

# Labb rapport 3

Internet Applications, ID1354

Evan Saboo  
saboo@kth.se

2015-10-04

## 1 Introduktion

Målet med tredje labben är att lära sig använda objekt orienterad design för PHP server genom att använda MVC mönster och/eller PHP framework. Målet gick också ut på att lära sig undvika vissa grundläggande säkerhetshot i en webbapplikation. Man ska också lära sig att använda en databas till PHP server och förbättra PHP prestandan genom att använda cacheminne och bevarande anslutning.

Den första uppgiften går ut på att använda MVC mönster eller PHP framework för att få ett bra mönster arkitektur. I den andra uppgiften ska man förbättra säkerheten i hemsidan genom att implementera tre säkerhetsfunktioner i hemsidan. I den sista uppgiften ska man använda en databas till sin hemsida för att hantera alla användare och kommentarer.

Jag samarbetade med David Nartey ([dnartey@kth.se](mailto:dnartey@kth.se)) med uppgifterna.

## 2 Litteratur Studie

PHP funktionerna utfördes med hjälp av dessa källor:

- \* Jag gick igenom alla tre föreläsningarna om PHPs objekt orienterad design och undvikandet av säkerhetshot.

- \* Vi fick hjälp i med hemsidans säkerhet i handledningarna.

- \* [http://www.tutorialspoint.com/php/php\\_object\\_oriented.htm](http://www.tutorialspoint.com/php/php_object_oriented.htm) användes för att få en bättre förståelse av PHPs objekt orienterad design.

- \* I boken 'Programming the WWW 6<sup>th</sup> av Sebesta Robert W' gick jag igenom kapitel 11.7 några delar i kapitel 13:

- 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

- \* Jag kollade också i Google och Youtube videor för att kunna fixa små och stora problem, och för att få en överblick om hur man implementerar objekt orienterad design i hemsidan.

### 3 Metod

I den första uppgiften valde vi att implementera MVC mönstret i vår hemsida. Vi började med att kolla hur man skapade klasser och funktioner på objektorienterat sätt. Det fanns flera instruktioner i webben på hur man implementerade det vilket vi använde. Vi skapade alla nödvändiga funktioner t.ex. login och logout som fanns i User klassen. Vi skapade också funktioner som inte var kopplat till User men klassificerades ändå som nödvändiga funktioner t.ex. validering av inloggningen och uppkoppling till databasen. Vi skapade en funktion som hanterade kopplingen mellan logiska funktioner och användargränssnittet, vilket liknade Controller i MVC mönstret. Vi skapade också olika namnrymder till olika klasser för att kunna skilja mellan klassernas betydelse och funktioner.

Den andra uppgiften gick ut på att förbättra säkerheten i hemsidan. Vi implementerade tre säkerhetsfunktioner vilket var databas säkerhet, lösenordskryptering och Cross Site Scripting. Det ända vi gjorde i databasen var att vi skapade en användare som bara hade tillgång till fåtal behörigheter i inloggnings- och kommentarskategorierna i databasen. Hemsidan kunde bara koppla till den begränsade användaren i databasen.

Lösenordskrypteringen utfördes genom att skapa ett sträng av slumpvisa värde och kombinera det med det angivna lösenordet vid registrering av användare. När den var kombinerade utfördes en md5 kryptering på den för att skapa ett lösenord som blir svårt att dekryptera. Den skapade strängen sparades i databasen för att kunna använda den vid validering av inloggning.

