Labb rapport 3

Internet Applications, ID1354

David Nartey
dnartey@kth.se

2015-10-04

1 Introduction

The goal of the third lab is to learn how to use object-oriented design of the PHP server, using the MVC pattern and / or PHP framework. The goal also went on to teach;

- How to avoid certain basic security threats in a web application .
- How to use a database to PHP server and improve PHP performance by using cache and conservation connectivity .

The first task is to use the MVC pattern or PHP framework to get a good pattern architecture.
The second task was to improve the security of the website; Students had the option of choosing ⅗ security measures to implement for the the website.
The final task required the usage of an external database platform to host the Web site, manage all users and comments as well

I worked together with Evan Saboo (saboo@kth.se).

2 Literature Studies / Resources

PHP functions were implemented with immense support from the following resources:
* All three lectures on implementing object oriented design with PHP.

* Lots of googling and youtube clips helped us to have a good overview of the object oriented designs.

* We also got help with the security features during the coach/guide sessions
* The following areas of the course litterature ’Programming the WWW 6th by Sebesta Robert W’ contributed immensely ;
- 11.7 Model–View–Controller Application Architecture
- 13.2 An Introduction to the Structured Query Language
- 13.3 Architectures for Database Access
- 13.4 The MySQL Database System
- 13.5 Database Access with PHP and MySQL

3 Method

Compulsory tasks:

1. In the first task , we chose to implement the MVC pattern in our website . We started to check how to create classes and functions on the object-oriented approach . There were several instructions of web on how it is implemented which we used . We created all the necessary functions such as login and logout that were in the User class. We also created features that were not linked to the User but still classified as essential functions such as validation of the login and connection to the database. We created a function that handled the link between business logic and user interface , which is similar to the Controller in the MVC pattern. We also created different namespaces for different classes to distinguish between the significance of ranges and features.
2. The second task was to improve the security of the website and we implemented (3) three security features which were
   a. Database security
   b. Password encryption and
   c. Cross Site Scripting .

Database security:

Our implementation of the database was that we created a user who only had access to very few permissions like login and comment in the database. The website can only allow a logged in user with a limited access to the database.

Password encryption:

This was implemented by using salt to provide randomly generated secure string of data unto the end of a password. We also ensured that the salt was generated with the aid of (sha256) Secure Hash Algorithm 256. (Password + salt = Hash). An md5 encryption is then performed on the generated hash and thereby making password decryption very strenuous. The securely created string is then saved in the database and used to validate the login of an existing user.

Cross Site Scripting:

It took two different functions in order to prevent security threats with Cross Site Scripting .

- ❏ The first function omvandlar a value to an html ' plain text ' . Feature prevents the user to create HTML, PHP or JAVASCRIPT code in the input fields because the code is displayed as plain text.
- ❏ The second function prevents the possibility of an attacker in successfully carrying database calls by entering codes in the input/text fields. This also ensures that all special characters are converted to plain text to prevent SQL injections to the database.

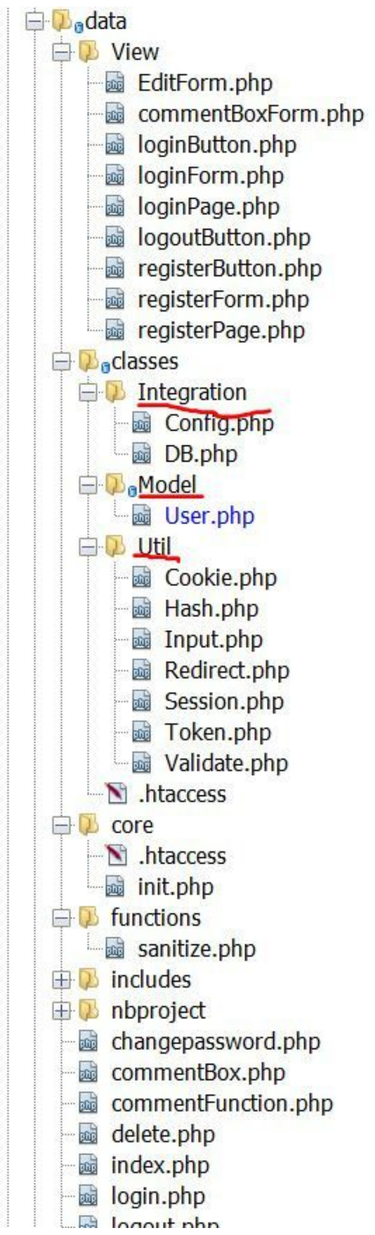
  We also implemented a feature that filters out all characters except letters and numbers when users must register to enter their user name.

Optional task 2, Use a Database:

The last optional task was about using a database to manage users and comments . We had already created a database with phpmyadmin which we had in lab 2 so this meant that the task did not require much time to perform. The only thing we did was we wrote about the switch function to the database to use the PDO (PHP Data Objects) instead of MySQL.

4 Results/ Discussions:

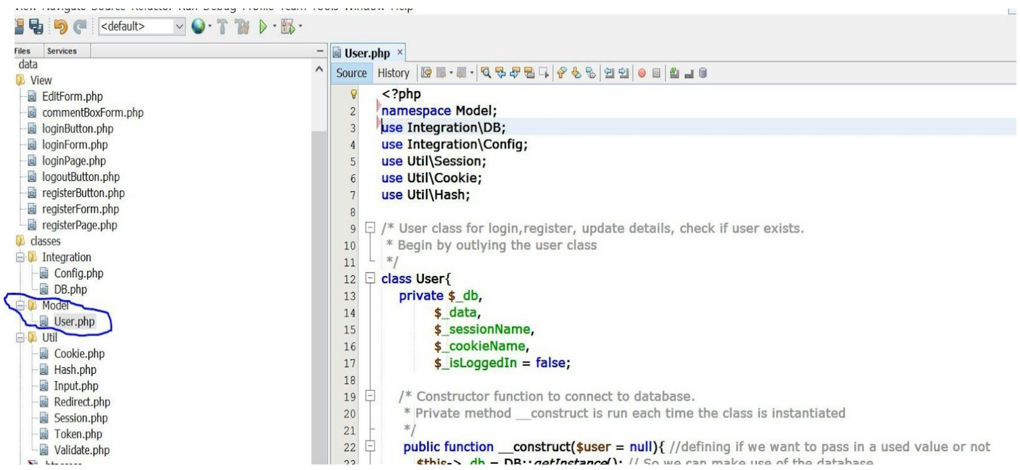
Results: <u>TASK 1</u>



```
data
  View
    EditForm.php
    commentBoxForm.php
    loginButton.php
    loginForm.php
    loginPage.php
    logoutButton.php
    registerButton.php
    registerForm.php
    registerPage.php
  classes
    Integration
      Config.php
      DB.php
    Model
      User.php
    Util
      Cookie.php
      Hash.php
      Input.php
      Redirect.php
      Session.php
      Token.php
      Validate.php
    .htaccess
  core
    .htaccess
    init.php
  functions
    sanitize.php
  includes
  nbproject
  changepassword.php
  commentBox.php
  commentFunction.php
  delete.php
  index.php
  login.php
  logout.php
```

