Web development exam project

PROJECT REPORT

24th February 2019

Mirjana Erceg

Copenhagen school of Design & Technology

Professional Bachelor in Web development

Project report

Project participants:

Mirjana Erceg

Supervisor:

Jorge Santiago Donoso Correa

Submission Date:

24th February 2019

Table of Contents

Introduction

The purpose of this report is to describe the structure of the Bank application which we have been working on during our lectures and in project week. I will try to explain the workflow of the whole application.

The application is developed in Visual Studio Code source code editor and is built with JavaScript, jQuery, HTML, CSS and PHP. The hosting and domain services I bought at Namecheap and the application is running on http://www.mimi-micek.website. The application is locally running on Apache XAMPP server.

Firstly, I am going to explain the whole concept of the application and how it works. Thereafter I will explain the core functionality of the application and the code.

Structure of the application

The application consists of several folders and files. The page files with a .php extension that are not in any folders are the pages that are shown in the User Interface. Some of them are *index.php, login.php, profile.php, logout.php* etc.

The folders contain CSS files used for styling of the UI, JS folder contains JS and jQuery files, DATA folder which mimics our database and APIS folder holding all the APIs.

Few parts of the application are structured to be as a Single Page application, since we are not using any frontend frameworks it is made so with jQuery and JS. Single Page approach is presented in the *index.php* file as it is shown in *Figure 1 Index.php.*

```
<?php require_once 'top.php';?>
  <div id="logo" class="page">
    <h1>WELCOME TO ERCEG BANK</h1>
  </div>
  <div id="signup" name="signup" class="page">
    <h1>SIGN UP</h1>
    <form id="frmSignup" method="POST">
      <div class="row" >
        <input name="txtSignupPhone" type="tel" placeholder="phone" maxlength="8" value="" required>
      </div>
      <div>
        <input name="txtSignupName" type="text" placeholder="name" value="Lala" minlength="2" maxlength="20" required>
      </div>
      <div class="row">
        <input name="txtSignupLastName" type="text" placeholder="last name" value="Lalic" minlength="2" maxlength="20" required>
      </div>
      <div class="row">
        <input name="txtSignupEmail" type="email" placeholder="email" value="a@a.com" required>
      </div>
      <div class="row">
        <input name="txtSignupCpr" type="text" placeholder="cpr" value="1234567891" minlength="10" maxlength="10" required>
      </div>
      <div class="row">
        <input name="txtSignupPassword" type="password" placeholder="password" value="1234"  minlength="4" maxlength="20" required>
      </div>
      <div class="row">
        <input name="txtSignupConfirmPassword" type="password" placeholder="confirm password" value="1234"  minlength="4" maxlength="20" required>
      </div>
      <div class="row">
        <button>Signup</button>
      </div>
    </form>
    </div>
  </body>
</html>
<?php require_once 'bottom.php';?>
```

Figure 1 Index.php

With the help of PHP we have created two separate files *top.php* and *bottom.php* which are going to be used throughout the whole application so we have decided this the best way to re-use existing code instead of duplicating it in every page. The *top.php* page contains the header with the navigation links for all pages and the *bottom.php* contains scripts for jQuery and Sweetalert, with jQuery we control the hiding and showing of the pages that is used when clicking on navigation links. In *index.php* I have included those two pages with *require_once* statement and *logo* and *signup* page with jQuery which presents SPA.

Signup

In the *signup* page I have implemented frontend form validation like *minlength="2" maxlength="20" required* that keeps the string length between 2 and 20 and ensures the input is not left empty. Once the form is submitted the request goes through AJAX for form validation as shown in *Figure 2 Signup.js.*

```
$('#frmSignup').submit(function(){

  $.ajax({
    method: "POST",
    url: "apis/api-signup",
    data: $('#frmSignup').serialize(),
    cache: false,
    dataType: "JSON"
  }).
  done(function(jData){
    if(jData.status == 1){
      swal({
        title:"CONGRATS", text:"You have signed up", icon: "success",
      });
      setTimeout(function(){location.href = 'login'}, 3000);
    }else{
      swal({
        title:"SYSTEM UPDATE", text:"System is under maintenance" + " " + jData.code, icon: "warning",
      });
    }
  }).
  fail(function(){
    console.log('error')
  })
  return false
})
```

Figure 2 Signup.js

The *frmSignup* is retrieved and submitted through AJAX that has a defined *"POST"* method since we are saving it to our data file, the url to which the request is sent, in this case the *api-signup.php* file, the form containing data that is serialized to a query string sent in a request by AJAX to the server, setting the *cache: false* appends a timestamped query parameter to the request URL which then makes sure that the script is downloaded each time as it is requested and *datatype: "JSON"* evaluates the response as JSON and returns a JS object. For the form's elements to be included in the serializes string they must have the *name=""* attribute like in *Figure 1 Index.php.* If the request passed successfully a notification is showed to the user and redirection to the *login* page is activated and if the request fails and *error* will be show in the console.