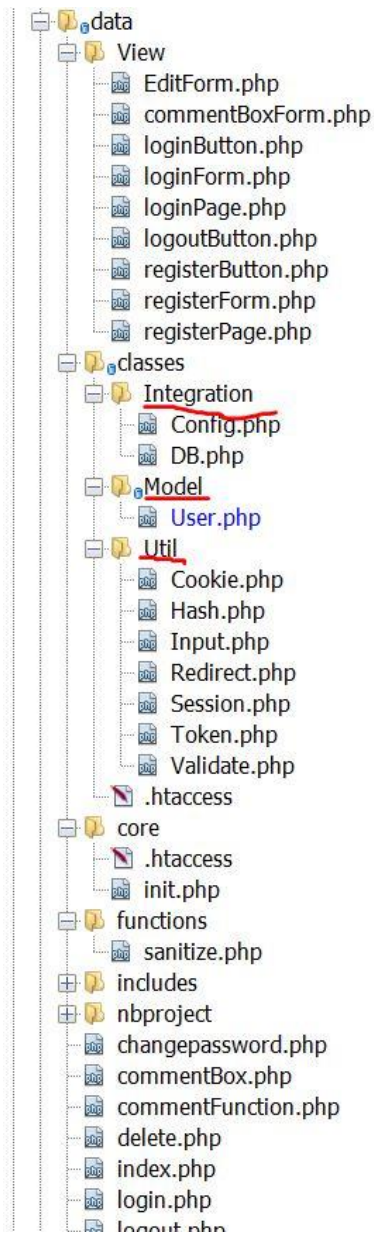
Det krävdes två olika funktioner för att undvika säkerhetshot mot Cross Site Scripting. Den första funktionen omvandlar ett värde till en html 'plain text'. Funktionen undviker att användaren kan skapa html, php eller java script kod i inmatningsfälten eftersom koden visas som vanlig text. Den andra funktionen undviker databas anropskoder i inmatningsfälten då användaren kan skicka farliga anrop till databasen om hemsidan inte har säkerhetsfunktioner mot kodinmatning. Funktionen gör att speciella tecken omvandlas till vanlig text för att förhindra farliga anrop till databasen.

Vi implementerade också en funktion som filtrerar bort alla tecken förutom bokstäver och siffror när en användare ska registrera skriva in sitt användarnamn.

Den sista uppgiften handlade om att använda en databas för att hantera användare och kommentarer. Vi hade redan skapat en databas i labb 2 gjorde att uppgiften inte krävde mycket tid att utföra. Det ända vi gjorde var att vi skrev om kopplingsfunktionen till databasen så att den använde PDO (Php Data Objects) istället för MySQL.