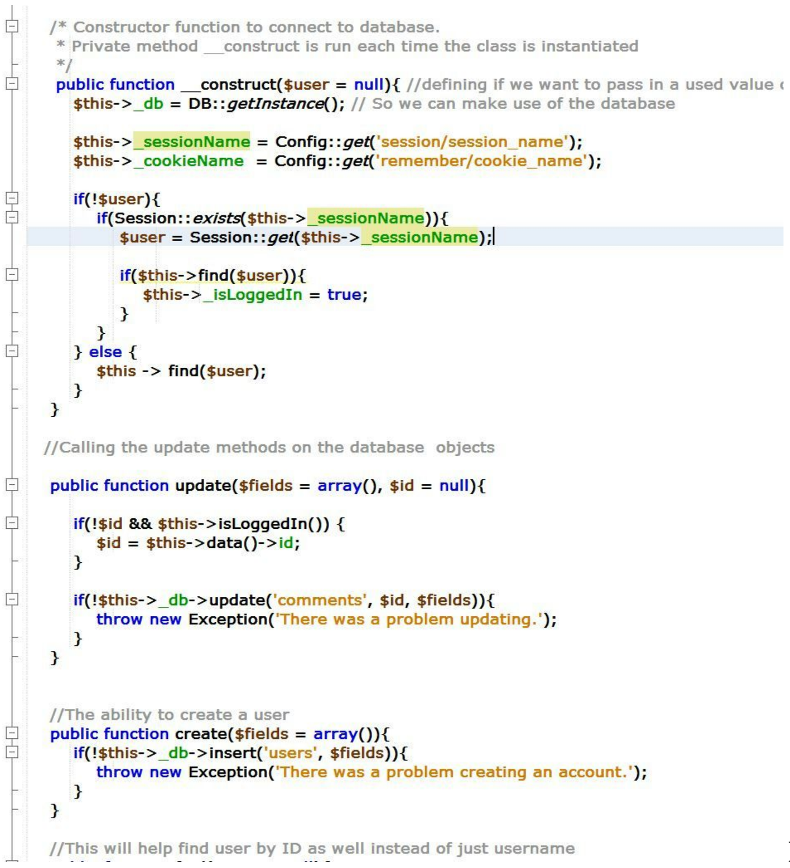
Figure 1: Class hierarchy

MODEL



```
data
View
    EditForm.php
    commentBoxForm.php
    loginButton.php
    loginForm.php
    loginPage.php
    logoutButton.php
    registerButton.php
    registerForm.php
    registerPage.php
classes
    Integration
        Config.php
        DB.php
    Model
        User.php
    Util
        Cookie.php
        Hash.php
        Input.php
        Redirect.php
        Session.php
        Token.php
        Validate.php
```

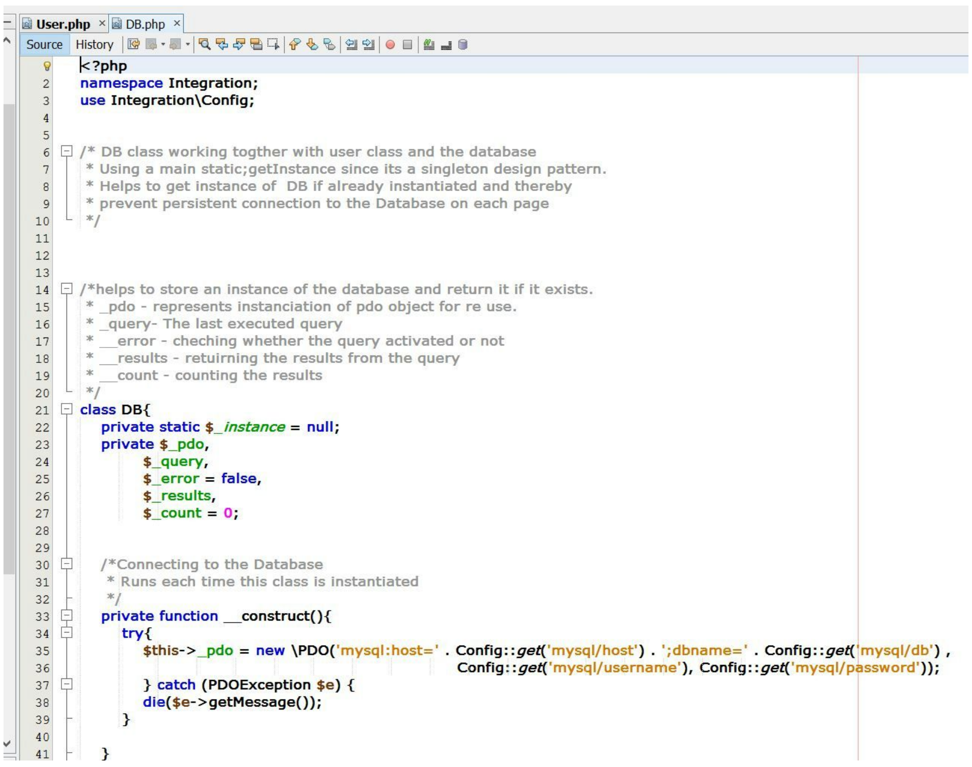
```php
<?php
namespace Model;
use Integration\DB;
use Integration\Config;
use Util\Session;
use Util\Cookie;
use Util\Hash;

/* User class for login,register, update details, check if user exists.
 * Begin by outlying the user class
 */
class User{
    private $_db,
            $_data,
            $_sessionName,
            $_cookieName,
            $_isLoggedIn = false;

    /* Constructor function to connect to database.
     * Private method __construct is run each time the class is instantiated
     */
     public function __construct($user = null){ //defining if we want to pass in a used value or not
```



