template.php be recognize by html. Add it at the bottom of the template.

This will be the main javascript file of the project. Remove demo.js in the head of the template.php. all the js to be execute will be written in template.js. this will replace the demo.js in the template.php

1. Delete the long <header> and <aside>, content-wrapper and <footer> make it as a module.
2. **Adding modules to POS system**
3. Open a php tag and change the syntax from html to php



1. Create the header.php inside the modules folder. Can copy the main-header section from index.html

Codes here >>> . Change the syntax to PHP

Start the code by opening and closing a PHP tag

1. Create the header.php



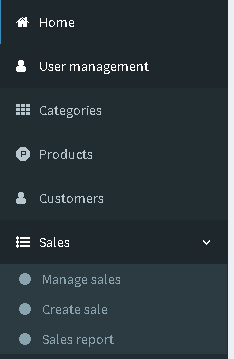
- Header.php content, can refer to main-header of the index.html under plugins folder

* + Logo; Create the img folder under views and create template and products folder inside it
  + Sidebar
  + User profile
  + Drop down toggle
  + Create the img folder under views and create template and products folder inside it
  + Output of the header.php



1. **Adding modules**

* Add the sidebar
* Create the menu.php inside the modules folder and include it in the template.php
  + This includes the side bar
  + Refer to content section of the blank.html
  + Output of the code

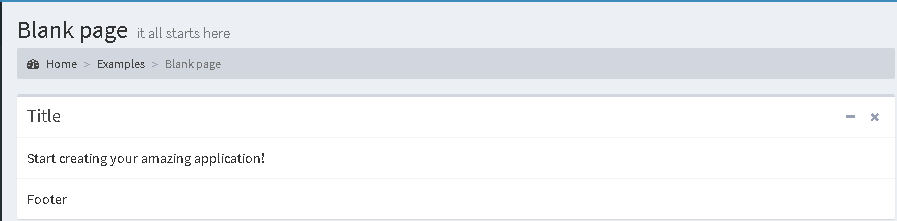
 

- Add the footer

* + Create the footer.php under \views\modules
  + Add and call it in template.php

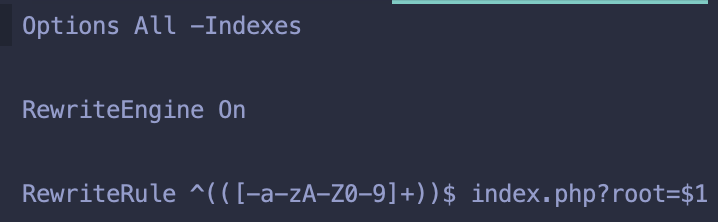


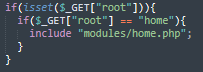
* Create the content.php
  + Can copy the content section from the blank.html
  + Refer to content section of the blank.html
  + Output of the cont.php



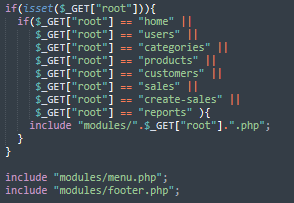
1. **Creating the views, controllers and models**. Create the CRUD Manager methods for Users, clients, categories, products and sales using the view >>> controllers >>> models >>> view architecture

* Customize the content.php, suggest to remove the examples, change the blank page to dashboard and setup the hyperlinks later. This will be the main template of each pages
* Rename the content.pho to home and fix the hyperlinks at the menu.php and change the setup in template.php as content.php to home.php
* **Setting up the routing**
* Setup the .htaccess for the URLs, then use it in the template.php for routing purposes

.htaccess = 

template.php = 

* Fix the routing for other pages such as the users, categories, products and clients and sales



* Use the home.php as the template for each pages to fill up the side slide. Customize each page as per titles and required features
* **Setup the controllers**
  + Create the controllers for each page (users.controllers.php)
  + Setup the controllers in index.php
* **Setup the models**
  + Create the models for each page (users.models.php)
  + Setup the models in index.php

1. **Setup the 404 page**

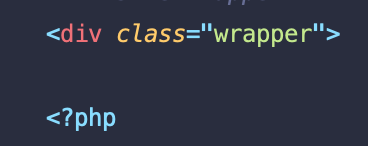
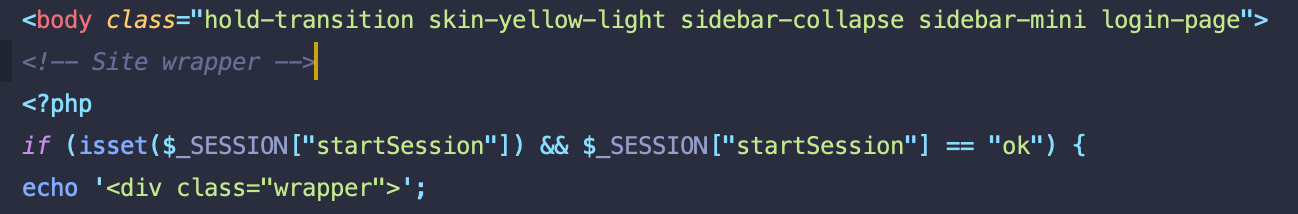
* Create an else stamen in the routing of the template.php



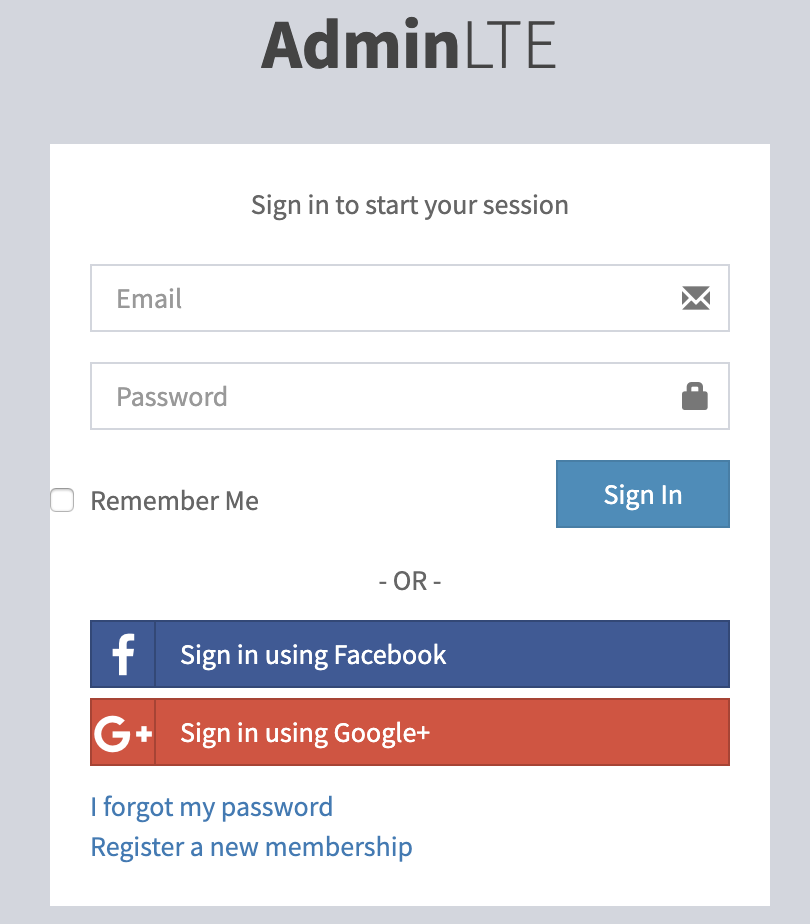
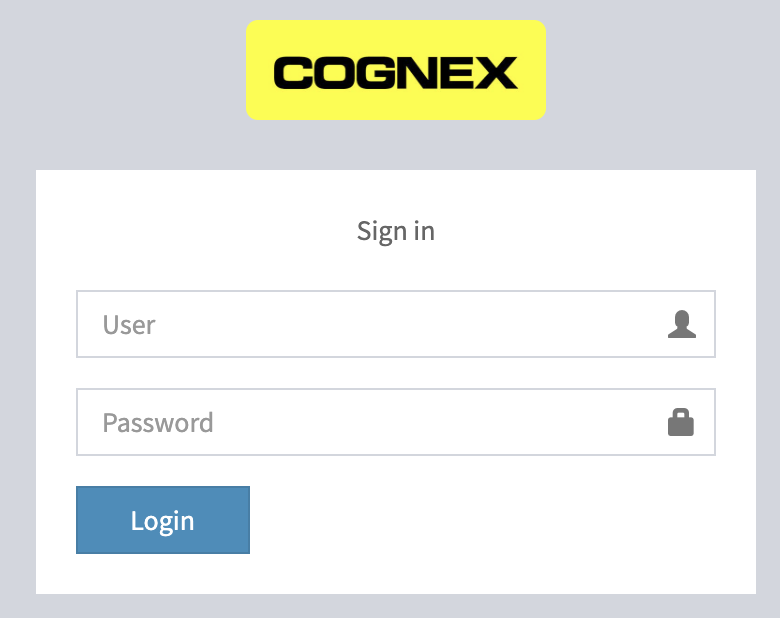
* Create the 404.php inside the modules folder, use any of the page and customize it as a 404 page. Remove unnecessary parts of the page.
* Setup the favicon in template.php

1. **Login Page**

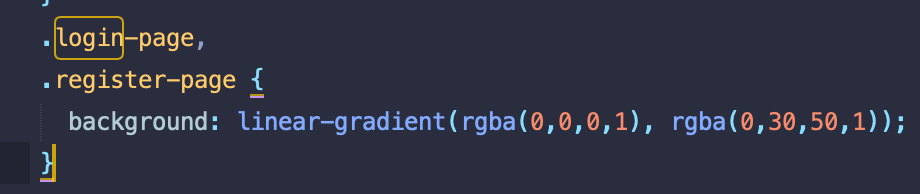
* Refer to login.html
* Configure the template.php to setup an if statement that the login page will be active if a user is not login, the class=”wrapper” will be active once a user login. Use the echo to set up the <div class=”wrapper”>

 >>>  >> 

* Create the login-page “login.php” with reference to login.html, customize if necessary
  + Use form method=POST
  + Change the email to user
  + Delete the col-xs-8, this will remove the “Remember Me”
  + Remove the facebook and google, forgot my password and register a new user links