## 4 Resultat

### Uppgift 1 MVC mönster



Figur 1.1: klasshierarki

**MODEL**

```

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
        $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
public function find($user = null){
    if($user){
        $field = (is_numeric($user)) ? 'id' : 'username';
        $data = $this->_db->get('users', array($field, '=', $user));

        if($data->count()){
            $this->_data = $data->first();
            return true;
        }
    }
    return false;
}

//Login the user in
public function login($username = null, $password = null, $remember = false){ //checking if username and password are set

    if(!$username && !$password && $this->exists()){
        Session::put($this->_sessionName, $this->data()->id);
        $this->_isLoggedIn = true;
    }else{
        $user = $this->find($username);

        if($user){
            if($this->data()->password === Hash::make($password, $this->data()->salt)){
                Session::put($this->_sessionName, $this->data()->id);

                if($remember){
                    $hash = Hash::unique();
                    $hashCheck = $this->_db->get('users_session', array('user_id', '=', $this->data()->id));

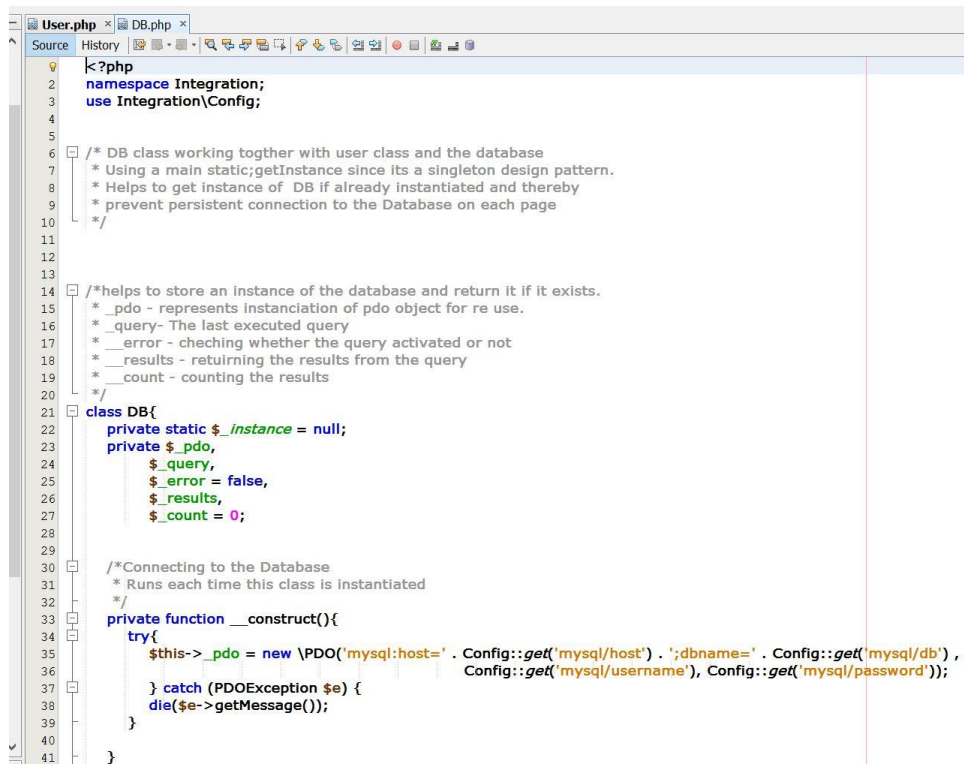
                    if(!$hashCheck->count()){
                        $this->_db->insert('users_session', array(
                            'user_id' => $this->data()->id,
                            'hash' => $hash
                        ));
                    } else {
                        $hash = $hashCheck->first()->hash;
                    }

                    Cookie::put($this->_cookieName, $hash, Config::get('remember/cookie_expiry'));
                }

                return true;
            }
        }
    }
    return false;
}

```

Figur 1.2: User klassen

**INTEGRATION**

```
1 <?php
2 namespace Integration;
3 use Integration\Config;
4
5
6 /* DB class working together with user class and the database
7  * Using a main static;getInstance since its a singleton design pattern.
8  * Helps to get instance of DB if already instantiated and thereby
9  * prevent persistent connection to the Database on each page
10  */
11
12
13
14 /*helps to store an instance of the database and return it if it exists.
15  * _pdo - represents instantiation of pdo object for re use.
16  * _query - The last executed query
17  * _error - checking whether the query activated or not
18  * _results - returning the results from the query
19  * _count - counting the results
20  */
21 class DB{
22     private static $_instance = null;
23     private $_pdo,
24             $_query,
25             $_error = false,
26             $_results,
27             $_count = 0;
28
29
30     /*Connecting to the Database
31     * Runs each time this class is instantiated
32     */
33     private function __construct(){
34         try{
35             $this->_pdo = new PDO('mysql:host=' . Config::get('mysql/host') . ';dbname=' . Config::get('mysql/db') ,
36                                 Config::get('mysql/username'), Config::get('mysql/password'));
37         } catch (PDOException $e) {
38             die($e->getMessage());
39         }
40     }
41 }
```

```

/*Checking if we've already instantiated 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 database
 */
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 queries
public function action($action, $table, $where = array()){
    if(count($where) == 3){
        $operators = array ('=', '>', '<', '>=', '<=');

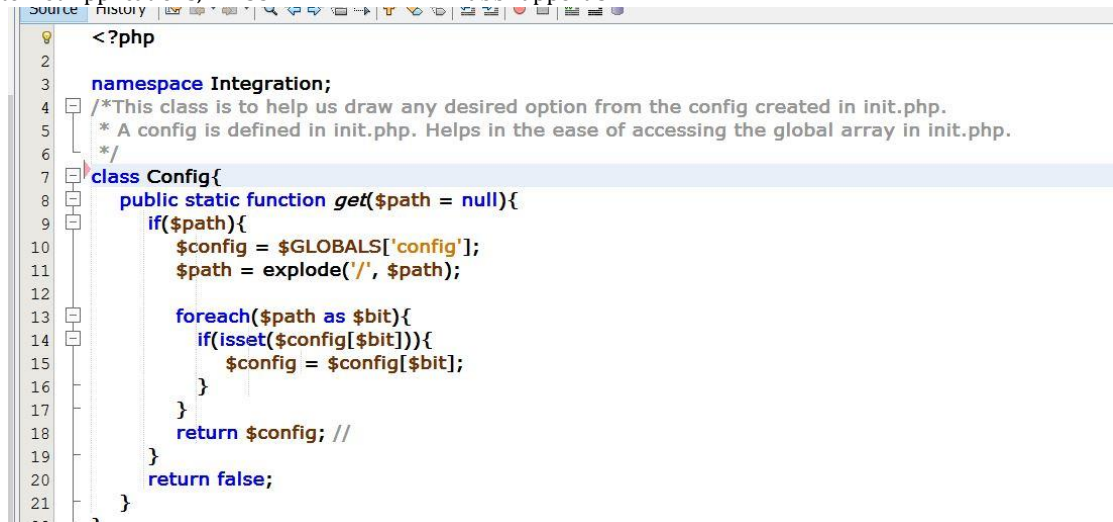
        $field = $where[0];
        $operator = $where[1];
        $value = $where[2];

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