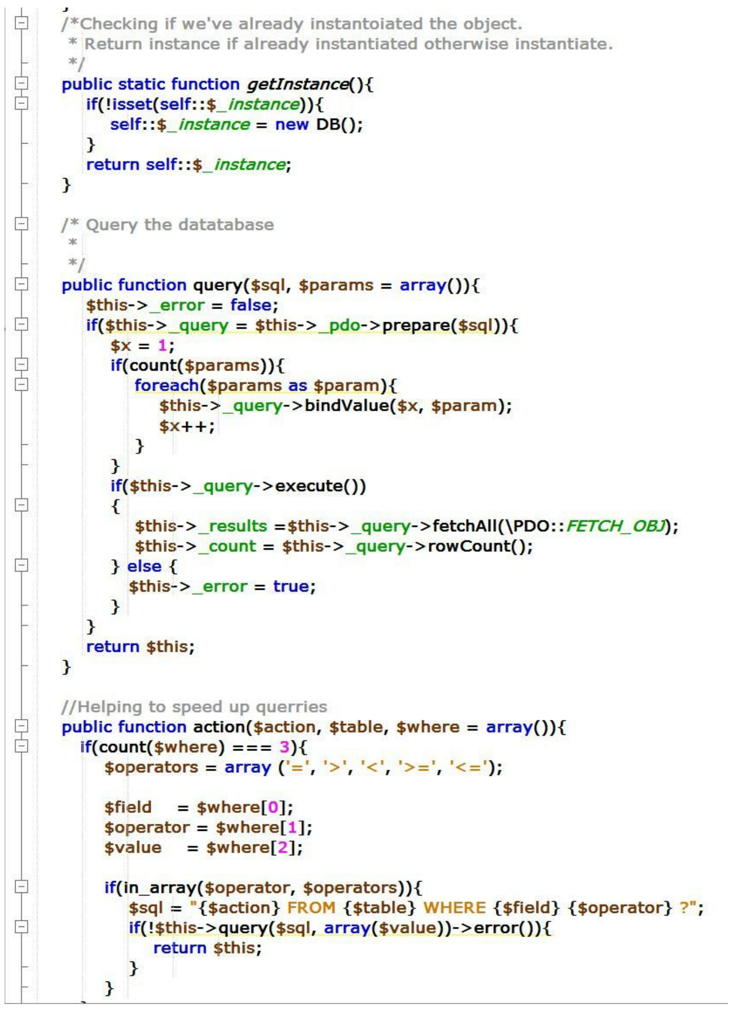
```php
    /* Constructor function to connect to database.
     * Private method __construct is run each time the class is instantiated
     */
     public function __construct($user = null){ //defining if we want to pass in a used value
        $this->_db = DB::getInstance(); // So we can make use of the database

        $this->_sessionName = Config::get('session/session_name');
        $this->_cookieName  = Config::get('remember/cookie_name');

        if(!$user){
            if(Session::exists($this->_sessionName)){
                $user = Session::get($this->_sessionName);

                if($this->find($user)){
                    $this->_isLoggedIn = true;
                }
            }
        } else {
            $this -> find($user);
        }
    }

    //Calling the update methods on the database  objects

    public function update($fields = array(), $id = null){

        if(!$id && $this->isLoggedIn()) {
            $id = $this->data()->id;
        }

        if(!$this->_db->update('comments', $id, $fields)){
            throw new Exception('There was a problem updating.');
        }
    }


    //The ability to create a user
    public function create($fields = array()){
        if(!$this->_db->insert('users', $fields)){
            throw new Exception('There was a problem creating an account.');
        }
    }

    //This will help find user by ID as well instead of just username
```

Figure 2.: User class

INTEGRATION

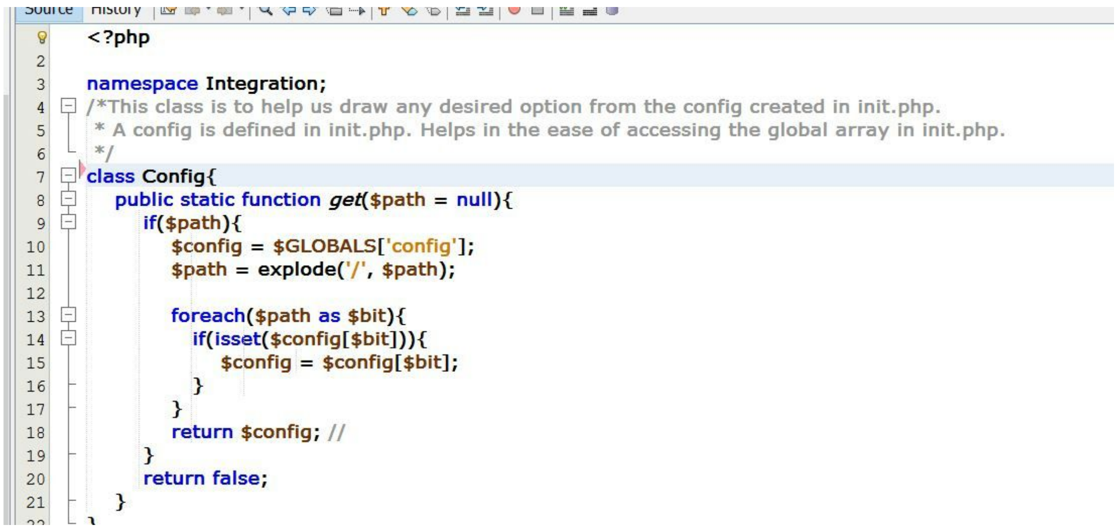


```
<?php
namespace Integration;
use Integration\Config;


/* DB class working togther with user class and the database
 * Using a main static;getInstance since its a singleton design pattern.
 * Helps to get instance of  DB if already instantiated and thereby
 * prevent persistent connection to the Database on each page
 */



/*helps to store an instance of the database and return it if it exists.
 * _pdo - represents instanciation of pdo object for re use.
 * _query- The last executed query
 * __error - cheching whether the query activated or not
 * __results - retuirning the results from the query
 * __count - counting the results
 */
class DB{
    private static $_instance = null;
    private $_pdo,
            $_query,
            $_error = false,
            $_results,
            $_count = 0;


    /*Connecting to the Database
     * Runs each time this class is instantiated
     */
    private function __construct(){
        try{
            $this->_pdo = new \PDO('mysql:host=' . Config::get('mysql/host') . ';dbname=' . Config::get('mysql/db') ,
                                                            Config::get('mysql/username'), Config::get('mysql/password'));
            } catch (PDOException $e) {
            die($e->getMessage());
        }

    }
```



```
/*Checking if we've already instantoiated the object.
 * Return instance if already instantiated otherwise instantiate.
 */
public static function getInstance(){
    if(!isset(self::$_instance)){
        self::$_instance = new DB();
    }
    return self::$_instance;
}

/* Query the datatabase
 *
 */
public function query($sql, $params = array()){
    $this->_error = false;
    if($this->_query = $this->_pdo->prepare($sql)){
        $x = 1;
        if(count($params)){
            foreach($params as $param){
                $this->_query->bindValue($x, $param);
                $x++;
            }
        }
        if($this->_query->execute())
        {
            $this->_results =$this->_query->fetchAll(\PDO::FETCH_OBJ);
            $this->_count = $this->_query->rowCount();
        } else {
           $this->_error = true;
        }
    }
    return $this;
}

//Helping to speed up querries
public function action($action, $table, $where = array()){
   if(count($where) === 3){
       $operators = array ('=', '>', '<', '>=', '<=');

       $field    = $where[0];
       $operator = $where[1];
       $value    = $where[2];

       if(in_array($operator, $operators)){
           $sql = "{$action} FROM {$table} WHERE {$field} {$operator} ?";
           if(!$this->query($sql, array($value))->error()){
               return $this;
           }
       }
```