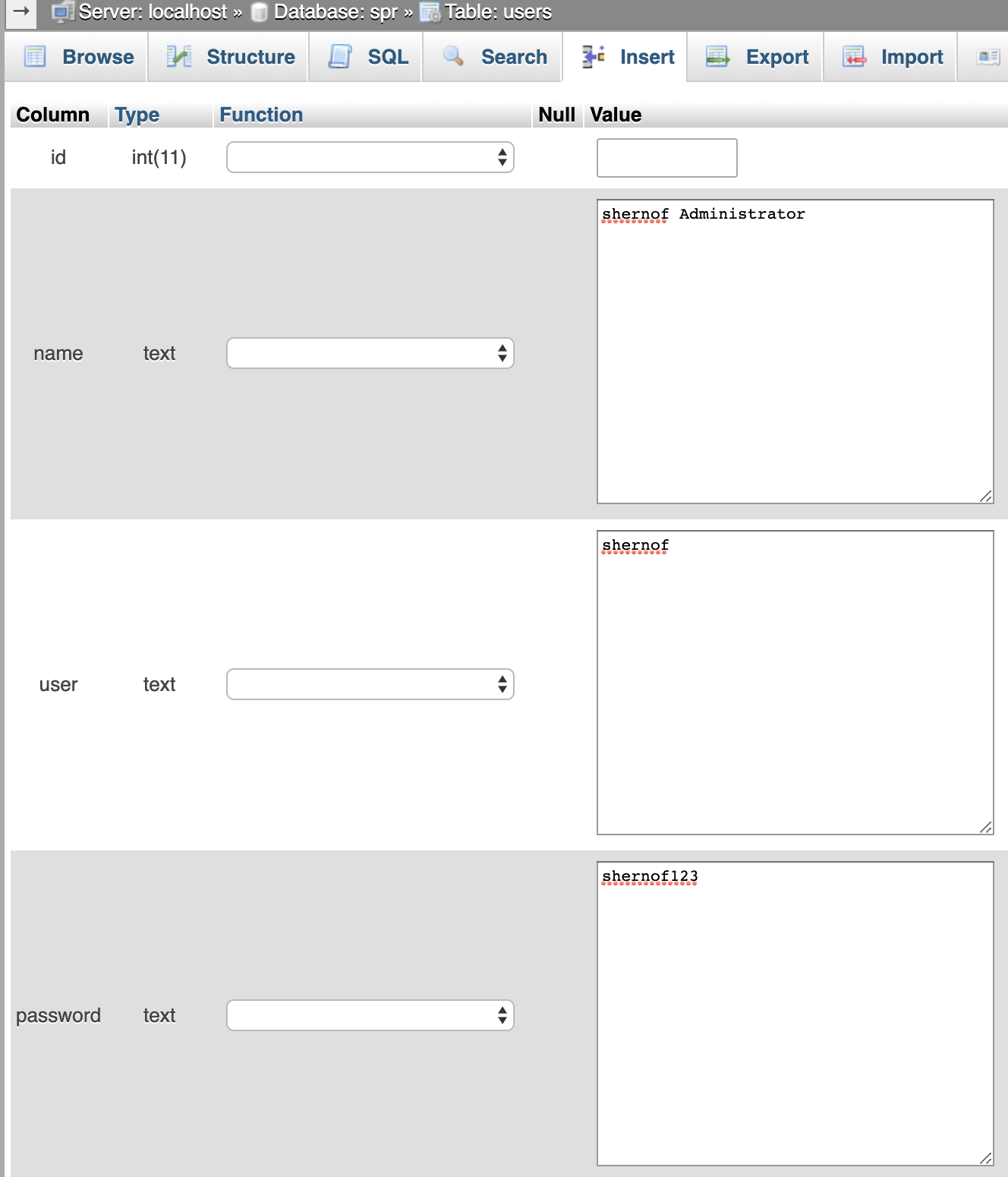
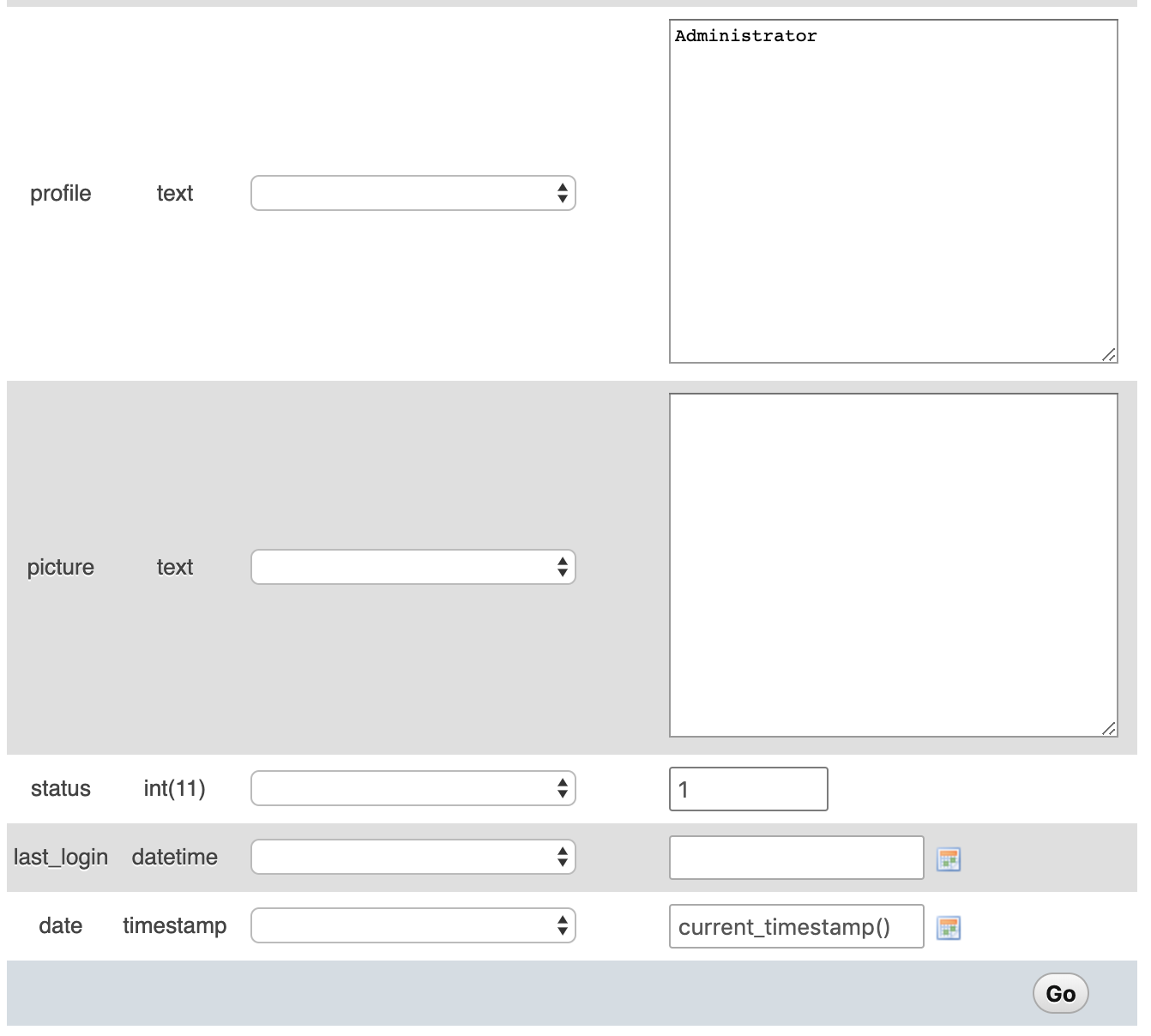
 >>> 

* + Can change the background in AdminLTE.css, can use color or background image



1. **Start Session Part 1**

* Create the database named “spr”
* Create tables for user, products, customers etc.
* Do for user management db first with 9 columns namely:
* id, name, nickname, password, photos, status, save last login of the user and date of creation of the user and profile roles.
  + profile roles:
    - Administrator – can do everything, complete power
    - Special – can add, categorize products, administer stocks, can view users but can’t access to create users or print bills. Can delete and edit products
    - Seller – can access only customers and sales. Can create customer profile but cannot delete it.
    - All deletion can be done by administrator.
* Create the 9 table in the spr database, then save
* Create the admin user by clicking the “Insert” tab

* Configure the login method before the closing tag of form at login.php

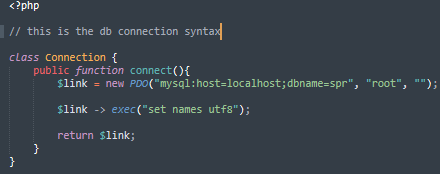


* Configure the user.controller.php to define the class ControllerUsers and ctrUserLogin mentioned in login.php

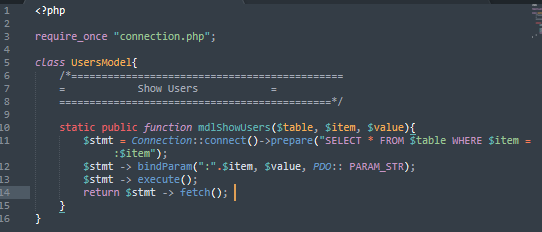


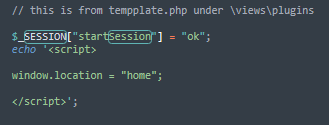
1. **Start session part 2**

* Create the connection.php inside modules folder. This will define the db connection method.

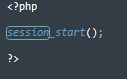


* Configure the user.model.php to define the class UsersModel mentioned in the users.controller.php

 >>>

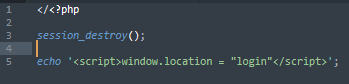
 >>> this will execute the session and will log on a valid user to “home” if the username and password is valid Call it in template. Php afterwards

* At templste.php, start thse session:



1. **Creating the log out function**

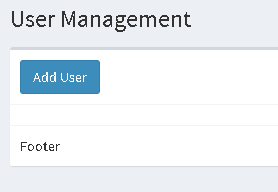
* Logout was called in header.php
* Create the route for the logout page in the template.php
* Create the logout.php under modules folder.
* Destroy the session in this logout.php by calling the method session\_destroy()



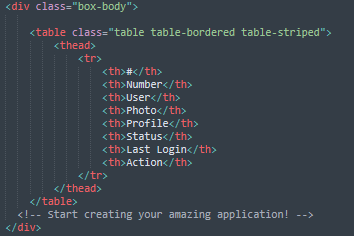
**A complete mvc architecture was done at this point**

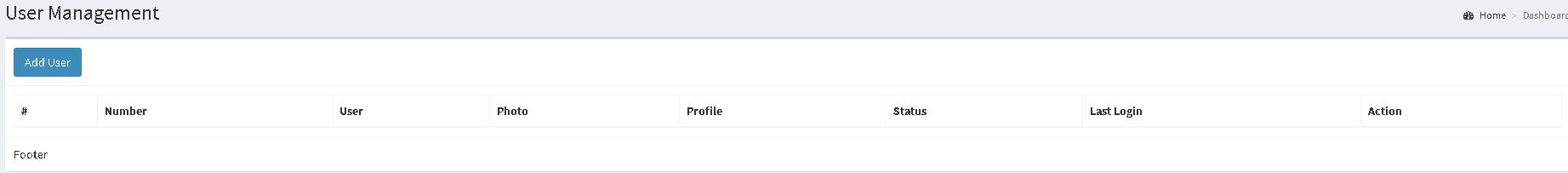
1. **Building User Module – Part 1**

* At users.php and other pages, set all the # to “home”. Customize the users.php and remove un-necessary features.
* Create the “Add User” button

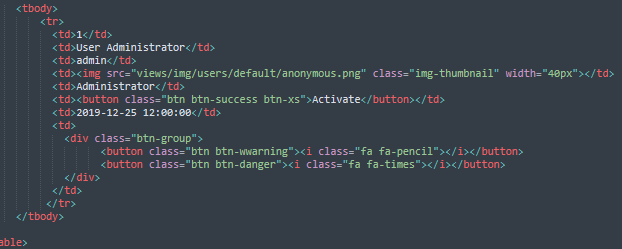
 >>> 

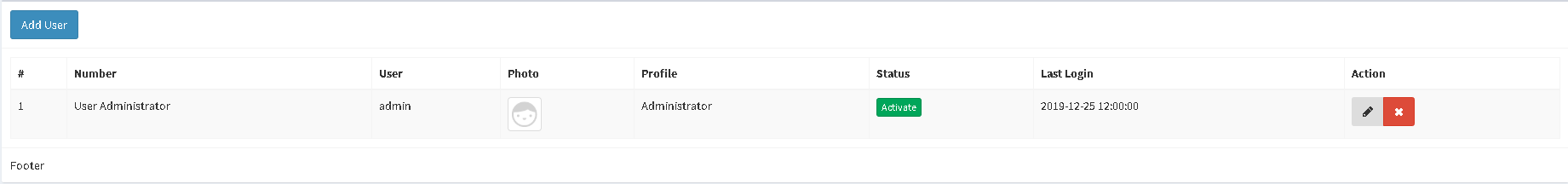
* Create the table for user information





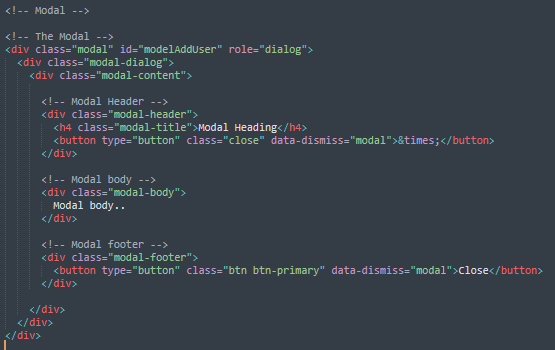
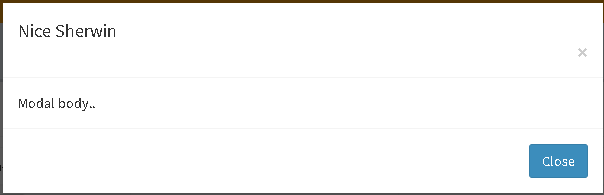
* Create the body of the table





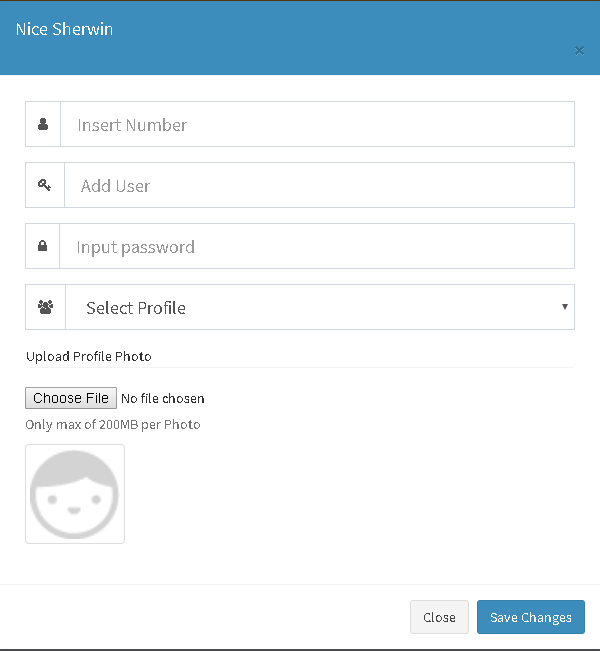
Now lay out is finished

* Create the modal box, can copy the sample at w3-bootstrap

1. **Building user module – Part 2**

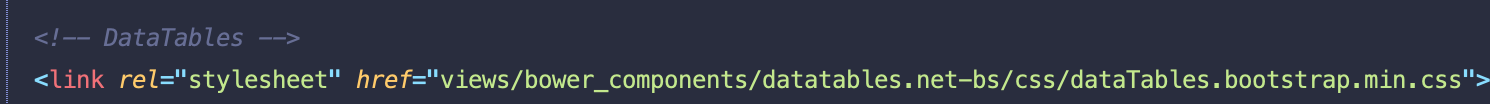
* Continue working on the lay out of the modal. Add the input number, username, password, profile, picture upload

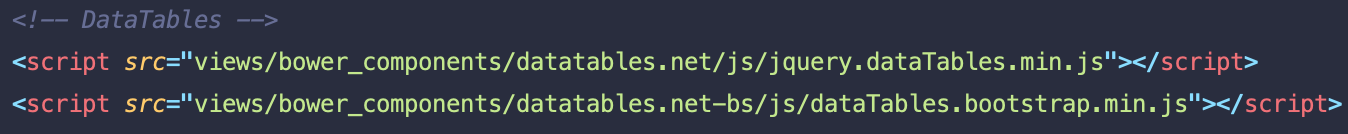


1. **Data table plug in** – this will make the table appear better

<https://datatables.net/>

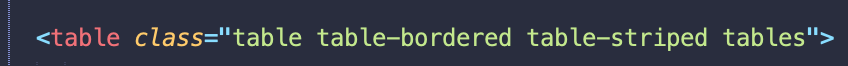
* Copy the <link> and <script> for the data table to template.php.