```

Figur 1.3: Databas klassen





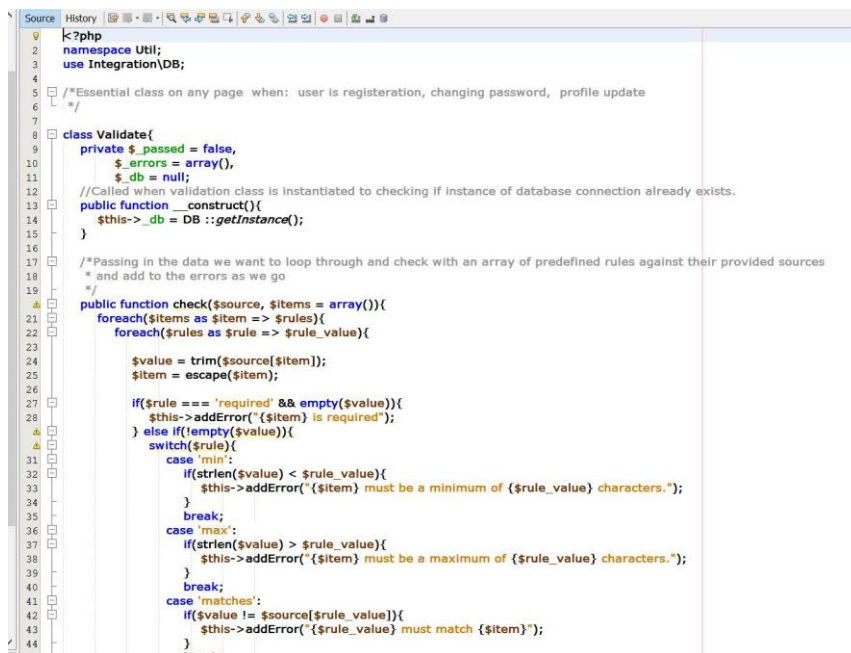
```

1 <?php
2
3 namespace Integration;
4 /*This class is to help us draw any desired option from the config created in init.php.
5  * A config is defined in init.php. Helps in the ease of accessing the global array in init.php.
6  */
7 class Config{
8     public static function get($path = null){
9         if($path){
10             $config = $GLOBALS['config'];
11             $path = explode('/', $path);
12
13             foreach($path as $bit){
14                 if(isset($config[$bit])){
15                     $config = $config[$bit];
16                 }
17             }
18             return $config; //
19         }
20         return false;
21     }
22 }