In the *api-signup.php* first we set to not display any errors with the *ini_set* statement. Thereafter the passed inputs from the form are retrieved from AJAX request and backend validated as in the examples shown below. If something is wrong the 0 response is sent together with the line of code where the issue is located.

```
$sEmail = $_POST['txtSignupEmail'] ?? '';
if(empty($sEmail)){ sendResponse(0, __LINE__); }
if(!filter_var($sEmail, FILTER_VALIDATE_EMAIL)){
  sendResponse(0, __LINE__);}
$sEmail = $_POST['txtSignupEmail'] ?? '';
if(empty($sEmail)){ sendResponse(0, __LINE__); }
if(!filter_var($sEmail, FILTER_VALIDATE_EMAIL)){
  sendResponse(0, __LINE__);
}
```

We open our database file, decode it from a JSON encoded string to a PHP variable and check if the file is corrupted and echo the object to test for objects we get.

```
$sData = file_get_contents('../data/clients.json');
$jData = json_decode($sData);
if($jData == null){ sendResponse(0, __LINE__); }
echo json_encode($jData);
```

With *$jInnerData* we point to data of each user in the database file and we create new objects with *new stdClass()* and save all the values we have received from the form in them and create some additional empty objects and group of objects that will be used later.

```
$jInnerData = $jData->data;

$jClient = new stdClass();
$jClient->name = $sName;
$jClient->lastName = $sLastName;
$jClient->email = $sEmail;
$jClient->cpr = $sCpr;
$jClient->password = password_hash($sPassword, PASSWORD_DEFAULT);
$jClient->active = 1;
$jClient->iLoginAttemptsLeft = 3;
$jClient->iLastLoginAttempt = 0;

$jClient->accounts = new stdClass();
$jAccounts = $jClient->accounts;
$jAccounts->checkingAccount = new stdClass();
$checkingAccount = $jAccounts->checkingAccount;
$checkingAccount->accountName = "checking";
$checkingAccount->accountBalance = 0;
$checkingAccount->active = 1;
$checkingAccount->currency= "EUR";
$jAccounts->debitAccount = new stdClass();
$debitAccount = $jAccounts->debitAccount;
$debitAccount->accountName = "debit";
$debitAccount->accountBalance = 0;
$debitAccount->active = 1;
$debitAccount->currency = "DKK";
$jAccounts->savingsAccount = new stdClass();
$savingsAccount = $jAccounts->savingsAccount;
$savingsAccount->accountName = "savings";
$savingsAccount->accountBalance = 0;
$savingsAccount->active = 1;
$savingsAccount->currency = "DKK";

$debitAccountBalance = $debitAccount->accountBalance;
$checkingAccountBalance = $checkingAccount->accountBalance;
```

```
$savingsAccountBalance = $savingsAccount->accountBalance;

$jClient->totalBalance = new stdClass();
$totalBalance = $jClient->totalBalance;
$totalBalance->name = "balance";
$overallBalance = ($checkingAccountBalance * 7.47) + $debitAccountBalance +
$savingsAccountBalance;
$totalBalance->balance = $overallBalance;
$totalBalance->currency = "DKK";

$jClient->transactions = new stdClass();
$jClient->transactionsNotRead = new stdClass();
$jClient->creditCards = new stdClass();

$jInnerData->$sPhone = $jClient;

$sData = json_encode($jData, JSON_PRETTY_PRINT);
if($sData == null){ sendResponse(0, __LINE__); }
file_put_contents('../data/clients.json', $sData);

sendResponse(1, __LINE__);

function sendResponse($bStatus, $iLineNumber){
  echo '{"status":'.$bStatus.', "code":'.$iLineNumber.'}';
  exit;
}
```

Then we encode the objects back to JSON strings, check if the data is corrupted and with *file_put_contents* statement we save it into our data file and a positive response is sent. Now when we check our *clients.json* file we can see out client is created successfully.