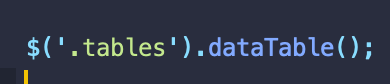


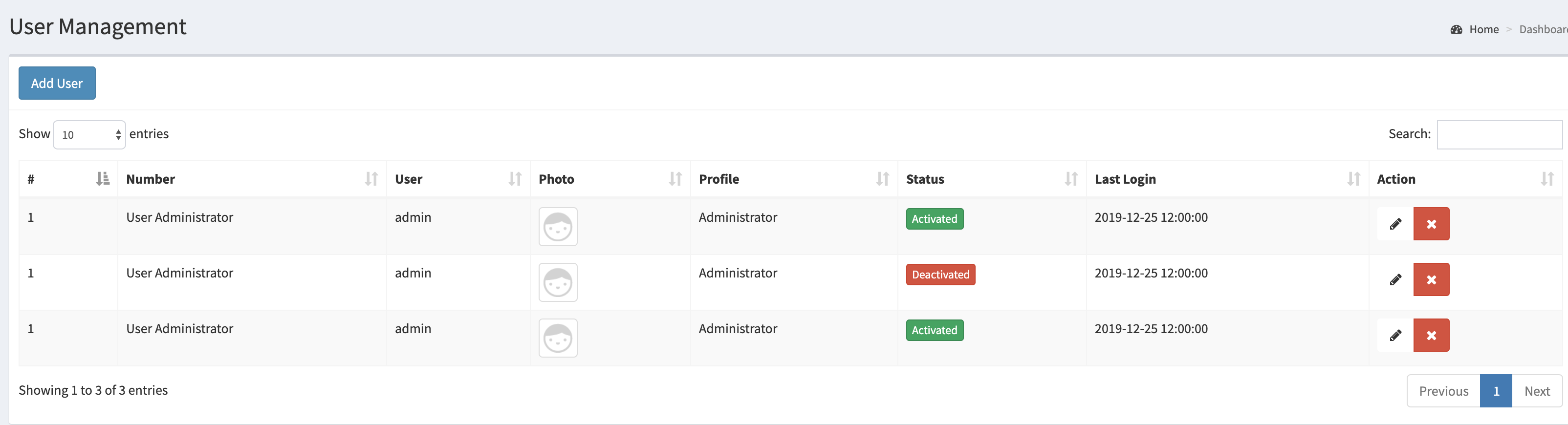
* Activate the code in users.php





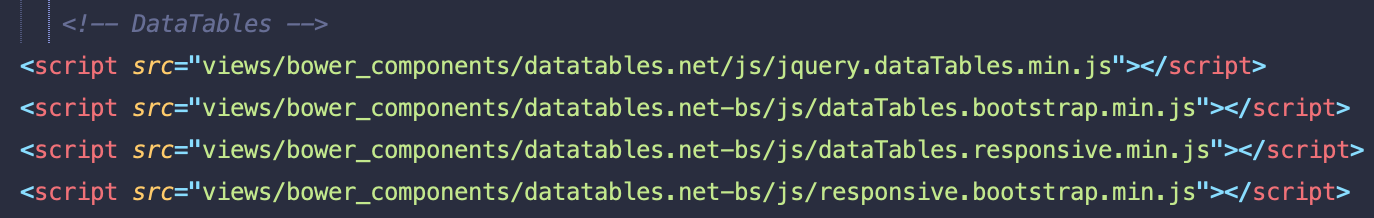
* Call and activate it in template.js



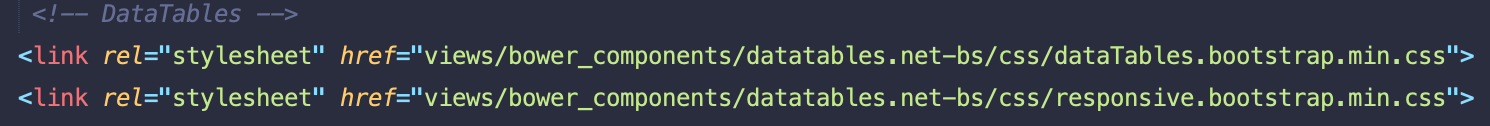


* Make the app responsive in mobile device by adding responsive css and js:

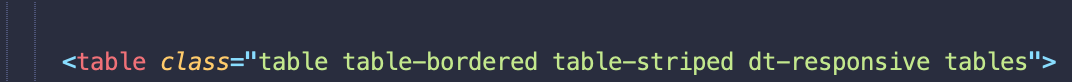
Last 2 scripts



Last CSS

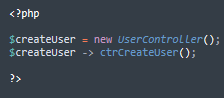


Add the dt-responsive in the table class



1. Adding users

* Create the users.html.php inside the modules folder
* Add some styling in users.php, in # field and apply it as well in users.html.php
* At “modal-form, in users.php open a php tag and create a php object before closing the modal form that will enable to save the user and add this info to database.



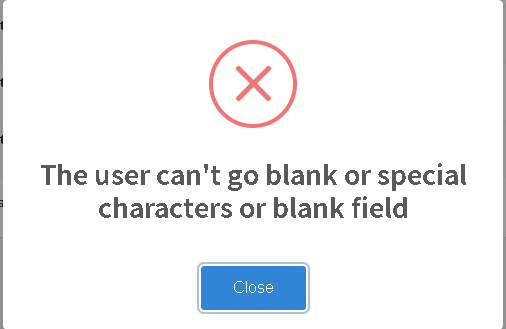
* Add “static” before “public” in every template. This will ensure that the function will work in any server.
* Close the connection to db for each successful transaction at users.model so that it will not execute in continuous way



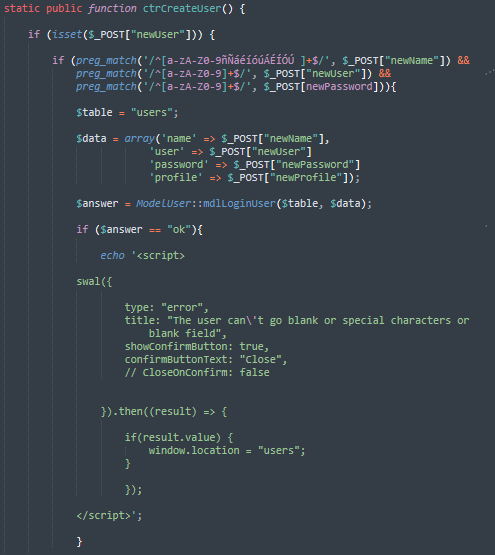
-create the user registration in users.controller.php, this is in response with “newUser’ in users.php and the method called before closing the modal form.



* This feature can include the “SweetAlert” function (<https://sweetalert2.github.io/>). Copy the cdn script and create the “sweetalert2.all.min.js” inside the plugins\sweetalert2, then call it in template.php

* Continue working in saving data to db in users.controller.php



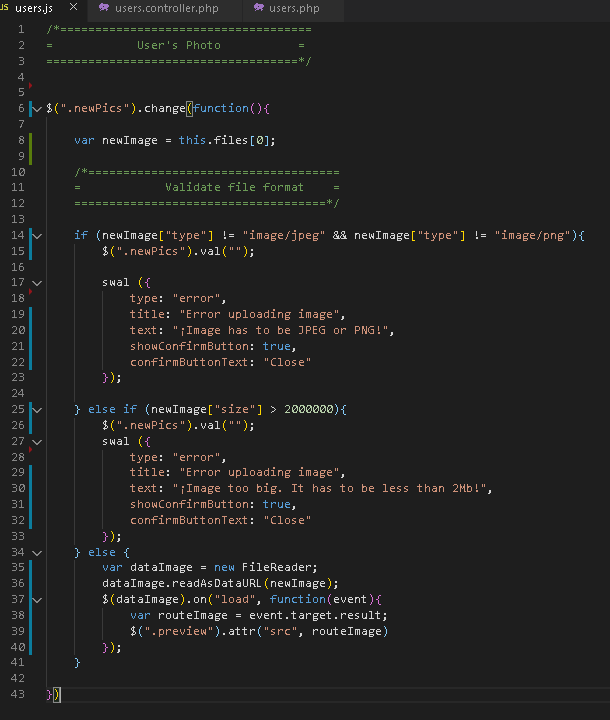
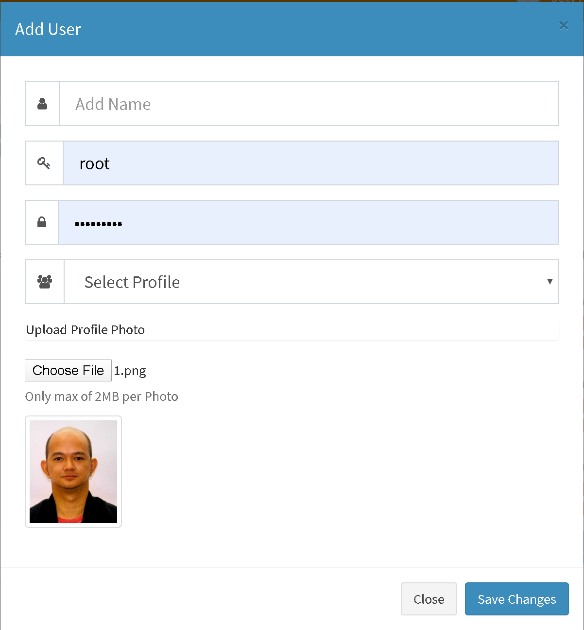
* Then work on the user registration in users.model.php

1. uploading user image

* Change the id to class for “newPhoto”, this will help to edit the user information.



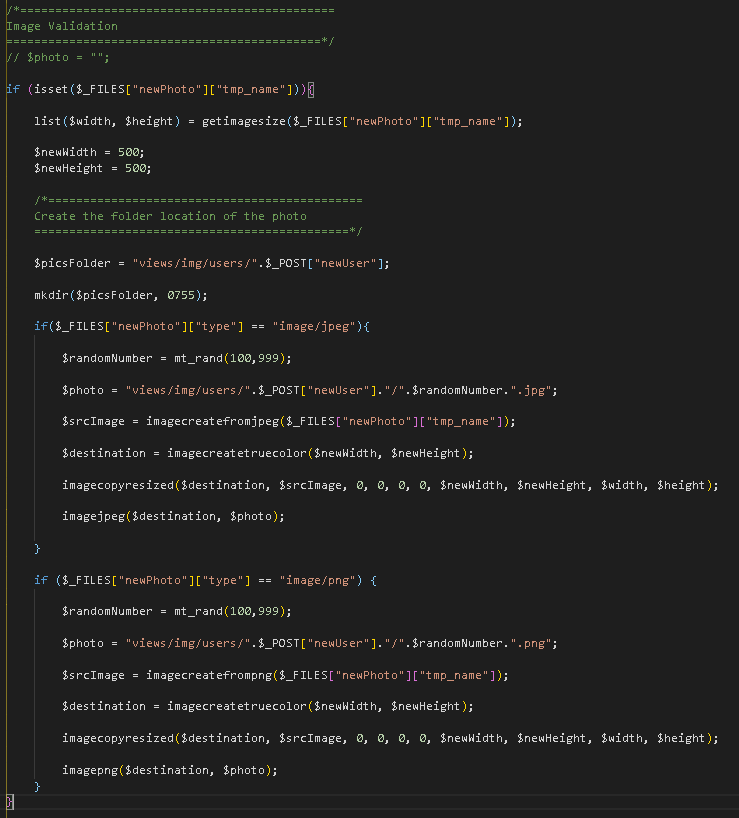
* create a new js file named users.js, this will contain the java script for the user. Work on below photo upload code. The following js code will embed the photo in the photo box as below.

1. Adding User Photo – Part 2

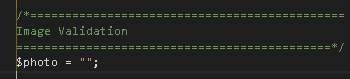
Saving photo to in a folder

* 1. In users.controller.php

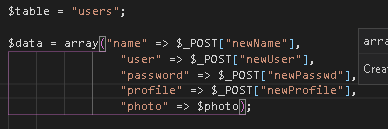


Saving photo to in database

* 1. Make the $photo = “”, to be validated



* 1. Add the $photo in $data array of the users.controller.php

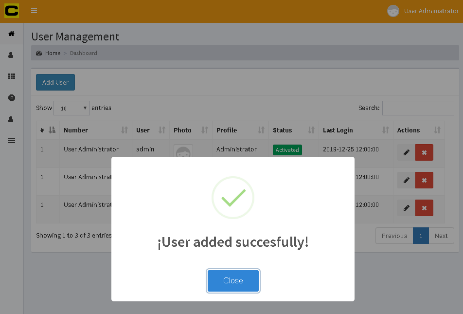


* 1. Execute adding to db by configuring it in users.model





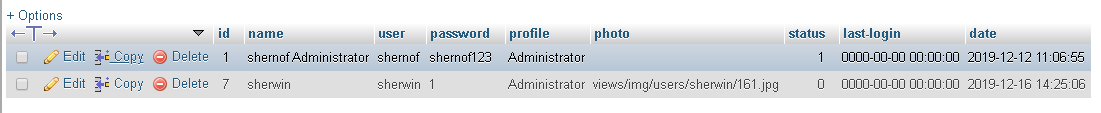
* 1. It now adds photo successfully with sweet alert



Save photo data in directory



Save it in db



1. Encrypting the password - utilize the crypt(a, b) of php in users.controller.php

a = parameter, b = salt

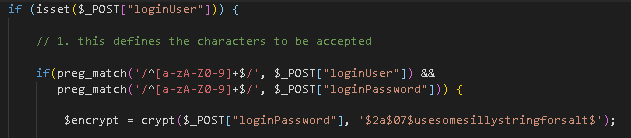
use the crypt\_blowfish, safest crypt



It will encrypt the password of the new user added



Adjust the comparison of password during login.