```

Figur 1.4 : Config klassen

## UTIL

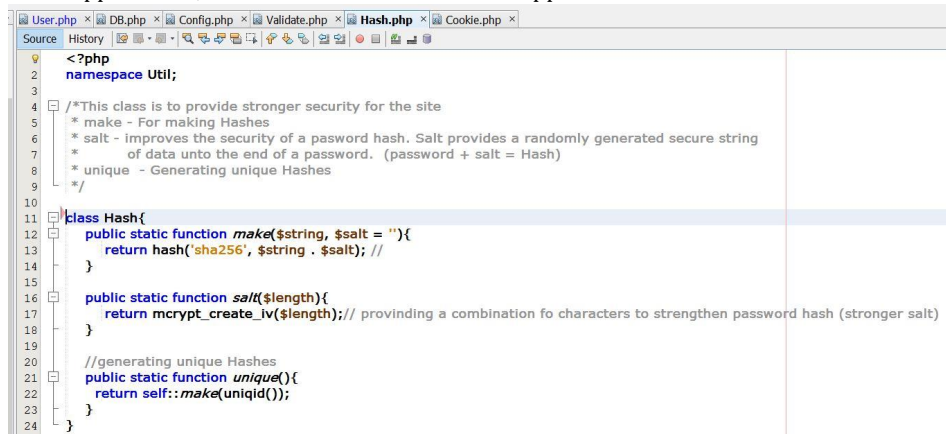


```

1 <?php
2 namespace Util;
3 use Integration\DB;
4
5 /*Essential class on any page when: user is registration, changing password, profile update
6  */
7
8 class Validate{
9     private $_passed = false,
10         $_errors = array(),
11         $_db = null;
12     //Called when validation class is instantiated to checking if instance of database connection already exists.
13     public function __construct(){
14         $this->_db = DB ::getInstance();
15     }
16
17     /*Passing in the data we want to loop through and check with an array of predefined rules against their provided sources
18     * and add to the errors as we go
19     */
20     public function check($source, $items = array()){
21         foreach($items as $item => $rules){
22             foreach($rules as $rule => $rule_value){
23
24                 $value = trim($source[$item]);
25                 $item = escape($item);
26
27                 if($rule == 'required' && empty($value)){
28                     $this->addError("$item is required");
29                 } else if(empty($value)){
30                     switch($rule){
31                         case 'min':
32                             if(strlen($value) < $rule_value){
33                                 $this->addError("$item must be a minimum of {$rule_value} characters.");
34                             }
35                             break;
36                         case 'max':
37                             if(strlen($value) > $rule_value){
38                                 $this->addError("$item must be a maximum of {$rule_value} characters.");
39                             }
40                             break;
41                         case 'matches':
42                             if($value != $source[$rule_value]){
43                                 $this->addError("$rule_value must match {$item}");
44                             }
45                             break;
46                     }
47                 }
48             }
49         }
50     }
51 }