Login

In the *login.php* page we have a Login form where the user enters his/her details and via AJAX we send a POST request because we do not want our details to pop up in the address bar when logging in. The values are validated both frontend and backend.

In *Figure 3 Api-login.php* I have implemented that the login blocks after 3 attempts, the user then must wait 10 seconds until he can try to login again. When the user logs in correctly the *iLoginAttemptsLeft* reset to 3 and *iLastLoginAttempt* to 0.

Every time the password does not match with the one in the data file, one *iLoginAttemptsLeft* is deducted and the *iLastLoginAttempt* time is saved.

Once the user successfully logs in, a session starts, and a positive response is sent.

```php
if($jInnerData->$sPhone->iLoginAttemptsLeft == 0){
  $iSecondsElapsedSinceLastLoginAttempt = $jInnerData->$sPhone->iLastLoginAttempt + 5 - time();
   if($iSecondsElapsedSinceLastLoginAttempt <= 0){
     if(!password_verify( $sPassword, $jInnerData->$sPhone->password)){
       echo "Wrong credentials. You have to wait again!";
       exit;
     }
        $jInnerData->$sPhone->iLoginAttemptsLeft = 3;
        $jInnerData->$sPhone->iLastLoginAttempt = 0;
        $sData = json_encode($jData, JSON_PRETTY_PRINT);//encode string back to object
        if($sData == null){ sendResponse(0, __LINE__); }//check if corrupted
        file_put_contents('../data/clients.json', $sData);//saving it back to the file
        echo 'You are logged in';
        exit;
     }
   echo "wait $iSecondsElapsedSinceLastLoginAttempt seconds to log in again";
   exit;
   }

if(!password_verify( $sPassword, $jInnerData->$sPhone->password )){
  $jInnerData->$sPhone->iLoginAttemptsLeft --;
  $jInnerData->$sPhone->iLastLoginAttempt = time();
  $sData = json_encode($jData, JSON_PRETTY_PRINT);
  if($sData == null){ sendResponse(0, __LINE__); }
  file_put_contents('../data/clients.json', $sData);
  echo "Wrong credentials. You have {$jInnerData->$sPhone->iLoginAttemptsLeft} attempts left";
  exit;
}

session_start();
$_SESSION['sUserId'] = $sPhone;
sendResponse(1, __LINE__);

function sendResponse($bStatus, $iLineNumber){
  echo '{"status":'.$bStatus.', "code":'.$iLineNumber.'}';
  exit;
}
```

Figure 3 Api-login.php

In the Login page there is Forgot password link which then redirects to *forgot-password.php* page. On this page the user inputs his user phone that is in the database and the form is submitted as a GET request to the *api-forgot-password.php*.

```php
//GET the Number from the database and send an email
$sPhone = $_GET['phone'];

$sData = file_get_contents('../data/clients.json');
$jData = json_decode($sData);
if( $jData == null ){ sendResponse(0, __LINE__); }
$jInnerData = $jData->data;

if(!$jInnerData->$sPhone){
  echo 'the phone doesnt match any in the database!';
}

//setting the activationKey and the link
$jInnerData->$sPhone->activationKey = uniqid();
$sActivationKey = $jInnerData->$sPhone->activationKey;
$sEmail = $jInnerData->$sPhone->email;
$sSubject = "Forgot password link";
$sContent = "Hello! Please click on the following link to get a new password :)!
http://mimi-micek.website/set-new-password.php?phone=$sPhone&activation-key=$sActivationKey";

/* if($sActivationKey != $jInnerData->$sPhone->activationKey){
  echo 'Cannot activate this';
  exit;
} */

$sData = json_encode($jData, JSON_PRETTY_PRINT);
file_put_contents('../data/clients.json', $sData);

mail($sEmail, $sSubject, $sContent);
```

Figure 4 Api-forgot-password.php

Figure 4 Api-forgot-password.php shows that the phone value is received through a GET request and it is checked if it is in the database. If the phone matches to any of the users in the database a unique id activation key is created and saved in the user's id. Email is pointed to the user's email in database and the variables *$sSubject* and *$sContent* are created. The *$sPhone* and the *$sActivationKey* are passed in the link which is emailed to the user so the user can access the page with the matching activation key and change his/her password and access the account again.