The user can now login with encrypted password

1. Create sessions – this will able to allow the system to know who are the users that logged in. this will

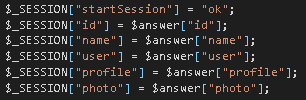
improve some of the layout of the page like display the profile picture and indicate the username in the

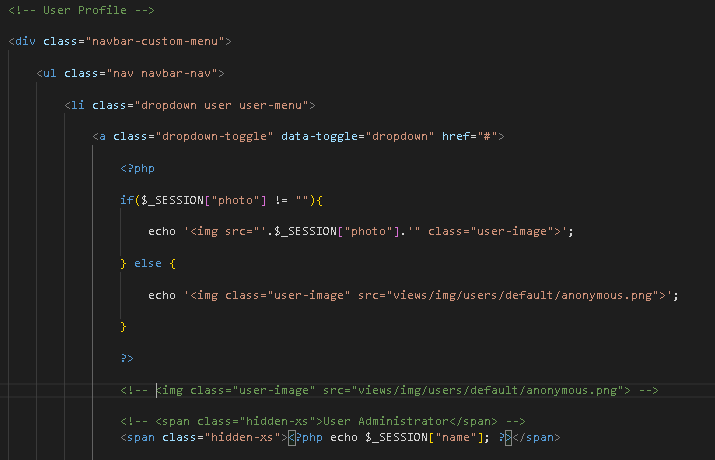
header. 

25. Create sessions – this will able to allow the system to know who are the users that logged in. this will

improve some of the layout of the page like display the profile picture and indicate the username in the

header. 

1. in users.cntroller.php, create the $SESSION variables of each db columns
2. 
3. At header.php, customise the profile picture and name. the if statement will make the profile picture of the user as default if the user didn’t upload a picture.

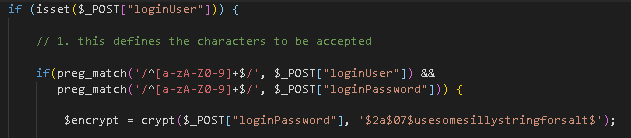


This will display the name and username of the user



24. Encrypting the password - utilize the crypt(a, b) of php in users.controller.php

* a = parameter, b = salt
* use the crypt\_blowfish, safest crypt
* 
* It will encrypt the password of the new user added
* 
* Adjust the comparison of password during login.



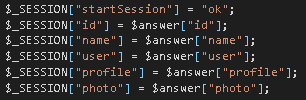


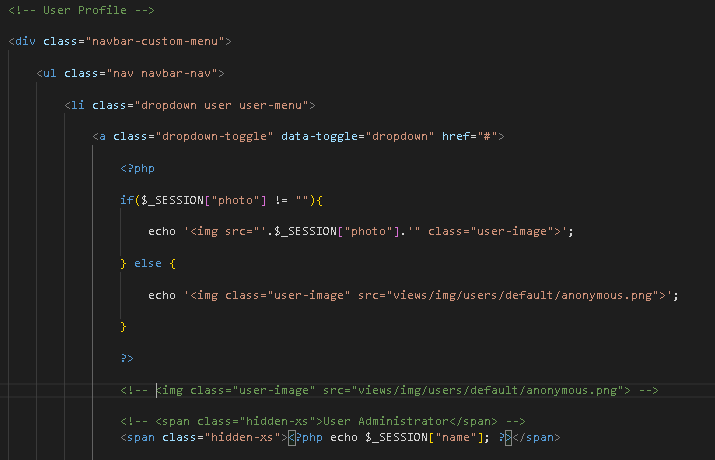
* The user can now login with encrypted password

25. Create sessions – this will able to allow the system to know who are the users that logged in. this will

improve some of the layout of the page like display the profile picture and indicate the username in the

header. 

* in users.cntroller.php, create the $SESSION variables of each db columns
* 
* At header.php, customise the profile picture and name. the if statement will make the profile picture of the user as default if the user didn’t upload a picture.



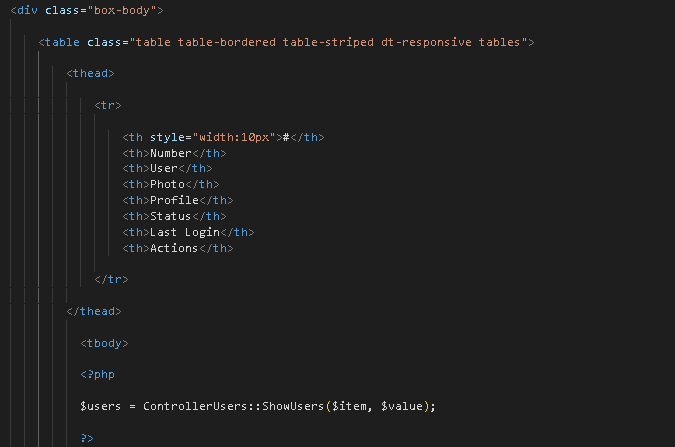
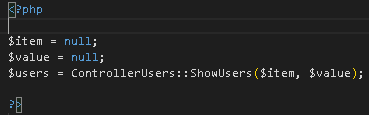
This will display the name and username of the user



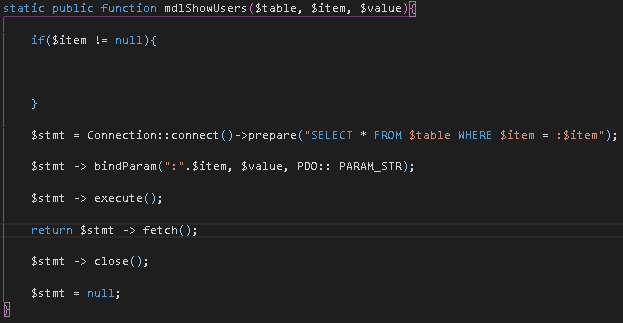
26. Selecting users and display it in the USERs main page. Replace the static display with the dynamic data

of the users from database.

* Open up a php tag to create the object to define how to display the users from the database in users.php. can remove the static users rows

 >  >>

* The $item and $value will be accepted in the method ctrShowUsers in the users.controller.php and be evaluated at users.model.php
* Create the function in the users.controller.php, the method mdlShowUsers will be connectedneed to be connected to ModelUsers.



* Create a foreach for the array data to display it properly to the webpage in users.php. echo the template format of the users table and make the static data be dynamic. Create a condition for the photo field just in case the user did not upload photo

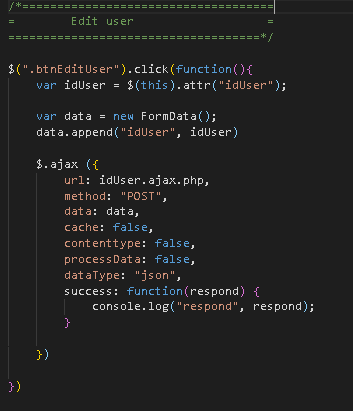


27. Editing Users – give power to edit the accounts by the Administrator profile when the pencil icon is press

* Assign a data-toggle function to fa-pencil in users.php



* Create the editUser, can utilize the existing modal for addUser. Change all the “new” to “edit” in each name property
* Change the “placeholder” to value, this will be the display view when we retrieve the data from db
* Setup ajax so that the database can be access via javascript add the btnEditUser in the class of fa-pencil and assign an id as idUser=”’.$value[“id”].’”. Create the user.ajax.php
* In users.js, setup the js to invoke the ajax



* Setup the ajax



* Assign an id of each element in the modal (name, user and profile). This will receive and display the data from db and display it in the modal when edit/pencil icon is click



* Then setup in users.js to retrieve the data from db

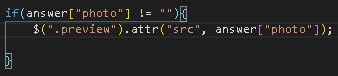
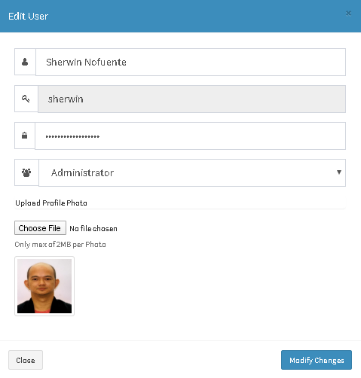


* The modal can now display the data from db



1. Editing User – Part 2

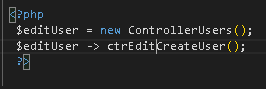
* Adding the picture, in users.js.

* In password field of users.php, create a hidden input with an id of currentPasswd and name of currentPasswd. Do the same with the photo. This is if in case the user did not update the all the fields in the modal.

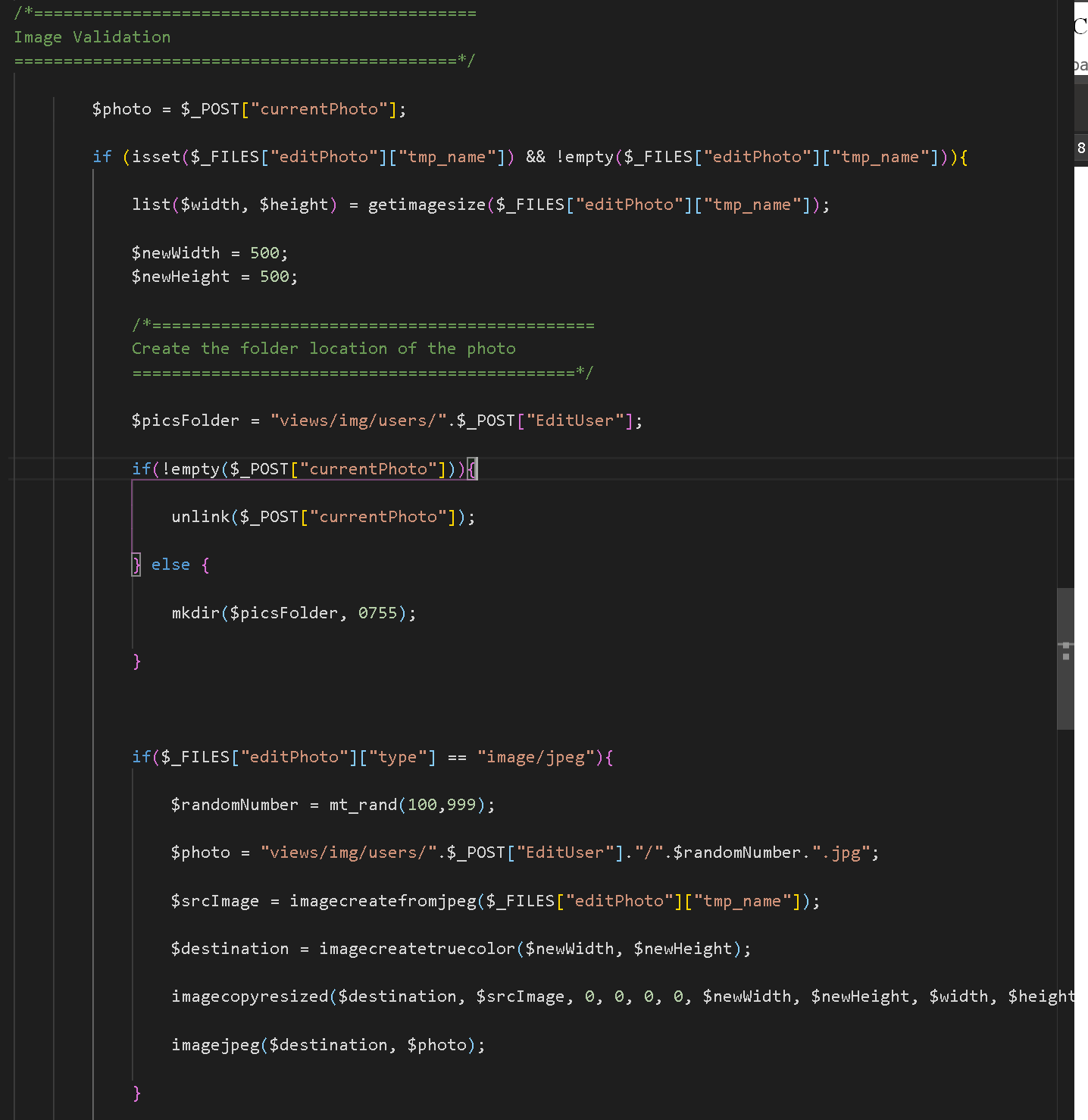
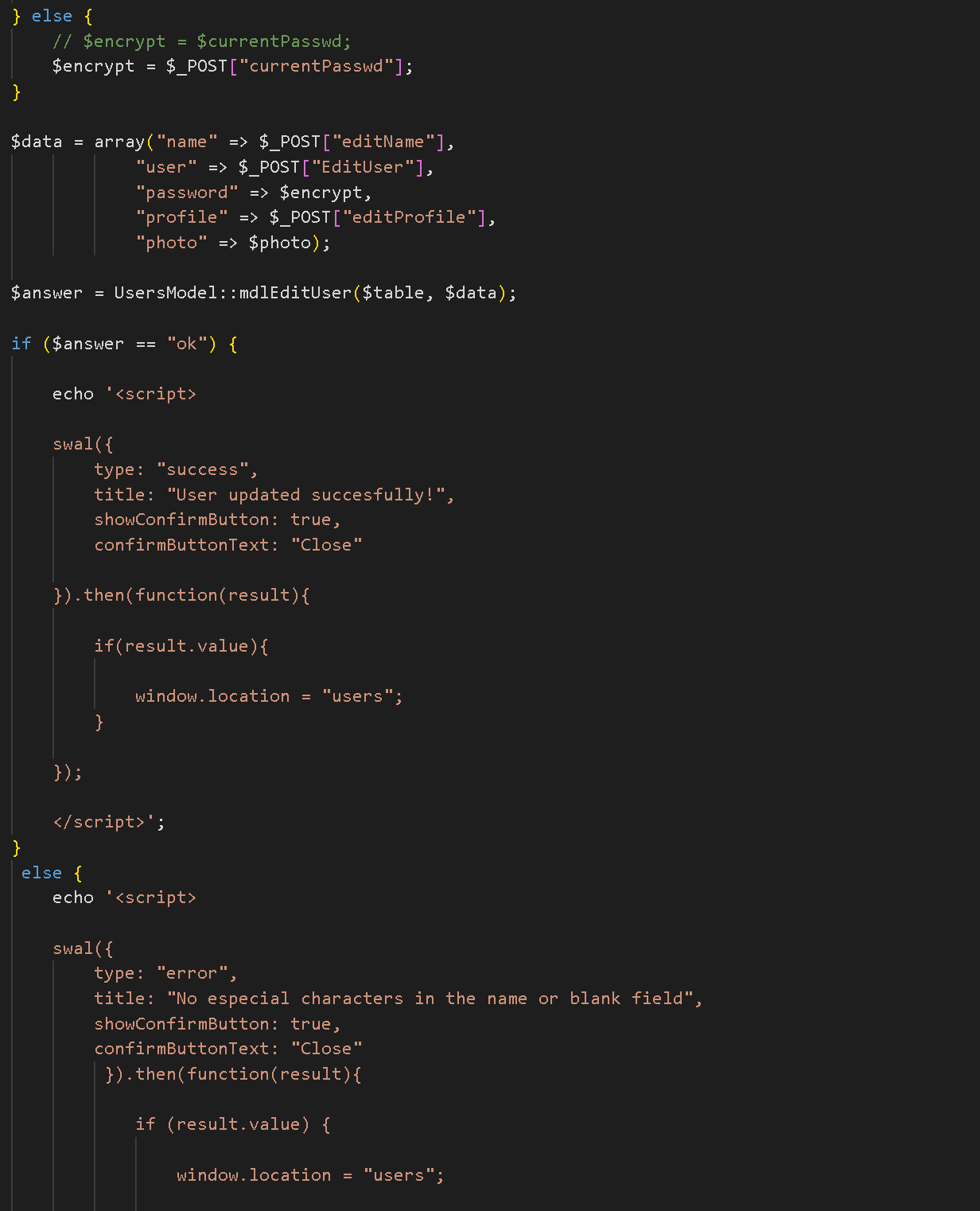