```

Figur 1.5: Valideringsklassen

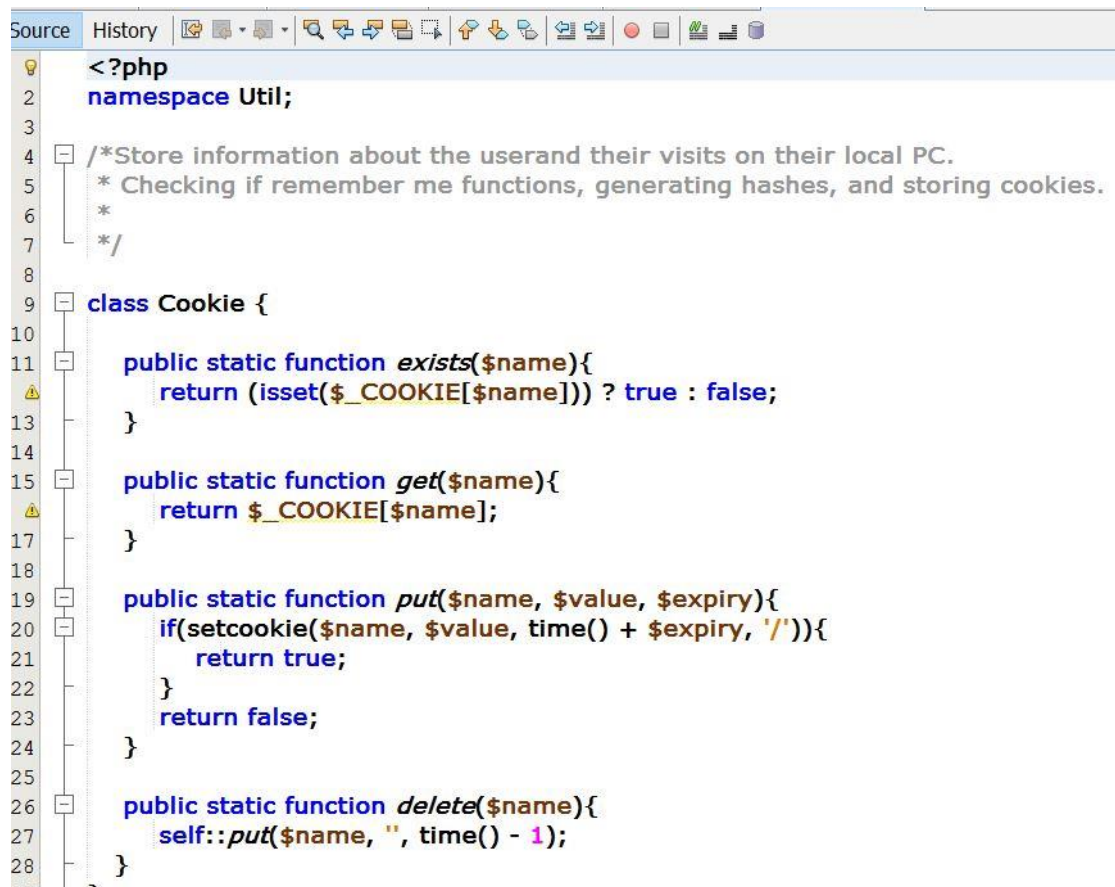


```

1  <?php
2  namespace Util;
3
4  /*This class is to provide stronger security for the site
5   * make - For making Hashes
6   * salt - improves the security of a password hash. Salt provides a randomly generated secure string
7   * of data unto the end of a password. (password + salt = Hash)
8   * unique - Generating unique Hashes
9   */
10
11 class Hash{
12     public static function make($string, $salt = ""){
13         return hash('sha256', $string . $salt); //
14     }
15
16     public static function salt($length){
17         return mcrypt_create_iv($length); // providing a combination fo characters to strengthen password hash (stronger salt)
18     }
19
20     //generating unique Hashes
21     public static function unique(){
22         return self::make(uniqid());
23     }
24 }

```

Figur 1.6: Hash klassen



```

1  <?php
2  namespace Util;
3
4  /*Store information about the user and their visits on their local PC.
5   * Checking if remember me functions, generating hashes, and storing cookies.
6   *
7   */
8
9  class Cookie {
10
11     public static function exists($name){
12         return (isset($_COOKIE[$name])) ? true : false;
13     }
14
15     public static function get($name){
16         return $_COOKIE[$name];
17     }
18
19     public static function put($name, $value, $expiry){
20         if(setcookie($name, $value, time() + $expiry, '/')){
21             return true;
22         }
23         return false;
24     }
25
26     public static function delete($name){
27         self::put($name, "", time() - 1);
28     }
29 }