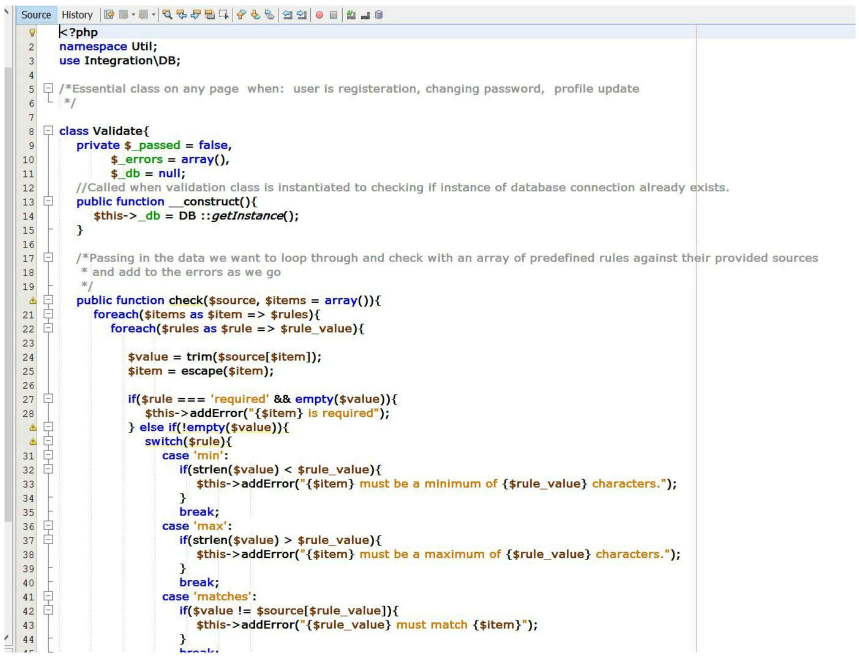
Figure 3: Database class



```php
<?php

namespace Integration;
/*This class is to help us draw any desired option from the config created in init.php.
 * A config is defined in init.php. Helps in the ease of accessing the global array in init.php.
 */
class Config{
    public static function get($path = null){
        if($path){
            $config = $GLOBALS['config'];
            $path = explode('/', $path);

            foreach($path as $bit){
               if(isset($config[$bit])){
                   $config = $config[$bit];
               }
            }
            return $config; //
        }
        return false;
    }
```

figure 4 : Config class

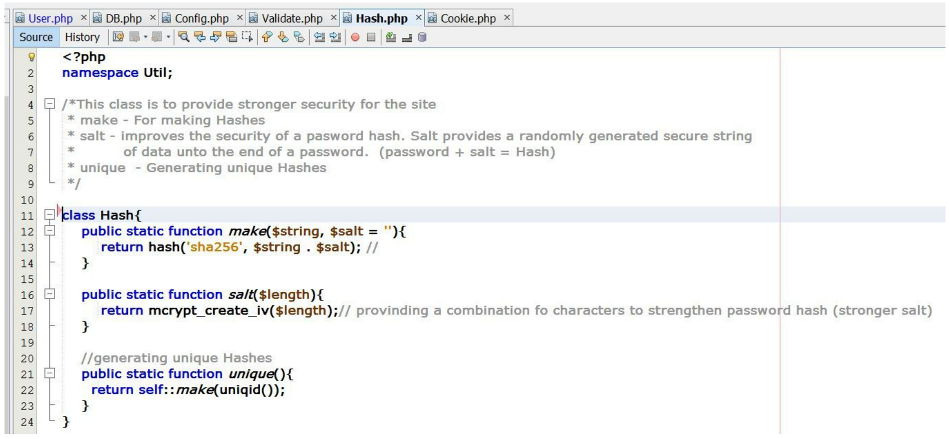
UTIL



Source History

```php
<?php
namespace Util;
use Integration\DB;

/*Essential class on any page  when:  user is registeration, changing password,  profile update
 */

class Validate{
    private $_passed = false,
            $_errors = array(),
            $_db = null;
    //Called when validation class is instantiated to checking if instance of database connection already exists.
    public function __construct(){
        $this->_db = DB ::getInstance();
    }

    /*Passing in the data we want to loop through and check with an array of predefined rules against their provided sources
     * and add to the errors as we go
     */
    public function check($source, $items = array()){
        foreach($items as $item => $rules){
            foreach($rules as $rule => $rule_value){

                $value = trim($source[$item]);
                $item = escape($item);

                if($rule === 'required' && empty($value)){
                    $this->addError("{$item} is required");
                } else if(!empty($value)){
                    switch($rule){
                        case 'min':
                            if(strlen($value) < $rule_value){
                                $this->addError("{$item} must be a minimum of {$rule_value} characters.");
                            }
                            break;
                        case 'max':
                            if(strlen($value) > $rule_value){
                                $this->addError("{$item} must be a maximum of {$rule_value} characters.");
                            }
                            break;
                        case 'matches':
                            if($value != $source[$rule_value]){
                                $this->addError("{$rule_value} must match {$item}");
                            }
```