The user is redirected to the *set-new-password.php* page with the passed phone and activation key value where he chooses a new password that is then POSTed via AJAX to *api-set-new-password.php*, validated, hashed and stored to the data file.

Profile

```php
session_start();

if(!isset($_SESSION['sUserId'])){
  header('Location: login');
}

$sUserId = $_SESSION['sUserId'];

$sData = file_get_contents('data/clients.json');
$jData = json_decode($sData);
if($jData == null){ echo 'System update'; }
$jInnerData = $jData->data;
$jClient = $jInnerData->$sUserId;
?>

  <div id="profile" class="page">
    <div class="row">
      <a href="admin">Admin page</a>
    </div>
    <h1>PROFILE</h1>
    <br>
    <br>
    <div><b>Name:</b> <?= $jClient->name; ?></div>
    <div><b>Last Name:</b> <?= $jClient->lastName; ?> </div>
    <div><b>Email:</b> <?= $jClient->email; ?> </div>
    <div><b>Phone:</b> <?= $sUserId; ?> </div>
    <div><b>Balance:</b>
      <span id="lblBalance">
        <?= $jClient->totalBalance->balance; ?>
        <?= $jClient->totalBalance->currency; ?>
      </span>
```

Figure 5 Profile.php

In *Figure 5 Profile.php* page we include the *$_SESSION* which holds the user id when logged in and if the user is not logged in it redirects to the Login page. We fetch the data from the file from the logged in user and show his details and account balance.

Profile page also contains Change password form and Transfer money form and shows all transactions that have been made from other banks. Similar like *set-new-password.php,* Change password form takes 3 inputs *txtOldPassword, txtNewPassword and txtConfirmNewPassword* that are submitted via the form, validated frontend and backend and saved to the database if everything matches.

In the *profile.js* we have a function *fnvGetBalance* which fetches the balance from server whenever the balance is changed and an AJAX request which fetches and loops through all the transactions from other banks that have not been read and shows them in the Profile page.

Transfer form is used for both transfers between users from different banks and between the users in the same bank and it takes three inputs that are then passed as a GET to the *transfer-money.js* file.

The *transfer-money.js* file passes the *phone, amount and message* via AJAX to *api-transfer.php* where the user id is fetched and passed values are validated. For transfer within our own bank we point to the passed phone which is also a user id receiver of the money, we deduct the passed amount from our user balance and add it to receiver user's balance and save it to the data file.

For the transfer from our bank to another bank from the list of Central bank we use a function *fnjGetListOfBanksFromCentralBank()* where we fetch a list of all banks and loop through it to find a matching user id where the money will be transferred. If successful we get a response from another bank and if not response 'Phone does not exist' is returned.

Accounts

```php
session_start();

if(!isset($_SESSION['sUserId'])){
  header('Location: login');
}

$sUserId = $_SESSION['sUserId'];

$sData = file_get_contents('data/clients.json');
$jData = json_decode($sData);
if($jData == null){ echo 'System update'; }
$jInnerData = $jData->data;
$jClient = $jInnerData->$sUserId;
$jAccounts = $jClient->accounts;
?>

  <div id="accounts" class="page">
    <h1>ACCOUNTS</h1>
    <br>
    <div><b> Checking account:</b>
      <?= $jAccounts->checkingAccount->accountBalance; ?>
      <?= $jAccounts->checkingAccount->currency;?>
    </div>
    <br>
    <div><b> Debit account:</b>
      <?= $jAccounts->debitAccount->accountBalance; ?>
      <?= $jAccounts->debitAccount->currency;?>
    </div>
    <br>
    <div><b>Savings account: </b>
      <?= $jAccounts->savingsAccount->accountBalance; ?>
      <?= $jAccounts->savingsAccount->currency;?>
    </div>
```

Figure 6 Accounts.php

Accounts page contains all the information about the user's accounts, firstly the check is performed if the user is logged in. From the data file we show the Checking account, Debit account and Savings account, account's balances and the account's currencies.

Accounts page also contains the Transfer between accounts form and link to the Credit cards page. With the *frmTransferBetweenAccounts* form we pass via GET request the *fromAccount* account name from which we are transferring from, *toAccount* name of the account to which we are transferring to and the *amount* value which we want to transfer.

In the *api-transfer-between-accounts.php* we checked the values passed from the *frmTransferBetweenAccounts* form. For the purpose of transfer, I have defined name, balance and currency variables for each of the accounts.

