Backend

Klasse Cookie

```
1
2
3  <?php
4
5  class Cookie{
6
7      public static function exists($name){
8      return(isset($_COOKIE[$name])) ? true : false;
9    }
10
11   public static function get($name){
12     return $_COOKIE[$name];
13   }
14
15   public static function put($name, $value, $expiry){
16       if(setcookie($name, $value, time() + $expiry, '/')) {
17         return true;
18       }
19     return false;
20   }
21
22   public static function delete($name){
23     self::put($name, '', time() - 1);
24   }
25 }
26
27 ?>
```

Klasse Config

```
1  <?php
2  class Config {
3    public static function get($path = null){
4      if($path){
5        $config = $GLOBALS['config'];
6        $path = explode('/',$path);
7        foreach ($path as $bit){
8          if(isset($config[$bit])){
9            $config = $config[$bit];
10                 }
11       }
12       return $config;
13     }
14     return false;
15   }
16 }
17 ?>
```

Klasse DB

```php
<?php
class DB {
    private static $_instance = null;
    private $_pdo,
            $_query,
            $_error = false,
            $_results,
            $_count = 0;

    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 $e){
      echo 'Fehler';
            die($e->getMessage());
        }
    }

  private function __clone() {
  }

  public static function getInstance(){
        if(!isset(self::$_instance)){
            self::$_instance = new self();
        }
        return self::$_instance;
    }

  public function query($sql,$params = array()){
    $this->_error = false;
    if ($this->_query = $this ->_pdo->prepare($sql)){
      $x = 1;
      // echo 'Success';
      if(count($params)){
        foreach ($params as $param){
            $this->_query->bindValue($x,$param);
          $x++;
        }
      }

      if($this->_query->execute()){
        //echo'Success';
        $this->_results = $this->_query->FetchAll(PDO::FETCH_OBJ);
        $this->_count = $this->_query->rowCount();
                }else{
            $this->_error = true;
                }
    }
    return $this;
  }


  public function action($action,$table,$where = array()){
    if(count($where) === 3){
```

```
55      $operators = array('=','>','<','>=','<=');
56
57      $field    =$where[0];
58      $operator   =$where[1];
59      $value    =$where[2];
60
61      if(in_array($operator,$operators)){
62        $sql = "{$action} FROM {$table} WHERE {$field} {$operator} ?";
63        if(!$this->query($sql, array($value))->error()){
64          return $this;
65        }
66      }
67    }
68    return false;
69  }
70
71  public function get($table,$where){
72    return $this->action('SELECT *', $table, $where);
73  }
74
75  public function delete ($table,$where){
76    return $this->action('DELETE', $table, $where);
77  }
78
79  public function insert ($table, $fields = array()){
80    if(count($fields)){
81      $keys = array_keys($fields);
82      $values = "";
83      $x = 1;
84
85      foreach($fields as $field){
86        $values .= '?';
87        if($x < count($fields)){
88          $values .= ', ';
89        }
90        $x++;
91      }
92
93      $sql = "INSERT INTO {$table} (`" . implode('` , `', $keys) . "`)
    VALUES ({$values})";
94
95      if($this->query($sql, $fields)->error()){
96        return true;
97      }
98    }
99    return false;
100 }
101
102 public function update($table, $id, $fields){
103   $set = '';
104   $x = 1;
105
106   foreach($fields as $name => $value){
107     $set .= "{$name} = ?";
108     if($x < count($fields)){
109       $set .= ', ';
```

```
      }
      $x++;
    }


    $sql = "UPDATE {$table} SET {$set} WHERE idUser = {$id}";

    if(!$this->query($sql, $fields)->error()){
      return true;
    }
    return false;
  }

  public function results(){
    return $this->_results;
  }

  public function first(){
    return $this->results()[0];
  }

  public function error(){
    return $this->_error;
  }

   public function count() {
        return $this->_count;
    }

}
?>
```

Klasse Hash

```
<?php

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

  public static function salt($length){
    return random_bytes($length);
  }

  public static function unique(){
    return self::make(uniqid());
  }

}

?>
```

Klasse Input

```php
<?php
class Input{

  public static function exists($type = 'post'){
    switch ($type){
      case 'post':
        return (!empty($_POST)) ? true : false;
      break;
      case 'get':
        return (!empty($_GET)) ? true : false;
      break;
      default:
        return false;
      break;
    }
  }

  public static function get ($item){
    if(isset ($_POST[$item])){
      return $_POST[$item];
    } else if(isset($_GET[$item])){
      return $_GET[$item];
    }
    return '';
  }
}
?>
```

Klasse Redirect

```php
<?php
class Redirect{
  public static function to($location= null){
    if($location){
      if(is_numeric($location)){
        switch($location){
          case 404:
            header('HTTP/1.0 404 NOT FOUND');
            include 'includes/errors/404.php';
            exit();
          break;

        }
      }

      header('Location: ' .$location);
      exit();
    }
  }
}
```

Klasse Session

```php
<?php
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);
    }
  }
}
?>
```

Klasse Token

```php
<?php
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;
  }
}

?>
```