Figure 5: Validation class

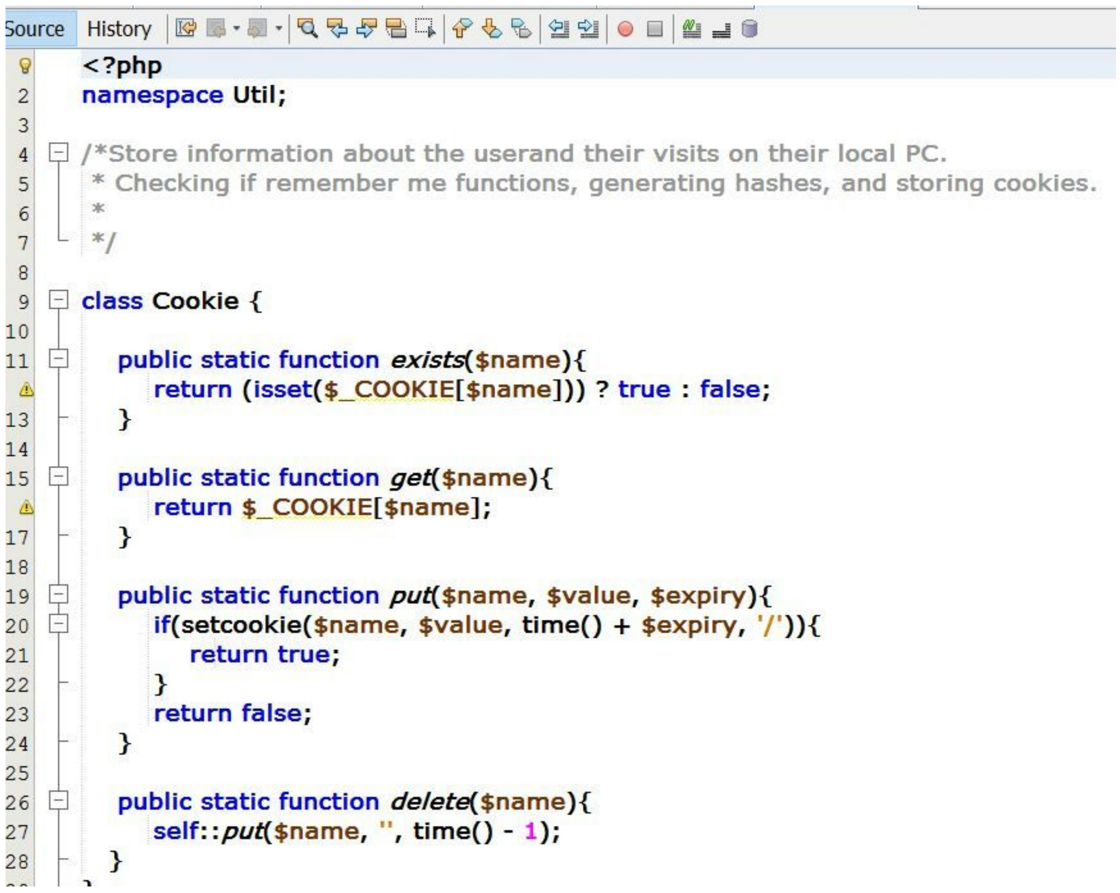


User.php × DB.php × Config.php × Validate.php × Hash.php × Cookie.php ×

Source History

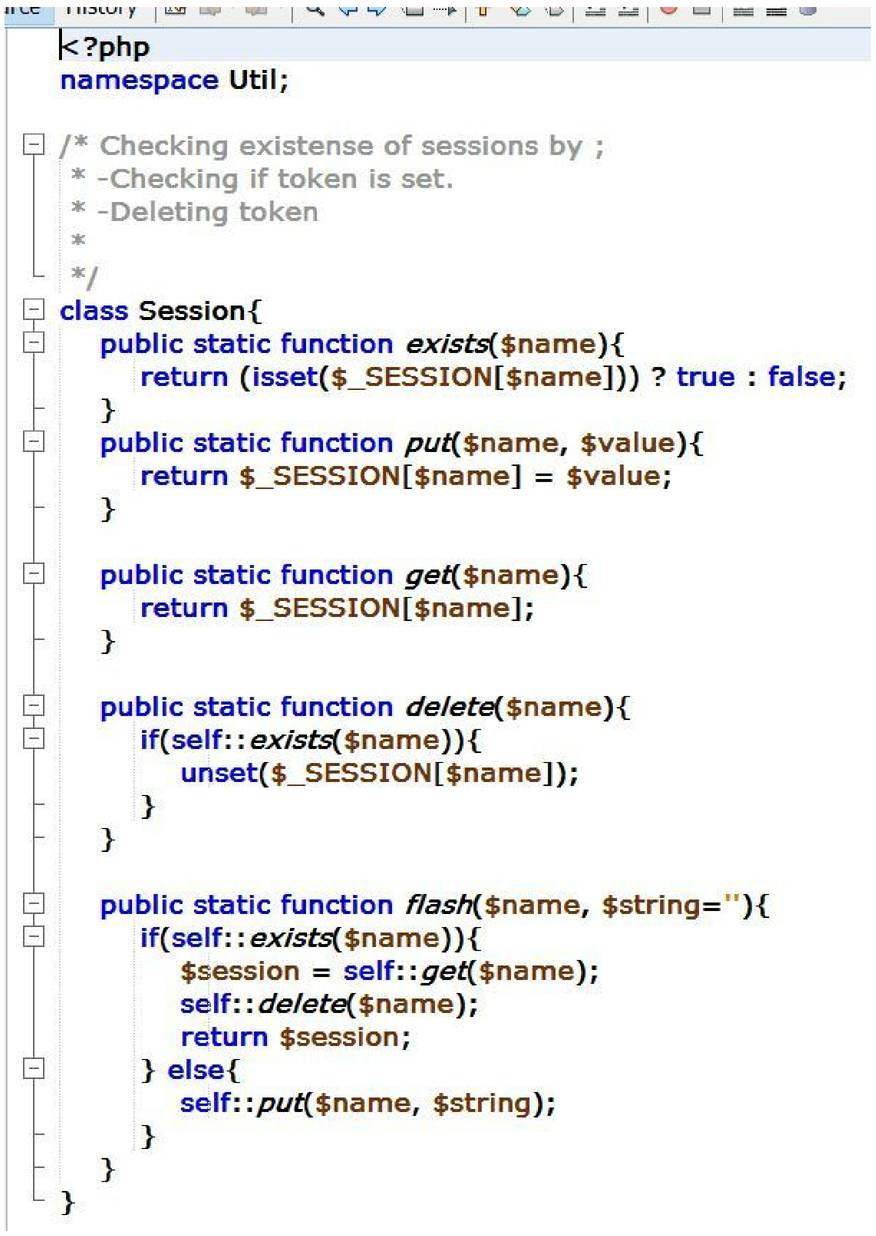
```php
<?php
namespace Util;

/*This class is to provide stronger security for the site
 * make - For making Hashes
 * salt - improves the security of a pasword hash. Salt provides a randomly generated secure string
 *        of data unto the end of a password.  (password + salt = Hash)
 * unique  - Generating unique Hashes
 */

class Hash{
    public static function make($string, $salt = ''){
        return hash('sha256', $string . $salt); //
    }

    public static function salt($length){
        return mcrypt_create_iv($length);// provinding a combination fo characters to strengthen password hash (stronger salt)
    }

    //generating unique Hashes
    public static function unique(){
      return self::make(uniqid());
    }
}
```

Figure 6: Hash class


Source History

```
<?php
namespace Util;

/*Store information about the userand their visits on their local PC.
 * Checking if remember me functions, generating hashes, and storing cookies.
 *
 */

class Cookie {

    public static function exists($name){
        return (isset($_COOKIE[$name])) ? true : false;
    }

    public static function get($name){
        return $_COOKIE[$name];
    }

    public static function put($name, $value, $expiry){
        if(setcookie($name, $value, time() + $expiry, '/')){
            return true;
        }
        return false;
    }

    public static function delete($name){
        self::put($name, '', time() - 1);
   }
```

Figure 7: Cookie class

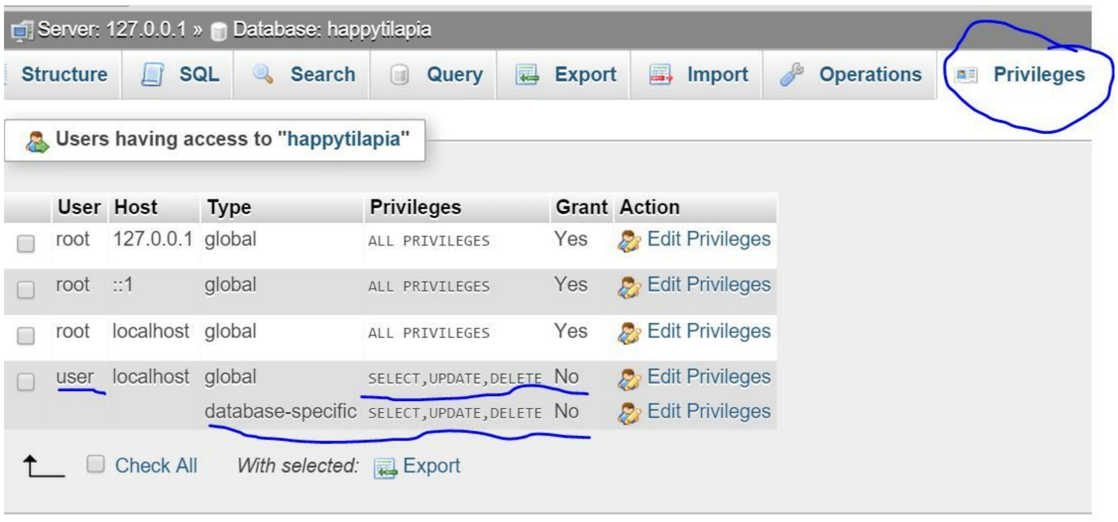


```
<?php
namespace Util;

/* Checking existense of sessions by ;
 * -Checking if token is set.
 * -Deleting token
 *
 */
class Session{
    public static function exists($name){
        return (isset($_SESSION[$name])) ? true : false;
    }
    public static function put($name, $value){
        return $_SESSION[$name] = $value;
    }

    public static function get($name){
        return $_SESSION[$name];
    }

    public static function delete($name){
        if(self::exists($name)){
            unset($_SESSION[$name]);
        }
    }

    public static function flash($name, $string=''){
        if(self::exists($name)){
            $session = self::get($name);
            self::delete($name);
            return $session;
        } else{
            self::put($name, $string);
        }
    }
}
```