I have decided to use *switch* to check if the account names from the inputs match any of the accounts in the data file, firstly setting the fetched strings to lower. Each case is representing one bank account or overall balance. The checking account is set up to be in EUR, so the amount is divided by the exchange rate. If both *$sFromAccount and $sToAccount* match and the amount is not greater than the account's balance, the amount passed is added to the account's balance and saved; if no account is found a negative response is sent back.

```
switch(strtolower($sFromAccount)){

  case $overallBalanceName:

    if($iAmount > $jInnerData->$sUserId->totalBalance->balance){
      sendResponse(-1, __LINE__, 'You dont have enough money in your account');
    }

    $jInnerData->$sUserId->totalBalance->balance -= $iAmount;

    if(strtolower($sToAccount) == $checkingAccountName){
      $jInnerData->$sUserId->accounts->checkingAccount->accountBalance += $iAmount / 7.47;
    }else if($sToAccount == $debitAccountName){
      $jInnerData->$sUserId->accounts->debitAccount->accountBalance += $iAmount;
    }else{
      $jInnerData->$sUserId->accounts->savingsAccount->accountBalance += $iAmount;
    }

    $sData = json_encode($jData, JSON_PRETTY_PRINT);
    file_put_contents('../data/clients.json', $sData);

    break;

  case $checkingAccountName:

    if($iAmount > $jInnerData->$sUserId->accounts->checkingAccount->accountBalance){
      sendResponse(-1, __LINE__, 'You dont have enough money in your account');
    }

    $jInnerData->$sUserId->accounts->checkingAccount->accountBalance -= $iAmount / 7.47;

    if(strtolower($sToAccount) == $overallBalanceName){
      $jInnerData->$sUserId->totalBalance->balance += $iAmount;
    }else if($sToAccount == $debitAccountName){
      $jInnerData->$sUserId->accounts->debitAccount->accountBalance += $iAmount;
    }else{
      $jInnerData->$sUserId->accounts->savingsAccount->accountBalance += $iAmount;
    }

    $sData = json_encode($jData, JSON_PRETTY_PRINT);
```

```
    file_put_contents('../data/clients.json', $sData);

    break;

  case $debitAccountName:

    if($iAmount > $jInnerData->$sUserId->accounts->debitAccount->accountBalance){
      sendResponse(-1, __LINE__, 'You dont have enough money in your account');
    }

    $jInnerData->$sUserId->accounts->debitAccount->accountBalance -= $iAmount;

    if(strtolower($sToAccount) == $overallBalanceName){
      $jInnerData->$sUserId->totalBalance->balance += $iAmount;
    }else if($sToAccount == $checkingAccountName){
      $jInnerData->$sUserId->accounts->checkingAccount->accountBalance += $iAmount / 7.47;
    }else{
      $jInnerData->$sUserId->accounts->savingsAccount->accountBalance += $iAmount;
    }

    $sData = json_encode($jData, JSON_PRETTY_PRINT);
    file_put_contents('../data/clients.json', $sData);

    break;

  case $savingsAccountName:

    if($iAmount > $jInnerData->$sUserId->accounts->savingsAccount->accountBalance){
      sendResponse(-1, __LINE__, 'You dont have enough money in your account');
    }

    $jInnerData->$sUserId->accounts->savingsAccount->accountBalance -= $iAmount;

    if(strtolower($sToAccount) == $overallBalanceName){
      $jInnerData->$sUserId->totalBalance->balance += $iAmount;
    }else if($sToAccount == $checkingAccountName){
      $jInnerData->$sUserId->accounts->checkingAccount->accountBalance += $iAmount / 7.47;
    }else{
      $jInnerData->$sUserId->accounts->debitAccount->accountBalance += $iAmount;
    }

    $sData = json_encode($jData, JSON_PRETTY_PRINT);
    file_put_contents('../data/clients.json', $sData);

    break;

  default:

    sendResponse( 0, __LINE__ , 'No account of that name could be found' );

}
```

Check my credit cards link takes the user to the *create-credit-card.php* page where he/she can create a new credit card and view list of all their credit cards with an option to block or unblock the card.

