Ett helt litet projekt: "`testa.roboro.se`"

Beskrivning

Vi ska bygga en webbapplikation för att skapa och använda glostester. Det kommer inte att finnas någon inloggning, utan vem som helst kan skapa och göra tester. Det typiska användningsfallet är att någon som ska lära sig glosor skapar ett test och sedan gör det om och om igen. Man kan tänka sig framtida utvecklingar där en lärare skapar tester som eleverna sedan gör, och där läraren kan se elevernas resultat.

Krav

- Det ska vara enkelt att skapa test
- Vem som helst ska kunna skapa test
- Man ska kunna editera ett skapat test
- Man ska inte kunna sabotera någon annans test
- Applikationen ska kunna driftsättas på olika operativsystem, med Apache och PHP
- Det ska vara enkelt att säkerhetskopiera databasen
- Man ska kunna byta databasmotor

Tekniska förutsättningar för projektet

- Apache
- PHP
- sqlite

Steg

1. Skapa en katalog med underkataloger "planning", "application", "www".
2. Formulera och skriv ner idén.
3. Skissa gränssnitt, på papper eller exempelvis med Paint.
4. Objektmodellera. Identifiera substantiv, gör UML.
5. Skapa underkatalogerna "model", "view" och "controller" i katalogen "application".
6. Skriv `database.php`.
7. Skriv `index.php` och `.htaccess`, "routern".
8. Skriv controllers.
9. Skriv motsvarande vyer.
10. Jobba med design.

I praktiken så hoppar man mellan steg 6-9, och itererar fram en bra lösning. Det är vanligt att man kommer på att man tänkt fel och måste gå tillbaka.

1. Katalogstrukturen

Skapa en katalog som heter "`testa`", skapa sedan underkatalogerna `planning`, `application` och `www`. I `planning` kommer vi att spara alla filer som har med planering av projektet att göra. I `application` kommer logiken och själva databasen att ligga. Den vill vi inte lägga ut direkt på nätet, så därför lägger vi den inte i `www` -katalogen. I `www`-katalogen kommer vi att lägga MVC-routern och annat som ska synas på webben, till exempel CSS och bilder.

2. Planering

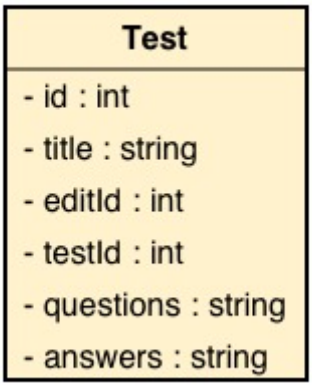
Skriv ner hur det är tänkt att man ska använda systemet och spara i en textfil. I det här fallet ska det finnas två vyer, en för att skapa och ändra tester och en för att göra testerna. Om en användare surfar in direkt på `testa.roboro.se` så ska det skapas ett tomt test, och editeringsvyn visas. På editeringsvyn ska det finnas en hemlig länk som man kan skicka till dem som ska använda testet. Det finns också en hemlig länk till själva editeringsvyn, så att man kan ändra på testet i efterhand.

3. Skissa på interfacet

Rita enkla bilder för varje vy som visar hur de kan användas.

4. Objektmodellera

Det centrala i systemet är tester som har en titel, två hemliga länkar och ett antal frågor. Vi väljer att nöja oss med en tabell, `Test`, med kolumnerna `id`, `title`, `editId`, `testId`, `questions` och `answers`. Se följande mycket enkla UML-diagram:



5. Skapa MVC-kataloger

Gå in i `application`-katalogen och skapa underkatalogerna `model`, `view` och `controller`. I `model` kommer databaskod att finnas, i `view` lägger vi koden för alla vyer och i `controller` hamnar logiken bakom varje vy. Vi separerar alltså vyn från logiken, och logiken från databasen.

6. `database.php`

Gå in i katalogen `model` och skriv filen `database.php`. När den körs ska den skapa en databas om det inte finns någon, och förbereda kopplingen till databasen. Det här är det enda stället där man ser vilken sorts databas som används. Att den skapas automatiskt gör att det blir enkelt att sjösätta applikationen.

Databasen är byggd som en klass, för att göra koden snyggare. Vi har också använt designmönstret singleton för att se till att det bara skapas en koppling till databasen i taget.

application/model/database.php

```
<?php
  class Database
  {
    private $db;
    private static $instance = null;

    public static function get_instance()
    {
      if (self::$instance == null)
        self::$instance = new Database();

      return self::$instance;
    }

    private function Database()
    {
      $db_file = 'database/database.sqlite';

      if (! file_exists($db_file))
      {
        $this->db = new PDO('sqlite:'. $db_file);
```

```
      $this->db->query('CREATE TABLE Test
                        (id INTEGER PRIMARY KEY,
                         editId INTEGER,
                         testId INTEGER,
                         title TEXT,
                         questions TEXT,
                         answers TEXT)');
    }
    else
    {
      $this->db = new PDO('sqlite:'. $db_file);
    }
  }

  function update_test($editId, $title, $questions, $answers)
  {
    $statement =
      $this->db->prepare('UPDATE Test
                            set title = ?,
                                questions = ?,
                                answers = ?
                          WHERE editId = ?');

    $statement->execute(array($title, $questions, $answers, $editId));
  }

  function get_test_by_editId($editId)
  {
    $statement =
      $this->db->prepare('SELECT title, questions, answers, testId, editId
                          FROM Test
                          WHERE editId = ?');

    $statement->execute(array($editId));

    return $statement->fetch();
  }

  function get_test_by_testId($testId)
  {
    $statement =
      $this->db->prepare('SELECT title, questions, answers
                          FROM Test
                          WHERE testId = ?');

    $statement->execute(array($testId));

    return $statement->fetch();
  }

  function create_test()
  {
    do
    {
      $editId = rand(10000, 99999);
    } while ($this->get_test_by_editId($editId));

    do
    {
      $testId = rand(10000, 99999);
    } while ($this->get_test_by_testId($testId));

    $statement =
      $this->db->prepare('INSERT INTO Test (editId, testId, title, questions, answers)
                          VALUES (?, ?, "", "", "")');

    $statement->execute(array($editId, $testId));

    return $editId;
  }
}
?>
```