Figure 8 : Session class

Task 2, Security

Database security:

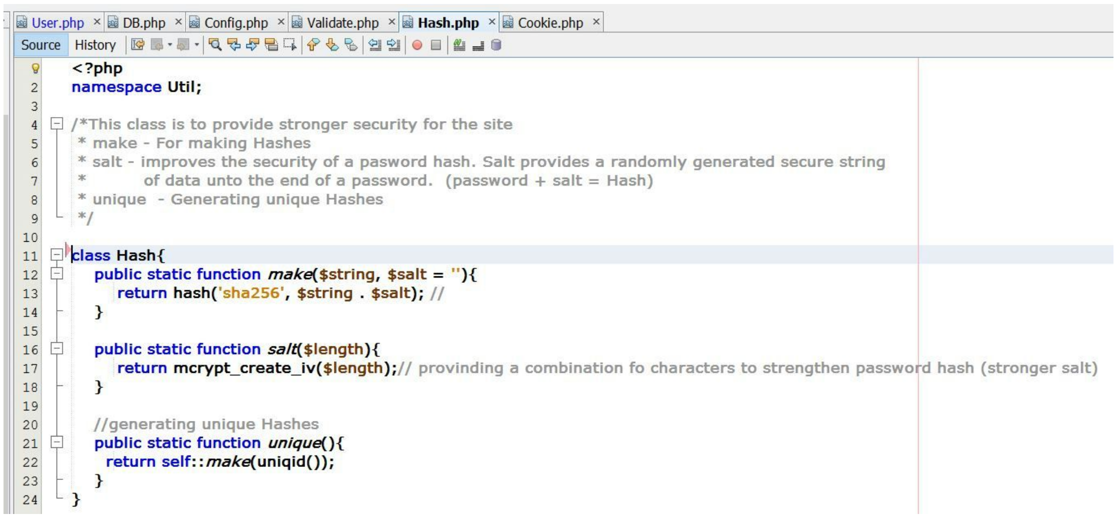


Users having access to "happytilapia"

	User	Host	Type	Privileges	Grant	Action
☐	root	127.0.0.1	global	ALL PRIVILEGES	Yes	Edit Privileges
☐	root	::1	global	ALL PRIVILEGES	Yes	Edit Privileges
☐	root	localhost	global	ALL PRIVILEGES	Yes	Edit Privileges
☐	user	localhost	global	SELECT,UPDATE,DELETE	No	Edit Privileges
			database-specific	SELECT,UPDATE,DELETE	No	Edit Privileges

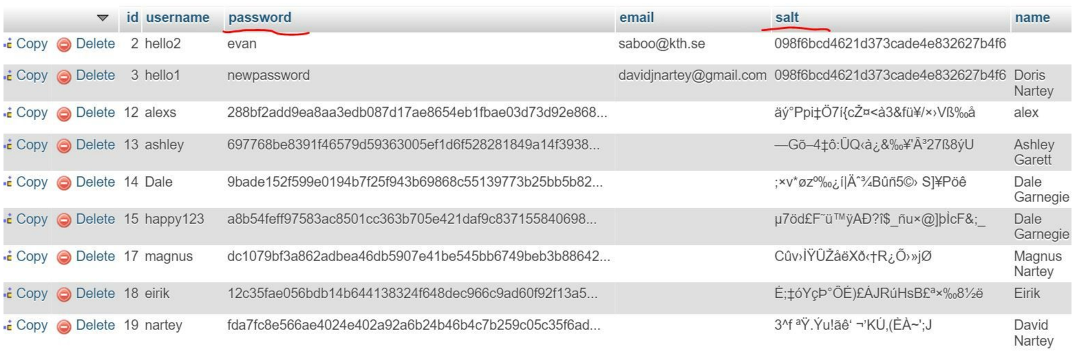
Figure 9 showing limited user privileges in to the database

Password encryption:



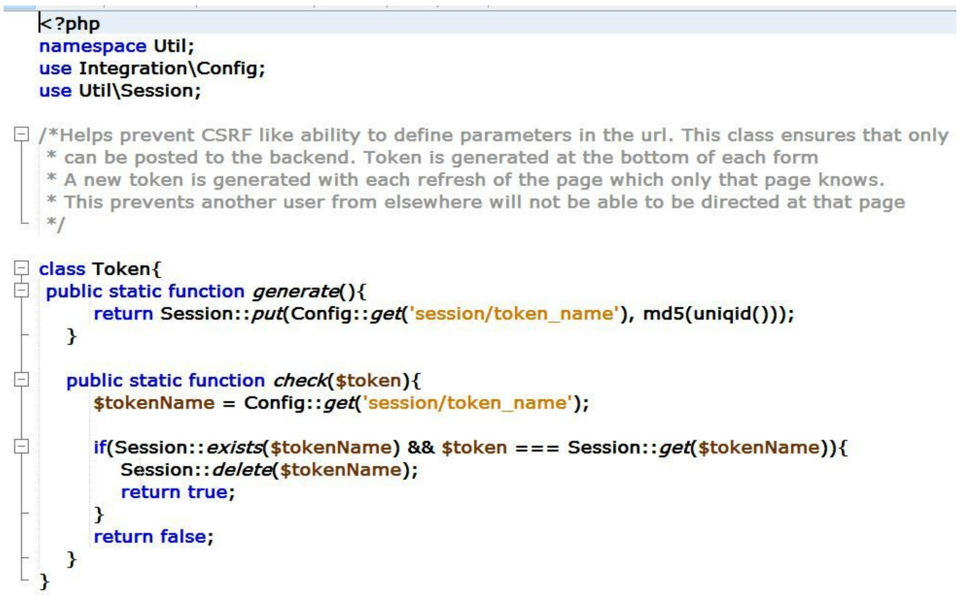
```php
<?php
namespace Util;

/*This class is to provide stronger security for the site
 * make - For making Hashes
 * salt - improves the security of a pasword hash. Salt provides a randomly generated secure string
 *         of data unto the end of a password.  (password + salt = Hash)
 * unique  - Generating unique Hashes
 */

class Hash{
    public static function make($string, $salt = ''){
        return hash('sha256', $string . $salt); //
    }

    public static function salt($length){
        return mcrypt_create_iv($length);// provinding a combination fo characters to strengthen password hash (stronger salt)
    }

    //generating unique Hashes
    public static function unique(){
      return self::make(uniqid());
    }
}
```