Klasse User

```php
<?php

class User{
  private $_db,
      $_data,
      $_sessionName,
      $_cookieName,
      $_isLoggedIn;

  public function __construct($user = null){
    $this->_db = DB::getInstance();

    $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 {
        }
      }
    } else{
      $this->find($user);
    }
  }
  public function update($fields = array(), $id = null){
      $id = $this->data()->idUser;

    if(!$this->_db->update('user',$id, $fields)){
      throw new Exception('Ein Fehler ist aufgetreten!');
    }
  }
  public function delete (){
    $id = $this->data()->idUser;
    if(!$this->_db->delete('user', array('idUser', '=', $id))){
      throw new Exception('Ein Fehler ist aufgetreten!');
    }
  }


  public function create($fields = array()){
    if($this->_db->insert('user', $fields)){
      throw new Exception('Fehler beim erstellen einer Account.');
    }
  }

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

```

```
55        if($data->count()){
56          $this->_data = $data->first();
57          return true;
58
59        }
60      }
61      return false;
62    }
63
64    public function login ($username = null, $password = null, $remember =
      false){
65
66      if(!$username && !$password && $this->exists()){
67        Session::put($this->_sessionName, $this->data()->idUser);
68      }else{
69        $user = $this->find($username);
70        if($user){
71          if($this->data()->Passwort === Hash::make($password, $this->data()
      ->salt)){
72            Session ::put($this->_sessionName, $this->data()->idUser);
73
74            if($remember){
75              $hash = Hash::unique();
76              $hashCheck = $this->_db->get('user_session', array('user_id', '
      =', $this->data()->idUser));
77              if(!$hashCheck->count()){
78                $this->_db->insert('user_session',array(
79                  'user_id' => $this->data()->idUser,
80                  'hash' => $hash
81                ));
82              }else{
83                $hash = $hashCheck->first()->hash;
84              }
85              Cookie::put($this->_cookieName, $hash, Config::get('remember/
      cookie_expiry'));
86            }
87
88            return true;
89          }
90        }
91      }
92
93      return false;
94    }
95
96    public function exists(){
97      return(!empty($this->_data)) ? true : false;
98    }
99
100
101   public function logout(){
102     $this->_db->delete('user_session', array('user_id', '=', $this->data()
      ->idUser));
103     Session::delete($this->_sessionName);
104     Cookie::delete($this->_cookieName);
105   }
```

```
106
107   public function data (){
108     return $this->_data;
109   }
110
111   public function isLoggedIn(){
112     return $this->_isLoggedIn;
113   }
114 }
115 ?>
```

Klasse Validate

```
1  <?php
2
3  class Validate {
4    private $_passed = false,
5        $_errors = array(),
6        $_db = null;
7
8    public function __construct(){
9      $this->_db = DB ::getInstance();
10   }
11
12   public function check ($source, $items = array()){
13     foreach($items as $item => $rules){
14       foreach($rules as $rule => $rule_value){
15       $value = trim ($source[$item]);
16         $item = escape($item);
17
18         if($rule === 'required' && empty($value)){
19           $this->addError("{$item} ist erforderlich");
20         } else if (!empty($value)){
21           switch ($rule) {
22             case 'min':
23               if(strlen($value) < $rule_value){
24                 $this->addError("{$item} muss aus mindestens {$rule_value}
     zeichen bestehen.");
25               }
26               break;
27             case 'max':
28               if(strlen($value) > $rule_value){
29                 $this->addError("{$item} muss aus mindestens {$rule_value}
     zeichen bestehen.");
30               }
31               break;
32             case 'matches':
33               if($value != $source[$rule_value]){
34                 $this->addError("{$rule_value} muss passen {$item}");
35
36               }
37
38               break;
39             case 'unique':
```

```
                $check = $this->_db->get($rule_value, array($item, '=',
    $value));
                if($check->count()){
                  $this->addError("{$item} existiert bereits.");

                }
                break;
            }

          }
        }
    }
    if(empty($this->_errors)){
      $this->_passed = true;

    }
    return $this;
  }
  private function addError($error){
    $this->_errors[]=$error;

  }

  public function errors (){

    return $this->_errors;
  }

  public function passed(){
    return $this->_passed;
  }
}
?>
```

Konfigurationsdatei

```
<?php
session_start();

$GLOBALS['config'] = array(
  'mysql' => array(
    'host' =>'127.0.0.1',
    'username' => 'root',
    'password' => '',
    'db' => 'feedbee'
  ),
  'remember' => array(
    'cookie_name' => 'hash',
    'cookie_expiry' => 604800
  ),
  'session' => array(
    'session_name' => 'user',
    'token_name' => 'token'
  )
);
```

```
20
21 spl_autoload_register(function($class){
22   require_once 'classes/'.$class.'.php';
23
24 });
25
26 require_once 'functions/sanitize.php';
27
28 if(Cookie::exists(Config::get('remember/cookie_name')) && !Session::exists(
     Config::get('session/session_name'))){
29   $hash = Cookie::get(Config::get('remember/cookie_name'));
30   $hashCheck = DB::getInstance()->get('user_session', array('hash', '=',
     $hash));
31
32   if($hashCheck->count()){
33     $user = new User($hashCheck->first()->idUser);
34     $user->login();
35   }
36 }
37 ?>
```

Funktion für die Ausgabe

```
1 <?php
2 function escape($string){
3   return htmlentities($string, ENT_QUOTES,'UTF-8');
4 }
5
6 ?>
```