```

Figur 1.7: Cookie klassen

```
<?php
namespace Util;

/* Checking existence 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;
    }

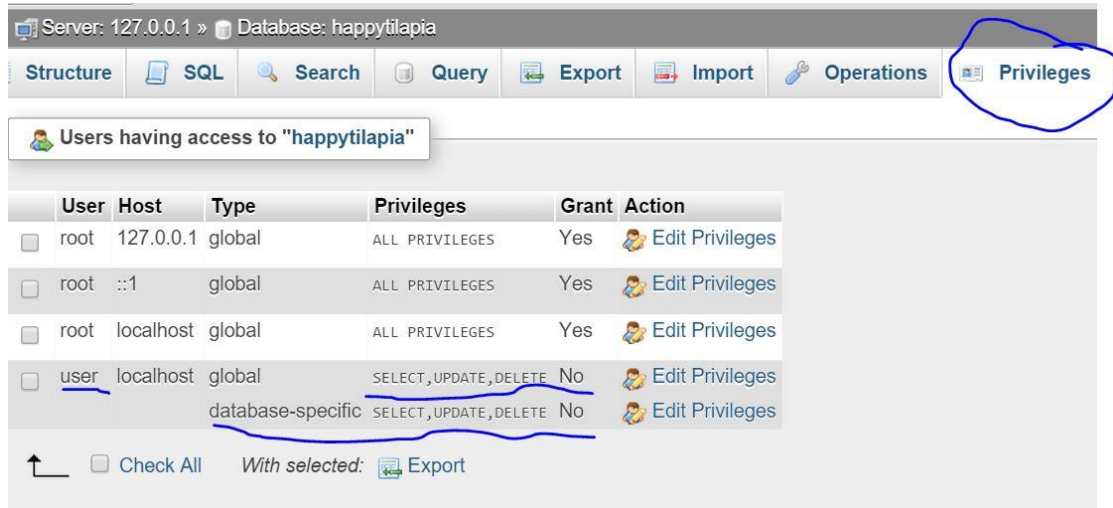
    public static function get($name){
        return $_SESSION[$name];
    }

    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);
        }
    }
}
```

Figur 1.8: Session klassen

***Databas säkerhet:***



	User	Host	Type	Privileges	Grant	Action
<input type="checkbox"/>	root	127.0.0.1	global	ALL PRIVILEGES	Yes	Edit Privileges
<input type="checkbox"/>	root	:::1	global	ALL PRIVILEGES	Yes	Edit Privileges
<input type="checkbox"/>	root	localhost	global	ALL PRIVILEGES	Yes	Edit Privileges
<input type="checkbox"/>	<u>user</u>	localhost	global	<u>SELECT,UPDATE,DELETE</u>	No	Edit Privileges
			database-specific	<u>SELECT,UPDATE,DELETE</u>	No	Edit Privileges

↑ ☐ Check All With selected: Export

Figur 2.1 visar visat begränsade användarrättigheter i databasen.

**Lösenordskryptering:**

```

1  <?php
2  namespace Util;
3
4  /*This class is to provide stronger security for the site
5   * make - For making Hashes
6   * salt - improves the security of a password hash. Salt provides a randomly generated secure string
7   *         of data unto the end of a password. (password + salt = Hash)
8   * unique - Generating unique Hashes
9   */
10
11 class Hash{
12     public static function make($string, $salt = ''){
13         return hash('sha256', $string . $salt); //
14     }
15
16     public static function salt($length){
17         return mcrypt_create_iv($length); // providing a combination fo characters to strengthen password hash (stronger salt)
18     }
19
20     //generating unique Hashes
21     public static function unique(){
22         return self::make(uniqid());
23     }
24 }

```

Figur 2.1: Hash klassen

	id	username	password	email	salt	name
Copy Delete	2	hello2	evan	saboo@kth.se	098f6bcd4621d373cade4e832627b4f6	
Copy Delete	3	hello1	newpassword	davidjnarthey@gmail.com	098f6bcd4621d373cade4e832627b4f6	Doris Narthey
Copy Delete	12	alexs	288bf2add9ea8aa3edb087d17ae8654eb1fbae03d73d92e868...		äy"PpiçÖ7i(cZ²<â3&fû¥/×)VB%ã	alex
Copy Delete	13	ashley	697768be8391f46579d59363005ef1d6f528281849a14f3938...		—Gö—4tô:ÜQ:â¿&%*Ä²27Û8yU	Ashley Garrett
Copy Delete	14	Dale	9bade152f599e0194b7f25f943b69868c55139773b25bb5b82...		;xv"øZ"%ø¿  Ä~%4Bûñ5© SJ¥Pöê	Dale Garnegie
Copy Delete	15	happy123	a8b54feff9f7583ac8501cc363b705e421daf9c837155840698...		µ7dEFü™yAD?i\$_ñu×@jplcF&_;	Dale Garnegie
Copy Delete	17	magnus	dc1079bf3a862adbea46db5907e41be545bb6749beb3b88642...		Cûv;IYÜZâeXð:†R¿Ö»jØ	Magnus Narthey
Copy Delete	18	eirik	12c35fae056bdb14b644138324f648dec966c9ad60f92f13a5...		Ê:†óYçp°ÖÊ)ÉAJRûHsBE³×%ø8%ø	Eirik
Copy Delete	19	narthey	fda7fc8e566ae4024e402a92a6b24b46b4c7b259c05c35f6ad...		3*f *ÿ.Yulâé' ~KÜ,(ÉA~;J	David Narthey