* Activate and rename the $createUser to $editUser, then create the EditUser in users.controller.php

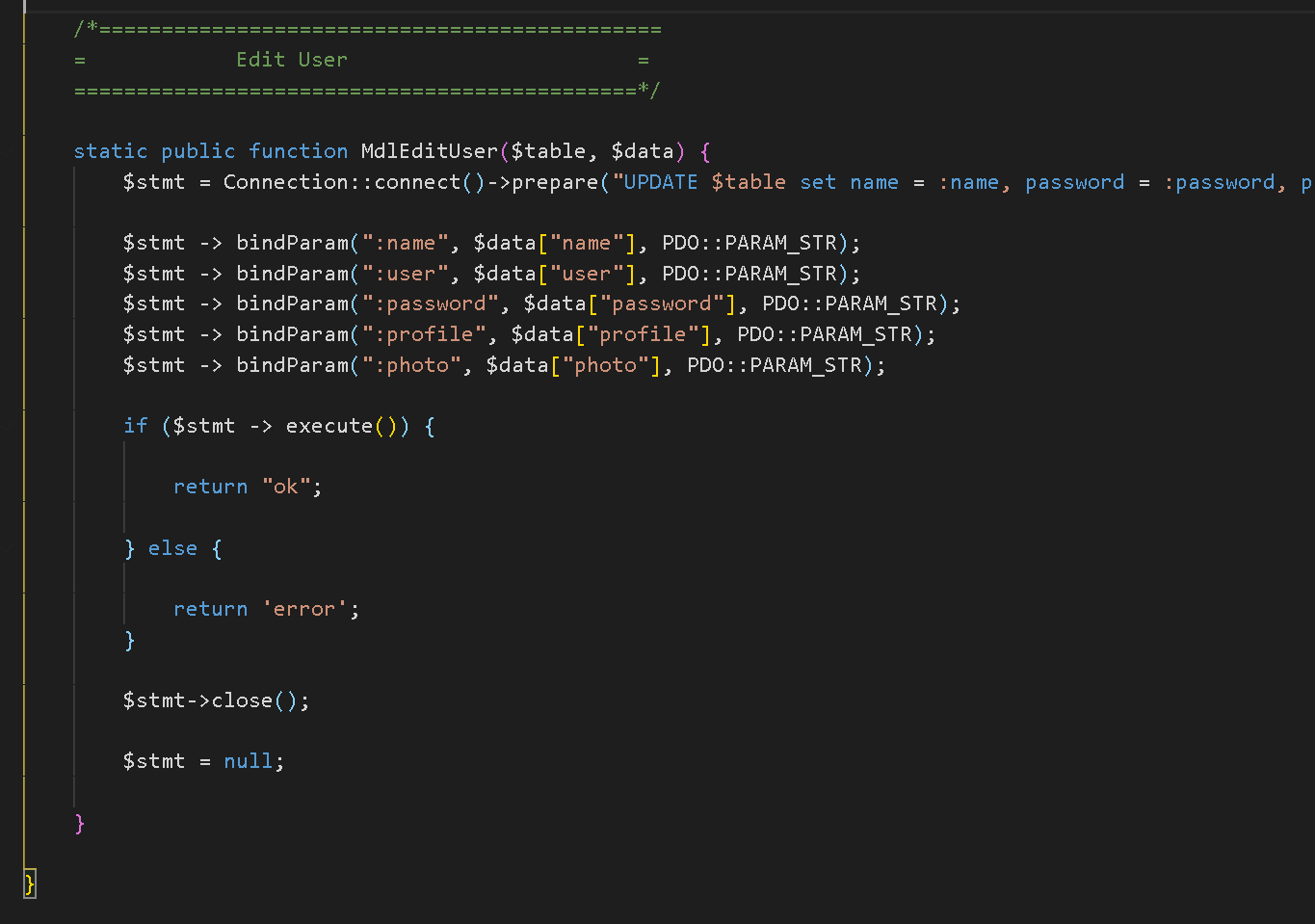


EditUser controller



* Create the user model



Users.js



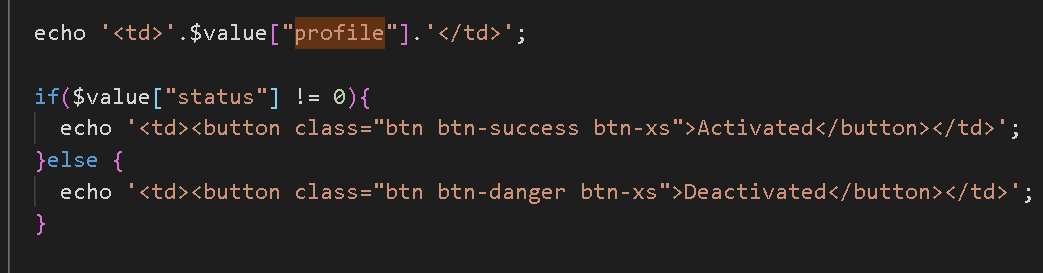
* Create the ajax



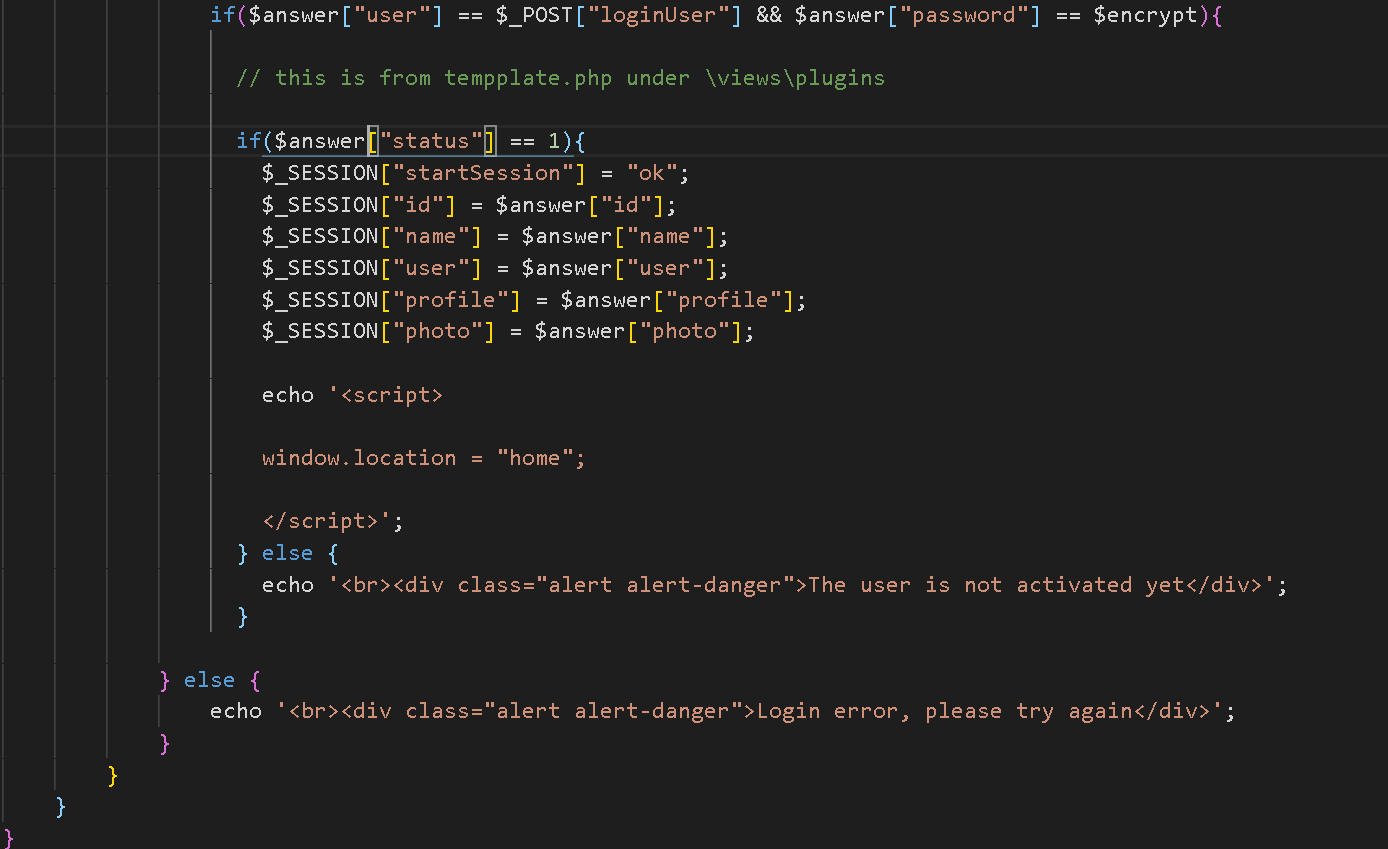
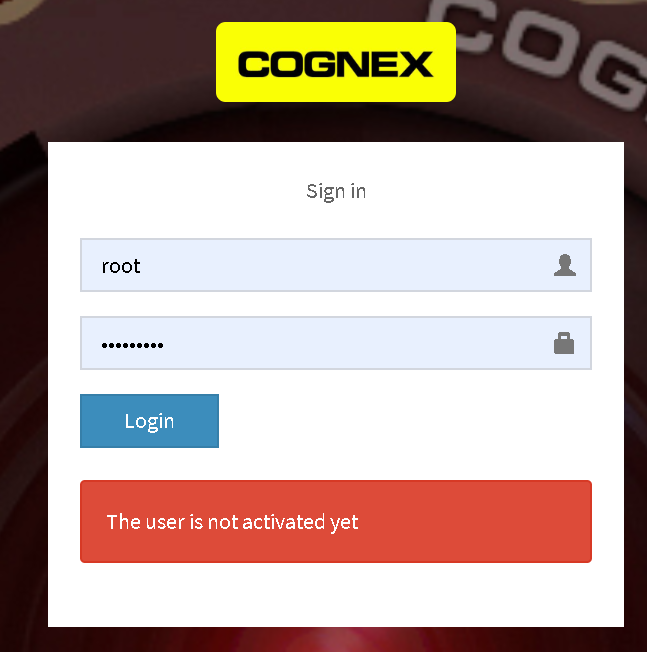
* **Initially did not worked. The “editUser and “EditUSer” created some confusion. Ended up using EditUser in JS**