Figure 10: Hash class



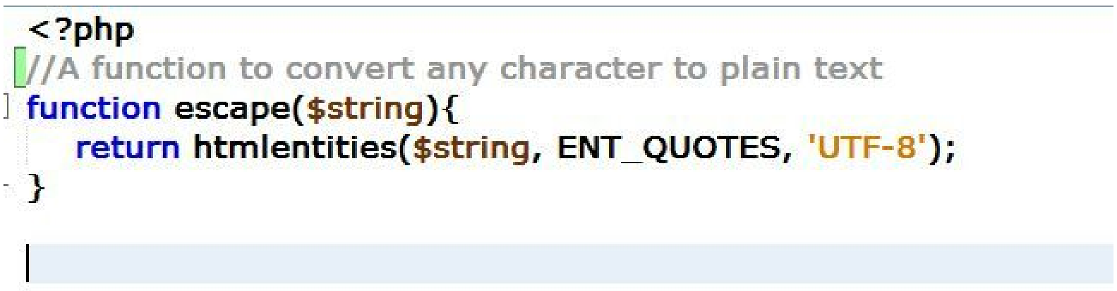
		id	username	password	email	salt	name
Copy	Delete	2	hello2	evan	saboo@kth.se	098f6bcd4621d373cade4e832627b4f6	
Copy	Delete	3	hello1	newpassword	davidjnartey@gmail.com	098f6bcd4621d373cade4e832627b4f6	Doris Nartey
Copy	Delete	12	alexs	288bf2add9ea8aa3edb087d17ae8654eb1fbae03d73d92e868...		äý°Ppi‡Ö7í{cŽ¤<à3&fü¥/×›Vß‰å	alex
Copy	Delete	13	ashley	697768be8391f46579d59363005ef1d6f528281849a14f3938...		—Gõ–4‡ô:ÛQ‹à¿&‰¥'Â³27ß8ýU	Ashley Garett
Copy	Delete	14	Dale	9bade152f599e0194b7f25f943b69868c55139773b25bb5b82...		;×v*øzº‰¿í|Äˆ¾Bûñ5©› S]¥Pöê	Dale Garnegie
Copy	Delete	15	happy123	a8b54feff97583ac8501cc363b705e421daf9c837155840698...		µ7öd£F˜ü™ÿAĐ?î$_ñu×@]þÌcF&;_	Dale Garnegie
Copy	Delete	17	magnus	dc1079bf3a862adbea46db5907e41be545bb6749beb3b88642...		Cûv›ÌŸÛŽåëXð‹†R¿Õ›»jØ	Magnus Nartey
Copy	Delete	18	eirik	12c35fae056bdb14b644138324f648dec966c9ad60f92f13a5...		É;‡óYçÞ°ÕÉ)£ÁJRúHsB£ª×‰8½ë	Eirik
Copy	Delete	19	nartey	fda7fc8e566ae4024e402a92a6b24b46b4c7b259c05c35f6ad...		3^f ªŸ.Ýu!ãê' ¬'KÚ,(ÈÀ~';J	David Nartey

Figure 11 showing hashed password (salt) in the Database



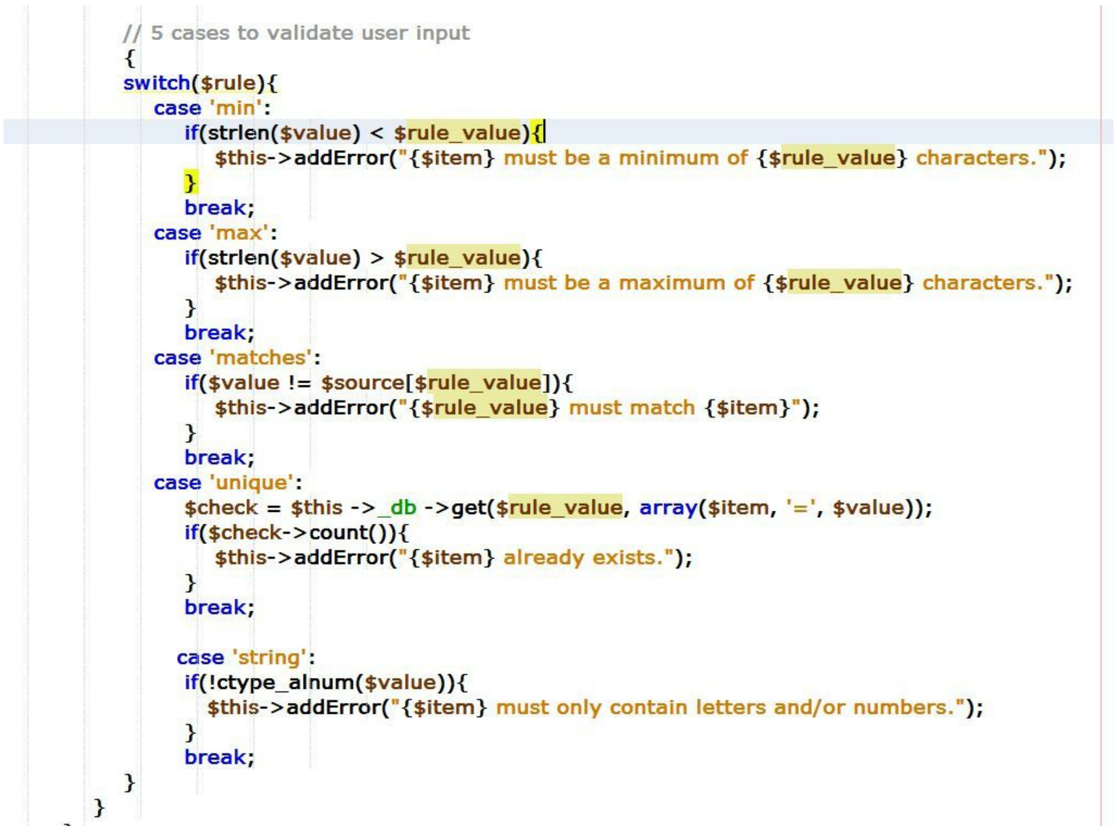
```php
<?php
namespace Util;
use Integration\Config;
use Util\Session;

/*Helps prevent CSRF like ability to define parameters in the url. This class ensures that only
 * can be posted to the backend. Token is generated at the bottom of each form
 * A new token is generated with each refresh of the page which only that page knows.
 * This prevents another user from elsewhere will not be able to be directed at that page
 */

class Token{
 public static function generate(){
        return Session::put(Config::get('session/token_name'), md5(uniqid()));
    }

    public static function check($token){
        $tokenName = Config::get('session/token_name');

        if(Session::exists($tokenName) && $token === Session::get($tokenName)){
            Session::delete($tokenName);
            return true;
        }
        return false;
    }
}
```

Figure 12 showing Cross Site Request Forgery:



```php
<?php
//A function to convert any character to plain text
function escape($string){
    return htmlentities($string, ENT_QUOTES, 'UTF-8');
}
```