frmCreateCard is submitted via AJAX and the POST request is passed to the *api-create-credit-card.php* where credit card group of objects is accessed, and new objects are created such as *$creditCardId* where all the cards will be stored with name, status and credit limit.

```php
ini_set('display_errors', 0);

session_start();

if(!isset($_GET['id'])){
  header('Location: ../create-credit-card');
}

$sUserId = $_SESSION['sUserId'];

$sData = file_get_contents('../data/clients.json');
$jData = json_decode($sData);
if($jData == null){ echo 'System update'; }
$jInnerData = $jData->data;
$jClient = $jInnerData->$sUserId->creditCards;

//looping through customers and finding a credit card id match
foreach($jClient as $sCreditCardId => $creditCard){
  if($_GET['id'] == $sCreditCardId){
    $creditCard->active = !$creditCard->active;
    $sData = json_encode($jData, JSON_PRETTY_PRINT);
    file_put_contents('../data/clients.json', $sData);
    header('Location: ../create-credit-card');
   }
}
```

Figure 7 Api-block-or-unblock-credit-card.php

In *Figure 7 Api-block-or-unblock-credit-card.php* we block the credit card which fetches the id via the GET request from the Create credit card page. The credit card id is looped through all the card ids of a user and if match is found the active status of a card is flipped to *false* and the card becomes inactive, if we want to unblock the card, we just do so in the Credit card page in the list of all cards.

Chat

The chat page is created only for existing users and it works on the principle that in the first input box the number of another user is entered and in the second the desired message is written and send. The messages are then passed via POST to the *api-set-message.php* that saves the user's number and message and saves is to the new data file. With the *api-get-message.php* we fetch the sent messages and append them to the chat via AJAX and jQuery.

Admin

Admin user is just a user registered in the data file. The Admin user can transfer money from his account balance to any other user's account, can see a list of all users in the system and their information details and balances, and is also able to block/unblock any user account. The Admin page is accessed via link in the Profile page.

```
<div id="admin" class="page">

    <h1>ACCOUNTS OVERVIEW</h1>
    <?php

        foreach ( $jInnerData as $jClientId => $jClient ) {
            // TERNARY: if true '?' unblock and if not block
            $sWord = ($jClient->active == 0) ? 'UNBLOCK' : 'BLOCK';
            $totalBalance = $jClient->totalBalance->balance;
            echo "<div class='client'>
            <div>ID: $jClientId</div>
            <div>Full name: $jClient->name $jClient->lastName</div>
            <div>Email: $jClient->email</div>
            <div>Balance: $totalBalance</div>
            <div>Status: $jClient->active</div>
            <br>
            <a href='apis/api-block-user-account?id=$jClientId'>$sWord</a>
            <br>
            <br>
            </div>";

        }
        ?>
    <br>
    <br>
    <form id="frmAdminTransfer" action="apis/api-admin-transfer" method="POST">
        <h2>Transfer money to any account</h2>
        <input name="phone" type="tel" placeholder="phone">
        <br>
        <input name="amount" type="number" placeholder="amount">
        <br>
        <button>Transfer money</button>
    </form>
    </div>
```

Figure 8 Admin.php

The Admin transfer is directly submitted to the *api-admin-transfer.php* which then validates the inputs and adds the amount to the balance. After the request is processed the user is redirected to the Admin page.

```php
if( !isset( $_GET['id'] )) {
    header('Location: ../admin');
}

$sData = file_get_contents('../data/clients.json');
$jData = json_decode($sData);

if ( $jData == null ) { echo 'json cannot be read'; }

// loop through customers and find id match
foreach( $jData->data as $jClientId => $jClient ) {

    if ( $jClientId == $_GET['id'] ) {

        // flip the active key to 0 - maths: true false flips
        $jClient->active = ! $jClient->active;

        if ( $jClient->active == false ) {
            $jClient->active = 0;
        }

        // convert json back to text and
        $sData = json_encode($jData, JSON_PRETTY_PRINT);

        // save data back to file
        file_put_contents('../data/clients.json', $sData);

        // redirect user to view-customers.php
        header ('Location: ../admin');
    }
}

echo 'no match';
```

Figure 9 Api-block-user-account.php

Figure 9 Api-block-user-account.php processes the Admin block/unblock request of any client/user in the database. With a *foreach* loop we loop through all the clients in the data file and when the passed id matches the id of the client in the data file the chosen client is blocked, and client's status is changed to inactive. The new status is then saved to the file and redirection to the Admin page is activated.

Conclusion

This application was very interesting to make from scratch because we have built both frontend and backend at the same time. If we had more time, I would expand the functionality of the application and I would make the UI look better and I would like to learn how to work better with jQuery. I am looking forward to learning how to connect this application to MySQL database.