1. Activate and de-activate User Status

* Defining the status of the user. In users.php, profile section, check the status of the user by creating an if statement

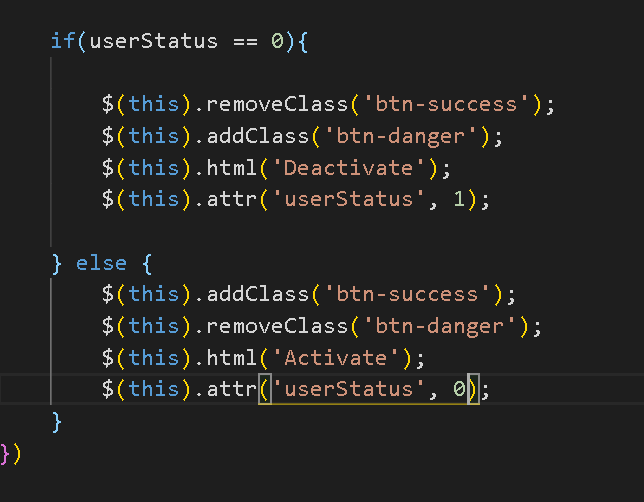


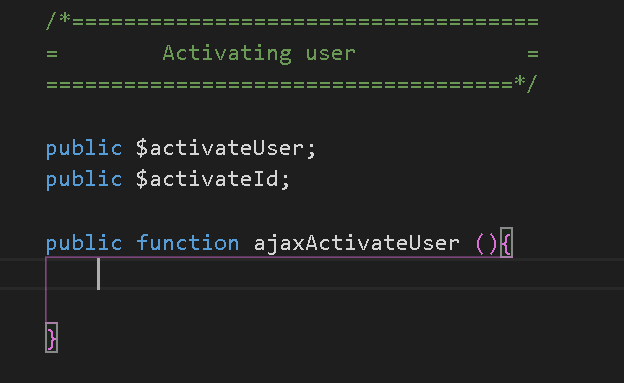
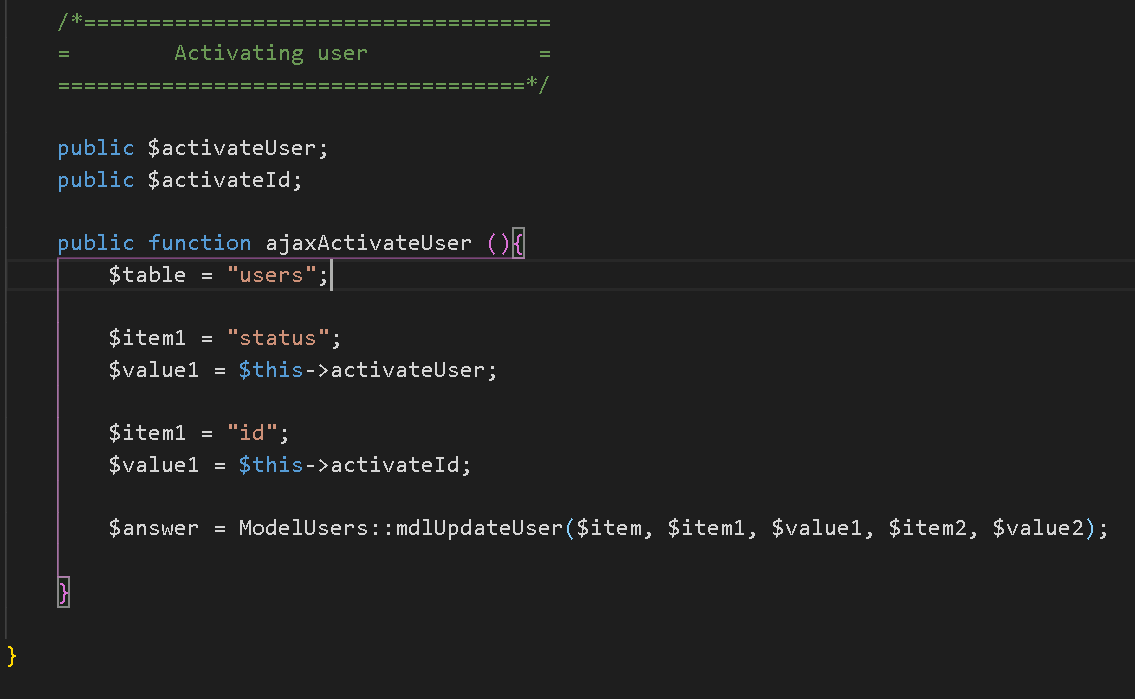
* Make the user unable to access if account is deactivated. In users.controller, at login section, create an if statement to check the user status.

* Activate the account once the “Deactivate” is press
  + Create a btnAcivate in profile section of the usrs.php
  + Create a function to execute when the deactivate button is press in users.js
  + Create a idUser in the buttons with the $value[“id”] and the userStatus to be called in users.js



* + At users.js,
  +  
  + By the help of ajax, update the database. Setup the ajax, create the ajaxActivateUser



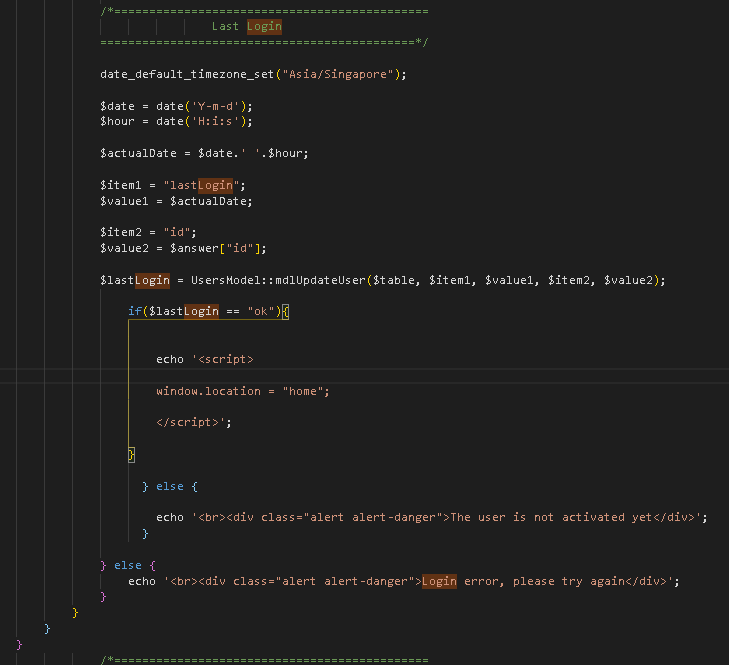
* + Create the mdlUpdateUser in users.model.php



* + Update the users.js to include the changing of status everytime the button is press

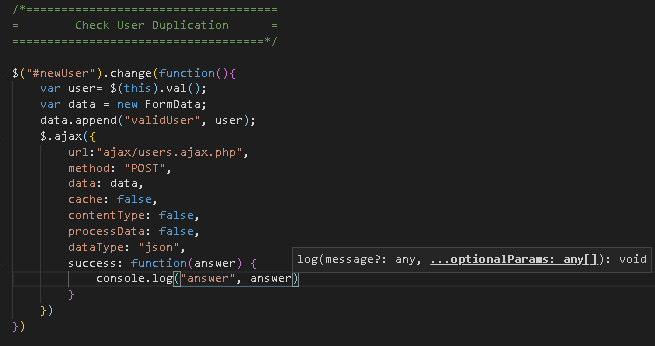
1. Last login –

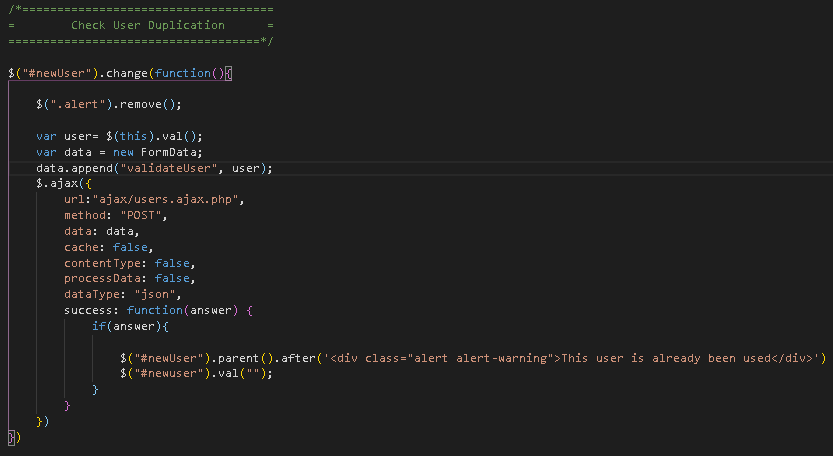
* In users.controller, in login portion before going back to homepage



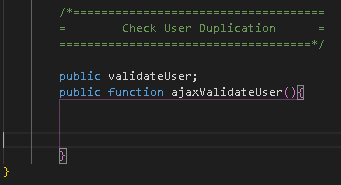
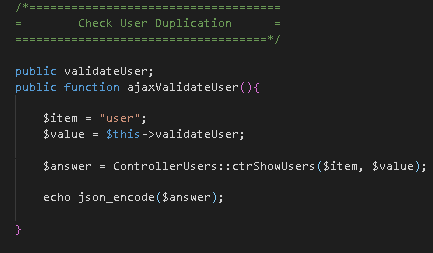
1. Avoiding repeated usernames

* In users.php, newUser section, create an id=newUser.
* In users.js, create the js to call the ajax and ask to check if there’s a user duplication

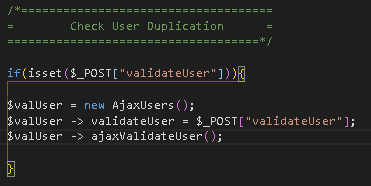
 

 <<< this fix the alert and remove the alert after a new input in the username field

* Use ajax to check in database if there’s any duplication. Declare public the $validateUser and create the method ajaxValidateuser(){} that will connect to the database.

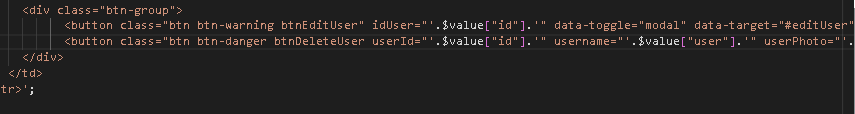
 

* Create the object to validate the user

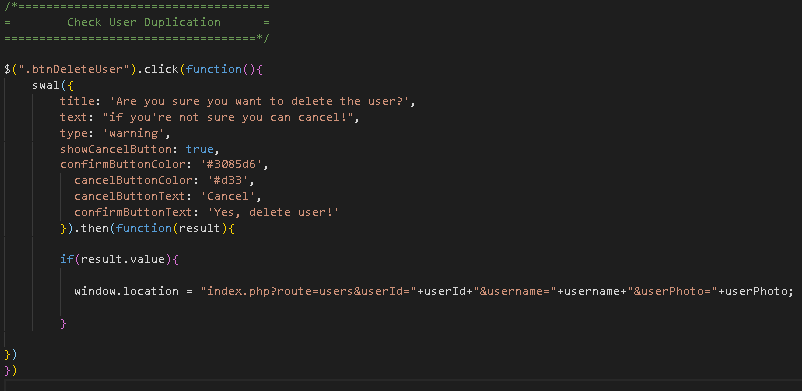
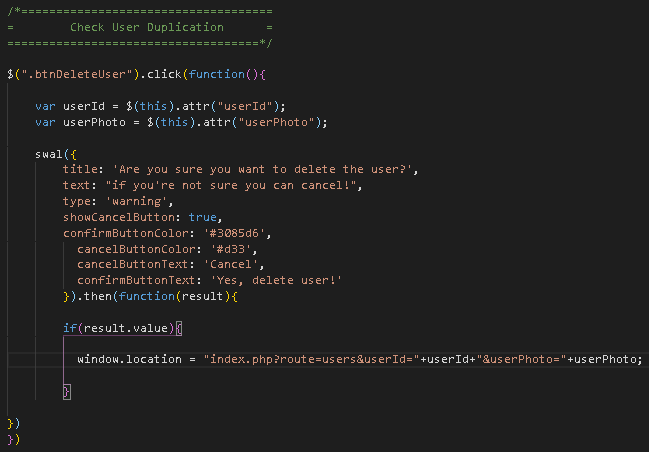


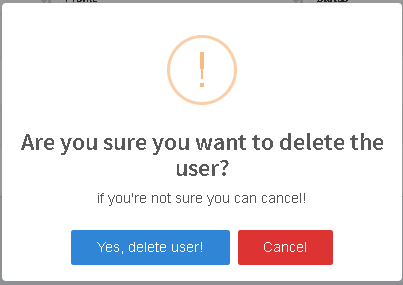
1. Deleting users – activate the x button, it will pop an alert to ask if the user is sure to delete the registered user. N to confirm before finalizing the delete.