Figur 2.2 visar lösenord och salt i databasen.

```

<?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(unidid()));
    }

    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;
    }
}

```

Figur 2.3 visar Cross Site Request Forgery:

Figur 2.4 och 2.5 visar hur funktioner mot Cross Site scripting implementeras:

```

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

```




















Figur 2.4 visar sanitize klassen.



```
// 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;
}
}
```

Figur 2.5 visar koden som hanterar valideringen av användarens inskrivning.




























**Valfri uppgift 2, Använda en databas:**

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

Figur 3.1 visar användarens id och kommentar.



Options

				id	username	password
<input type="checkbox"/>				2	hello2	evan
<input type="checkbox"/>				3	hello1	newpassword
<input type="checkbox"/>				12	alexs	288bf2add9ea8aa3edb087d17ae8654eb1fbae03d73d92e868...
<input type="checkbox"/>				13	ashley	697768be8391f46579d59363005ef1d6f528281849a14f3938...
<input type="checkbox"/>				14	Dale	9bade152f599e0194b7f25f943b69868c55139773b25bb5b82...
<input type="checkbox"/>				15	happy123	a8b54feff97583ac8501cc363b705e421daf9c837155840698...
<input type="checkbox"/>				17	magnus	dc1079bf3a862adbea46db5907e41be545bb6749beb3b88642..
<input type="checkbox"/>				18	eirik	12c35fae056bdb14b644138324f648dec966c9ad60f92f13a5...
<input type="checkbox"/>				19	nartey	fda7fc8e566ae4024e402a92a6b24b46b4c7b259c05c35f6ad...

Figur 3.2 visar användaren lösenord som har krypterats till md5 kod.

## 5 Diskussion

### Uppgift 1a, MVC arkitektur utan Framework

Denna uppgift krävde att vi skulle skriva om Tasty recept hemsidan för att det ska följa MVC mönstret eller framework och implementera grundläggande objektorienterade designkoncept.

Vi valde att genomföra MVC mönstret på vår hemsida eftersom vi redan hade läst om MVC i objektorienterad design kursen. Vi implementerade också singleton mönstret som såg till att vi hade en instans av databasanropen istället för att koppla till databasen varje gång. Vi skapade också klasser och funktioner som fungerar på ett objektorienterat sätt. Vi skapade en funktion som hanterade kopplingen mellan logik och användargränssnitt, vilket liknar Controller i MVC mönstret. Vi skapade också olika namnrymder för olika klasser för att skilja mellan betydelsen av klasser och funktioner.

### Uppgift 2, Säkerhet

Den andra uppgiften gick ut på att förbättra säkerheten vilket vi lyckades göra genom att implementera tre säkerhetsfunktioner. Det var Databassäkerhet, lösenord kryptering och Cross Site Scripting.

### Valfriuppgift 2, använda en databas

Denna uppgift krävde användning av en databas för att lagra ständigt kommentarer och användardata på servern. All data måste vara i databasen och inte lagras i filer. Det finns inga krav på databasdesign. Vi använde phpmyadmin för att genomföra uppgiften.

## 6 Kommentarer om kursen

Vi har spenderat minst 4 timmar varje dag åt uppgifterna vilket blev totalt 48 timmar inklusive föreläsningarna och handlerdingarna.