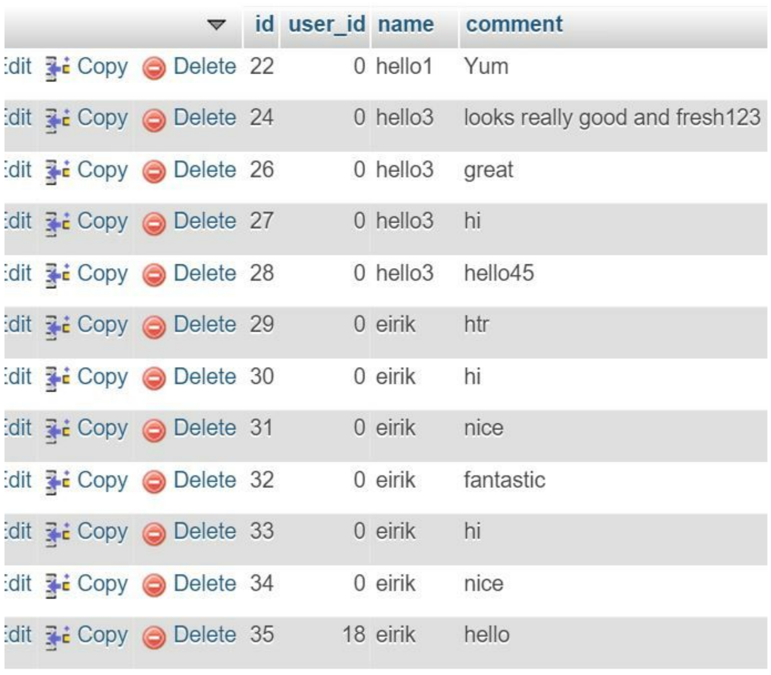
Figure 13 showing Sanitize class



```
        // 5 cases to validate user input
        {
        switch($rule){
           case 'min':
              if(strlen($value) < $rule_value){
                 $this->addError("{$item} must be a minimum of {$rule_value} characters.");
              }
              break;
           case 'max':
              if(strlen($value) > $rule_value){
                 $this->addError("{$item} must be a maximum of {$rule_value} characters.");
              }
              break;
           case 'matches':
              if($value != $source[$rule_value]){
                 $this->addError("{$rule_value} must match {$item}");
              }
              break;
           case 'unique':
              $check = $this ->_db ->get($rule_value, array($item, '=', $value));
              if($check->count()){
                 $this->addError("{$item} already exists.");
              }
              break;

             case 'string':
              if(!ctype_alnum($value)){
                $this->addError("{$item} must only contain letters and/or numbers.");
              }
              break;
        }
    }
```

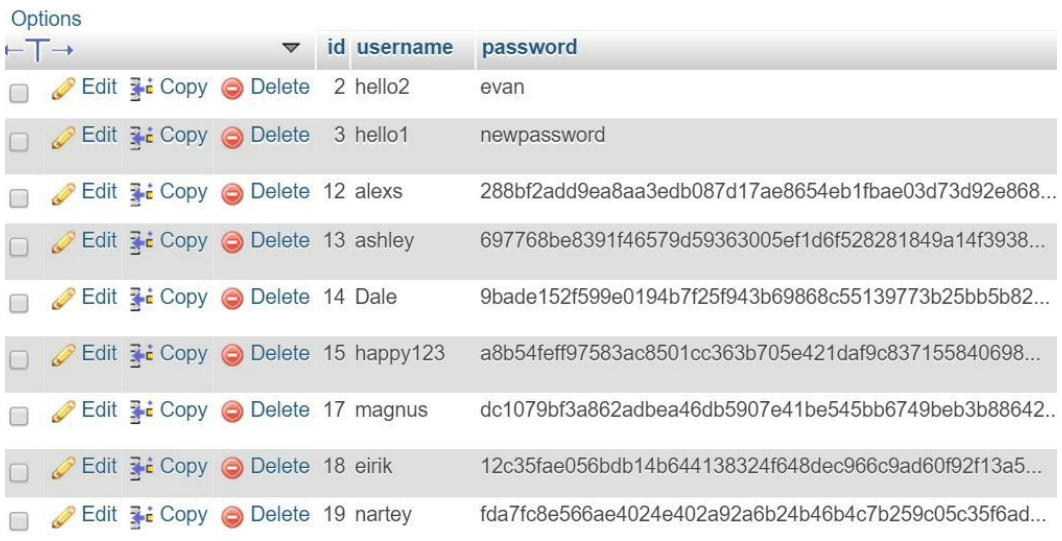
Figure 14: showing the section of the validation class handling user inputs

Optional task 2, Use a Database:



	id	user_id	name	comment
:dit Copy Delete	22	0	hello1	Yum
:dit Copy Delete	24	0	hello3	looks really good and fresh123
:dit Copy Delete	26	0	hello3	great
:dit Copy Delete	27	0	hello3	hi
:dit Copy Delete	28	0	hello3	hello45
:dit Copy Delete	29	0	eirik	htr
:dit Copy Delete	30	0	eirik	hi
:dit Copy Delete	31	0	eirik	nice
:dit Copy Delete	32	0	eirik	fantastic
:dit Copy Delete	33	0	eirik	hi
:dit Copy Delete	34	0	eirik	nice
:dit Copy Delete	35	18	eirik	hello

Figure 15 showing user data and comments in the database



Options				id	username	password
☐	Edit	Copy	Delete	2	hello2	evan
☐	Edit	Copy	Delete	3	hello1	newpassword
☐	Edit	Copy	Delete	12	alexs	288bf2add9ea8aa3edb087d17ae8654eb1fbae03d73d92e868...
☐	Edit	Copy	Delete	13	ashley	697768be8391f46579d59363005ef1d6f528281849a14f3938...
☐	Edit	Copy	Delete	14	Dale	9bade152f599e0194b7f25f943b69868c55139773b25bb5b82...
☐	Edit	Copy	Delete	15	happy123	a8b54feff97583ac8501cc363b705e421daf9c837155840698...
☐	Edit	Copy	Delete	17	magnus	dc1079bf3a862adbea46db5907e41be545bb6749beb3b88642..
☐	Edit	Copy	Delete	18	eirik	12c35fae056bdb14b644138324f648dec966c9ad60f92f13a5...
☐	Edit	Copy	Delete	19	nartey	fda7fc8e566ae4024e402a92a6b24b46b4c7b259c05c35f6ad...

Figure 16 showing user data and passwords in the database.

Discussion:

Task 1a, MVC Architecture Without Framework

This task requires that we rewrite the Tasty Recipes web site to make it follow the MVC architectural pattern and basic object-oriented design concepts. The following requirements had to be met.

In the first task , we chose to implement the MVC pattern in our website .We also implemented the singleton design pattern which ensured that we had an instance of the database call and use that instead of connecting to the database each time.We also created classes and functions on the object-oriented approach . We created a function that handled the link between business logic and user interface , which is similar to the Controller in the MVC pattern. We also created different namespaces for different classes to distinguish between the significance of ranges and features.

Task 2, Security

3. The second task was to improve the security of the website and we implemented (3) three security features which were
   a. Database security
   b. Password encryption and
   c. Cross Site Scripting .

Optional Task 2, Use a Database

This task required the use of a database to store comments and user data persistently on the server. All data must be in the database, do not store any data in plain files. There are no requirements on database design.

We used phpmyadmin to successfully implement this as well.