* In user.php, fa-times, put a class btnDeleteUser



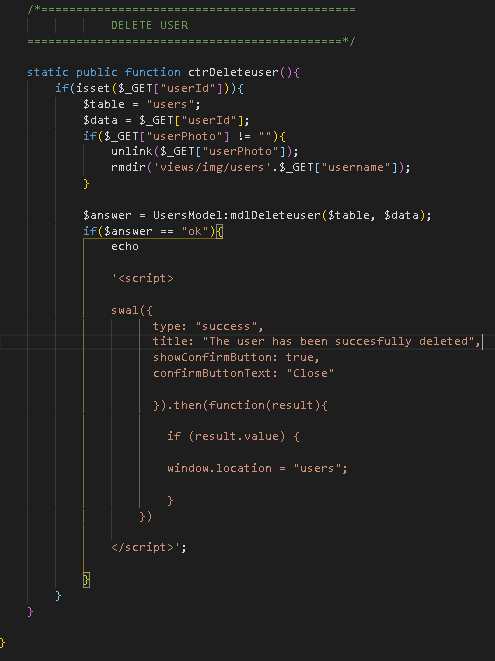
* In users.js, create the javascript to delete the user and invoke the sweet alert to confirm deletion

>> 

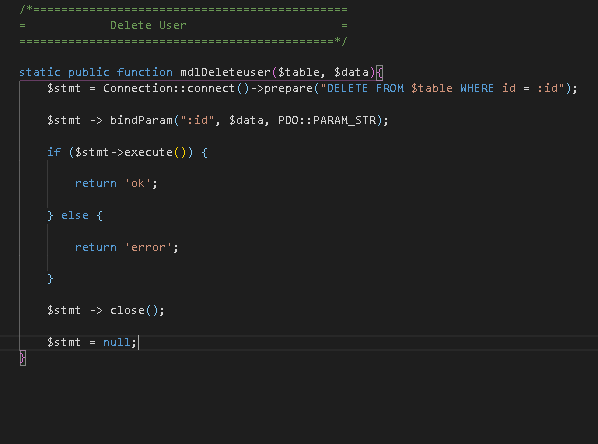


If click yes,

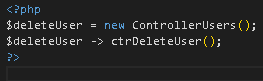


In users.controller, create the ctrDeleteUser function

* Create the mdlDeleteuser in users.model.php



* Enable the DeleteUser in users.php



1. Internet explorer debugging

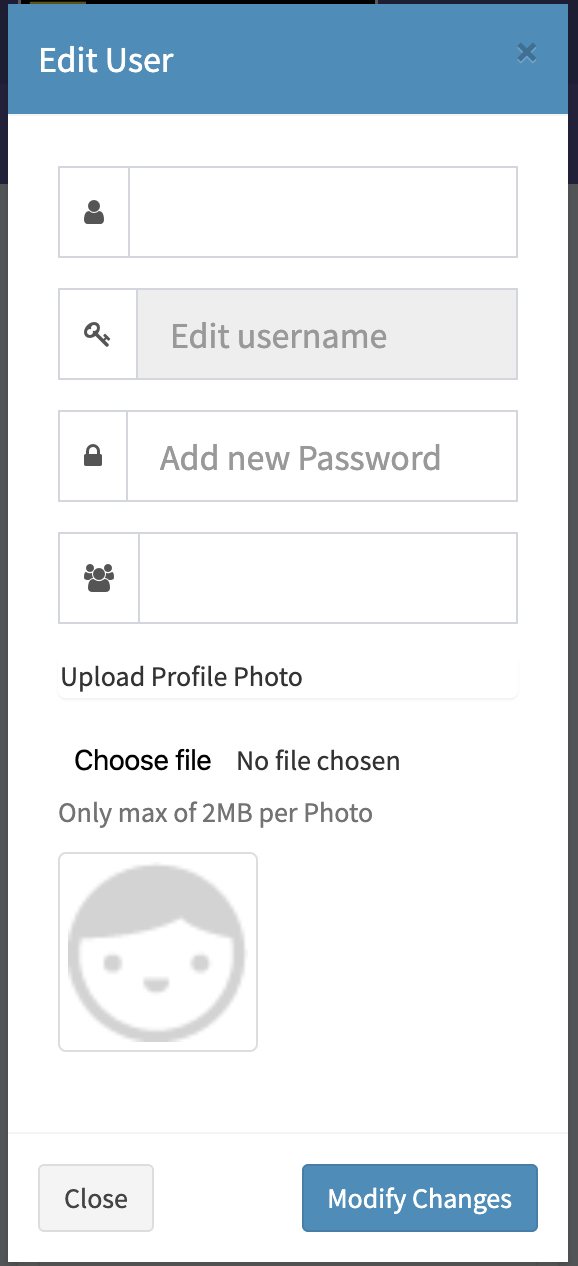
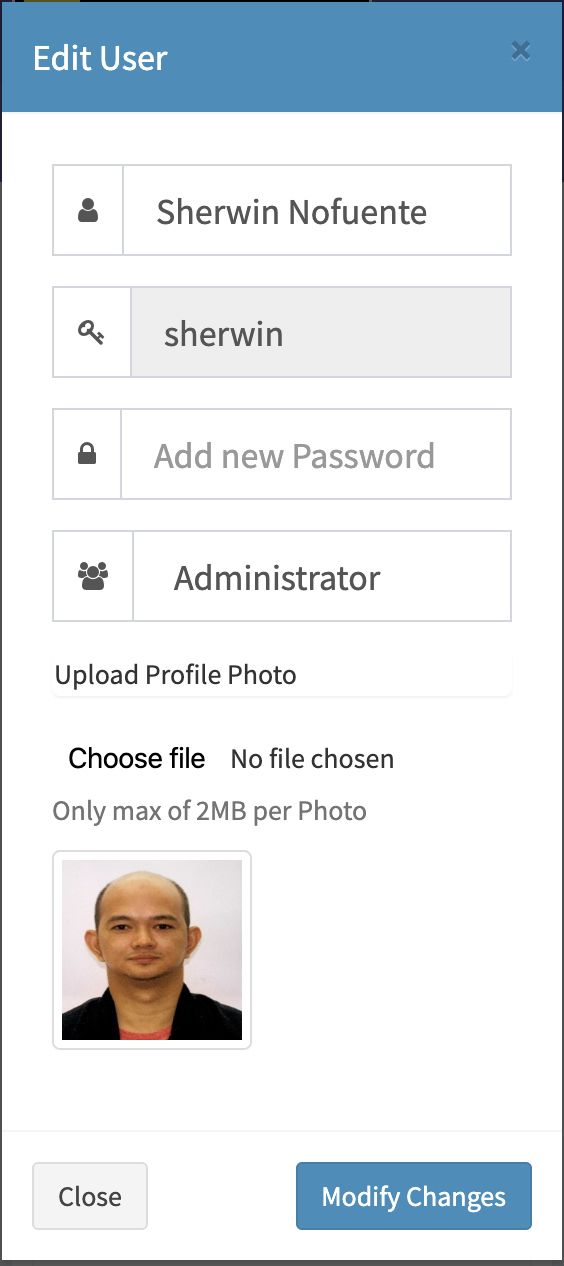
* Use “<script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/core-js/2.4.1/core.js"></script>

” to fix the bug for intertnet explorer plugin.

* Use width 100% for all the pages to properly display the pages.
* Use the normal function instead of arrow function.

1. Solution for click’s action on hidden buttons

* The issue is in the mobile view, the user information is not displayed when edit button is press, it doesn’t pull out the info from db. This is because the html was loaded already prior to the pressing of the + button thus it did not display the html when the fa-pencil is press

*  >>> 
* The correction will make the html be displayed on the document and display the .btnEditUser class to be display when the fa-pencil is click, it will display the info from database
* Do this similarly with btnDeleteUser and btnActivate
* Update the sweetAlert in btnActivate to display it properly in mobile device.

Section 4 – Categories Module

1. Build the CRUD for Categories module

* In categories.php, create the similar table that was created in users.php. we can again use the dataTables plugin, can use the users.php as the base
* Customize the page base from the requirement for categories

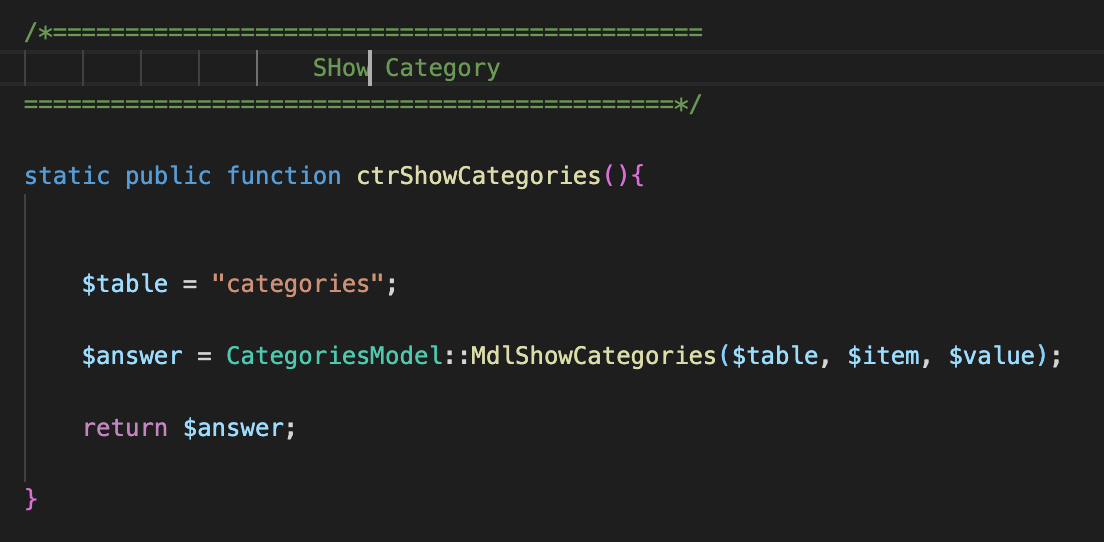
1. Adding Categories (C-RUD)

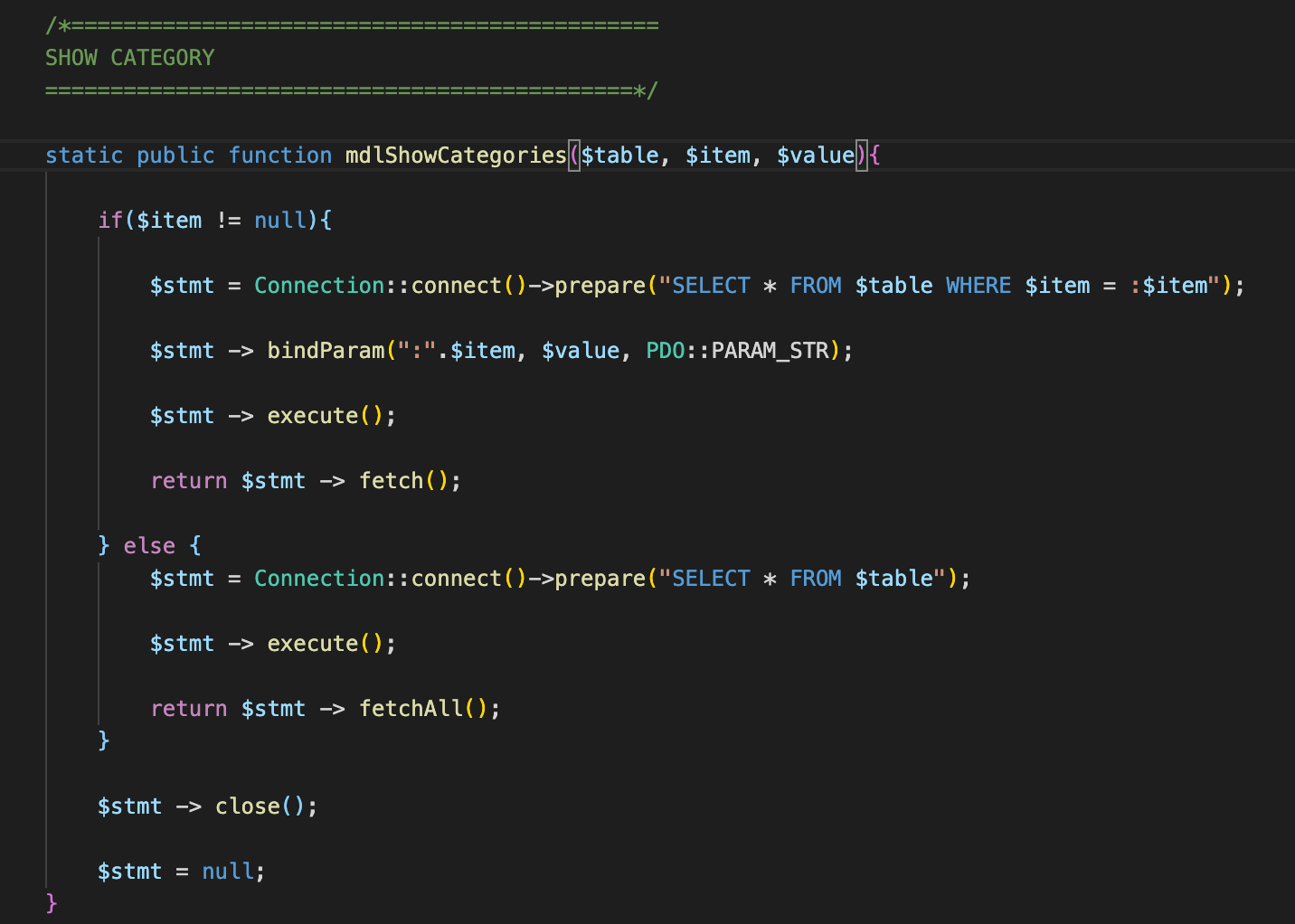
* Declare the controllers in categories. Pphp
* 
* Create the ControllerCategory and ctrCreateCategory in categories.controller.php

* Create the categories table with 3 columns in db
* Create the category.model, create the CategoriesModel and the class mdlAddCategory.
* Detect duplication in JS (optional), create also the ajax to detect duplication

1. Selecting Categories

* In categories.php, create the calling of categories. Open a php tag and create
* 
* Create the method ctrShowCatergories in categories.controller.php.
* 
* Define the model



* Create the forEach to display the array in the page



1. Edit Categories

* Reuse the add category modal and customize base from the reqt of the categories
* Create the JS fir edit category
  + Add a class id of EditCategory, btnEditCategory and idCategory=”’.value[“id”]. In fa-pencil
  + Add the categories.js in the template.php as a js source
  + In categories.js



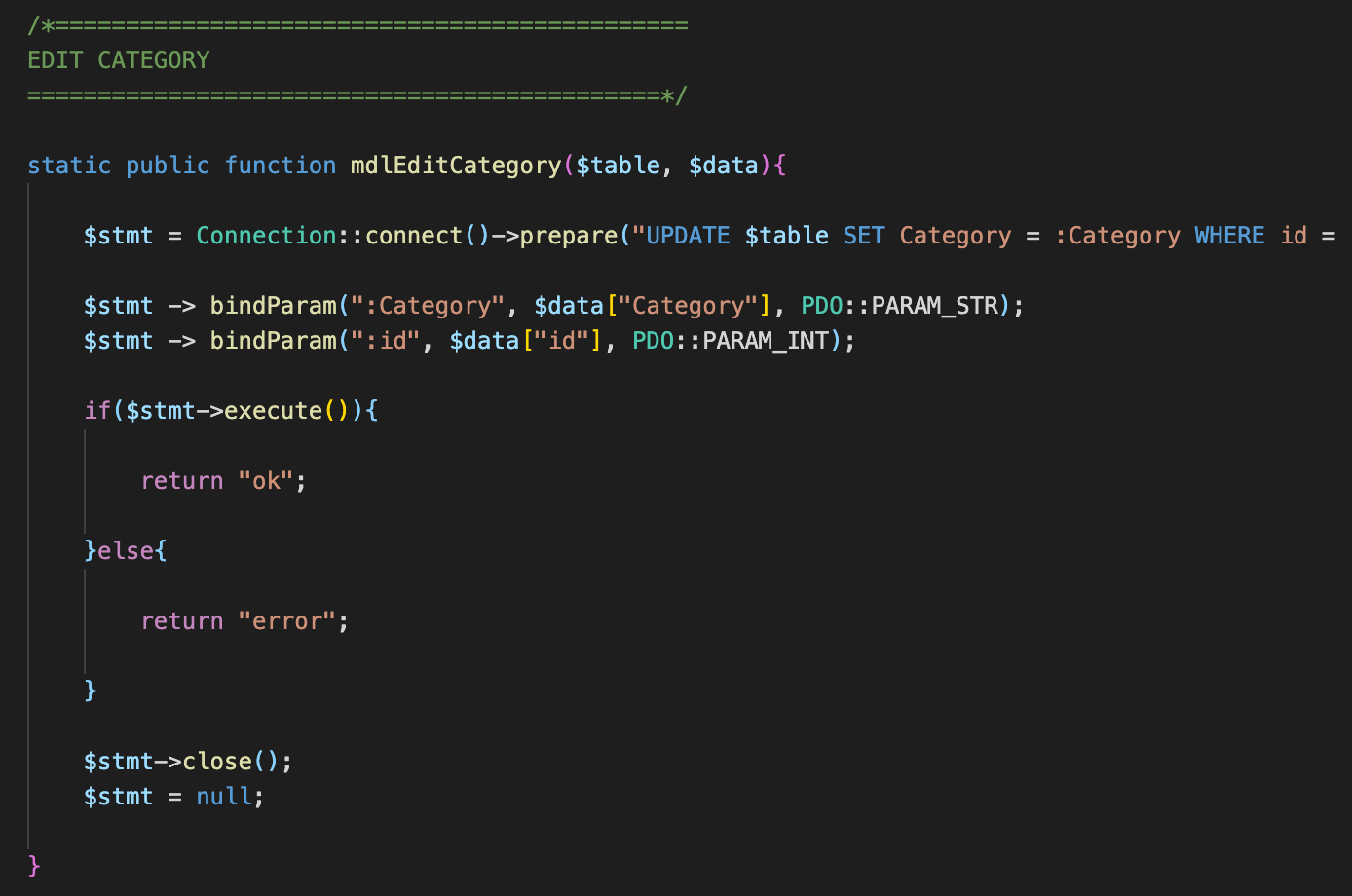
* Create the ajax



* Create the categories.controller.php,



* Create the mode

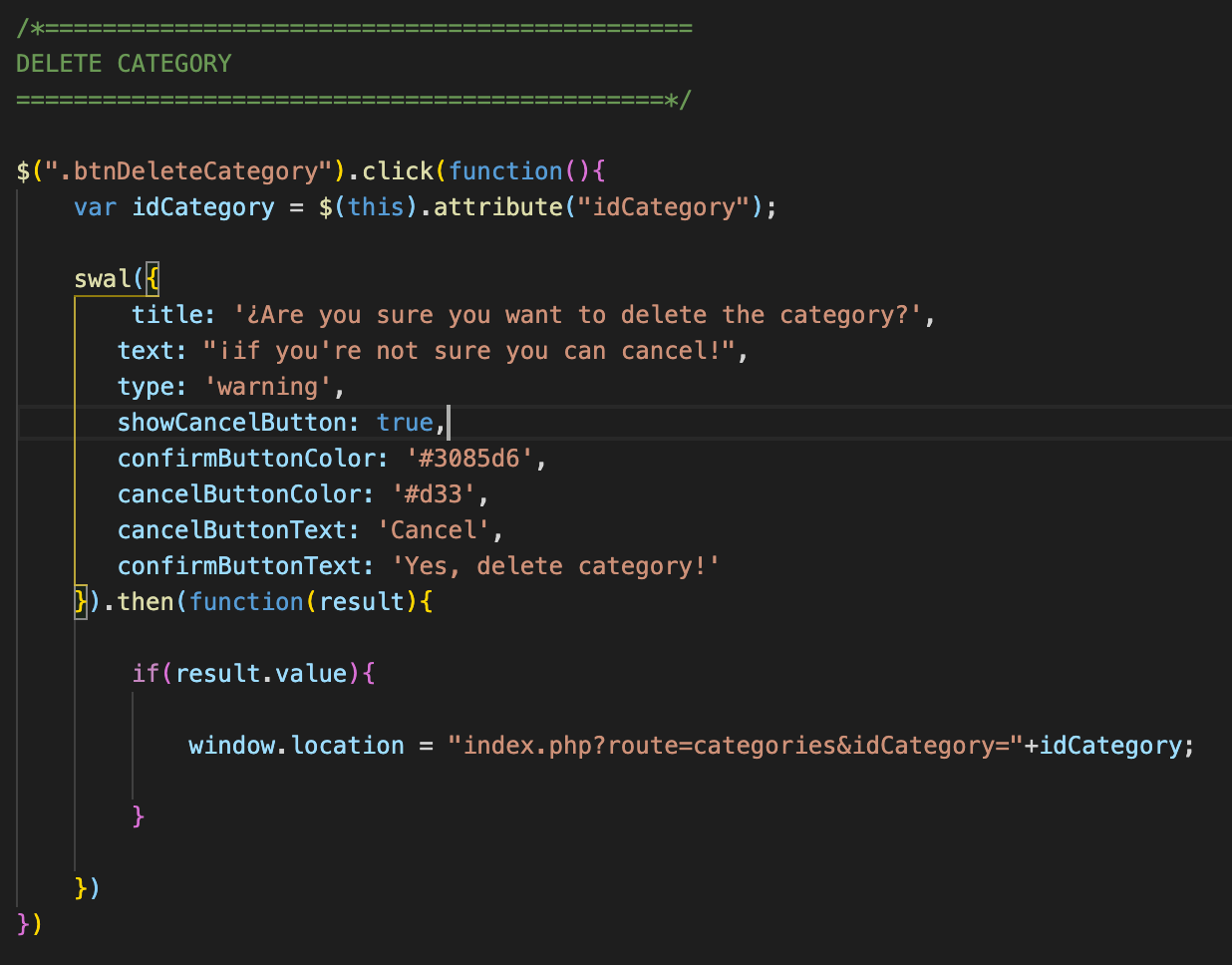


1. Deleting a category

* Assign “btnDeleteCategory” to fa-times.



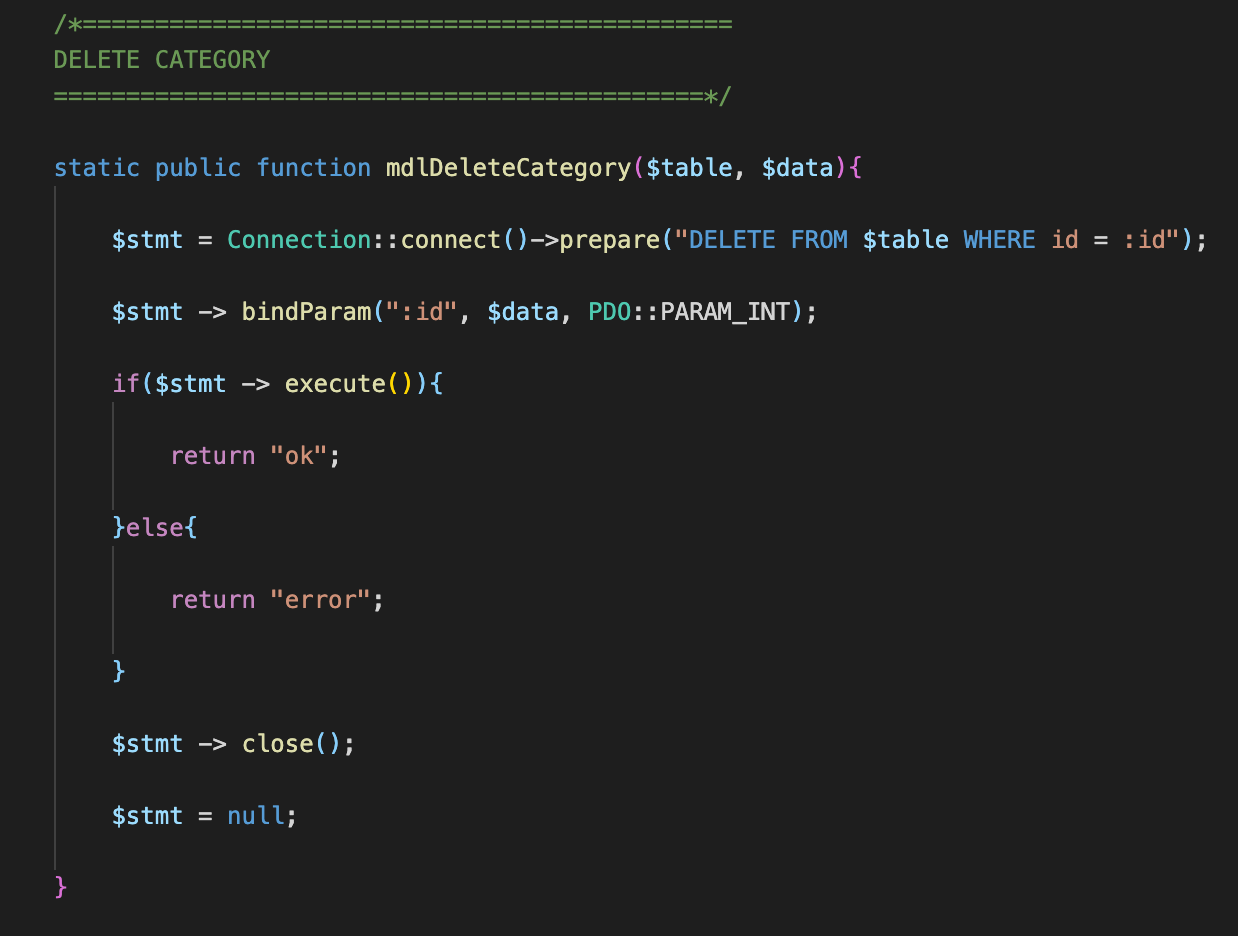
* Create the js in categories.js



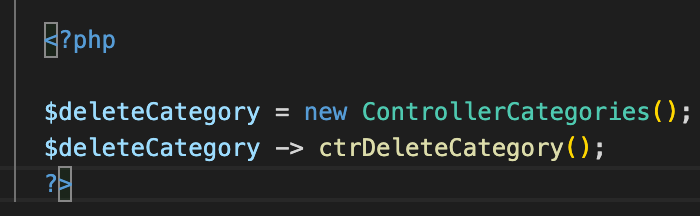
* Create the Delete controller



* Create the delete model



* Add the modules to call in controller.php



1. Building Product Modules

* Create the layout for the Parts module, can use the users.php as the base.