7. Routern

För att få snygga URL:er måste vi utnyttja funktionalitet i webbservern. Normalt ser URL:er till php-skript ut så här:

```
http://testa.roboro.se/index.php?action=edit&test=123
```

Men vi vill att de ska se ut så här:

```
http://testa.roboro.se/edit/123
```

I Apache kan man lösa det med en funktion som kallas *rewrite*. Vi skriver en fil som heter `.htaccess` och sparar den i `www`-katalogen. Den innehåller inställningar till webservern som gäller i den katalogen och eventuella underkataloger. Följande exempel är ganska generellt. Om URL:en pekar på en fil eller katalog som finns så skrivs den inte om, allt annat skickas till `index.php`, routern.

www/.htaccess

```
RewriteEngine on
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule ^(.*)$ index.php? [L,QSA]
```

www/index.php

```
<?php
  chdir('../application');

  include 'model/database.php';

  foreach (glob('controller/*.php') as $controller)
    include $controller;

  $query = $_SERVER['QUERY_STRING'];

  $routes = array
  (
    '/^$/'            => function($m) { control_new(); },
    '/^edit\/(\d+)$/' => function($m) { control_edit($m[1]); },
    '/^(\d+)$/'       => function($m) { control_test($m[1]); },
    '/.*/'            => function($m) { control_error('Hittade inte sidan.'); }
  );

  foreach ($routes as $regex => $function)
  {
    if (preg_match($regex, $query, $matches))
    {
      $function($matches);
      break;
    }
  }
?>
```

8. Controllers

Varje controller är en funktion som hämtar allt som behövs för att rendera en vy, och kallar på vyn. Controllern vet alltså inget om hur datat lagras eller hur vyn ser ut.

application/controller/new.php

```
<?php
  function control_new()
  {
    $db = Database::get_instance();
    $editId = $db->create_test();
    header('Location: /edit/'. $editId);
  }
?>
```

application/controller/edit.php

```
<?php
  function control_edit($editId)
  {
    $db = Database::get_instance();

    if ($db->get_test_by_editId($editId))
    {
      if (isset($_REQUEST['submit']))
      {
        $db->update_test($editId,
                         $_REQUEST['title'],
                         $_REQUEST['questions'],
                         $_REQUEST['answers']);
      }
    }
    else
    {
      control_error("Hittar inte testet!");
      return;
    }

    $test = $db->get_test_by_editId($editId);

    $title      = preg_replace('/</', '&lt;', $test['title']);
    $questions  = preg_replace('/</', '&lt;', $test['questions']);
    $answers    = preg_replace('/</', '&lt;', $test['answers']);
    $testId     = $test['testId'];
    $stylesheet = "edit";

    include 'view/header.php.html';
    include 'view/edit.php.html';
    include 'view/footer.php.html';
  }
?>
```

application/controller/test.php

```
<?php
  function control_test($testId)
  {
    $db = Database::get_instance();

    $test = $db->get_test_by_testId($testId);

    if ($test)
    {
      $title      = preg_replace('/</', '&lt;', $test['title']);
      $questions  = explode("\n", preg_replace('/</', '&lt;', $test['questions']));
      $answers    = explode("\n", preg_replace('/</', '&lt;', $test['answers']));
      $stylesheet = 'test';

      if (isset($_REQUEST['answers']))
        $user_answers = preg_replace('/</', '&lt;', $_REQUEST['answers']);

      foreach ($questions as $index => $question)
      {
        $test_questions[$index]['question'] = $questions[$index];

        if (isset($user_answers) && isset($user_answers[$index]))
        {
          $test_questions[$index]['user_answer'] = $user_answers[$index];
          $test_questions[$index]['is_correct'] =
            trim($user_answers[$index]) == trim($answers[$index]);
        }
        else
        {
          $test_questions[$index]['user_answer'] = '';
          $test_questions[$index]['is_correct'] = true;
        }
      }

      include 'view/header.php.html';
      include 'view/test.php.html';
      include 'view/footer.php.html';
    }
    else
    {
      control_error('Hittade inte testet!');
    }
  }
?>
```

application/controller/error.php

```
<?php
  function control_error($message)
  {
    $title      = "Error i modermodemet";
    $stylesheet = "error";

    include 'view/header.php.html';
    include 'view/error.php.html';
    include 'view/footer.php.html';
  }
?>
```

9. Vyer

Vyerna är filer som innehåller html, css och lite php. De får inte innehålla logik som styr hur applikationen fungerar.

application/view/header.php.html

```
<!DOCTYPE html>
<html lang="sv">
  <head>
    <meta charset="utf-8" />
    <title><?php print $title ?></title>
  </head>
  <body>
```

application/view/footer.php.html

```
  </body>
</html>
```

application/view/test.php.html

```
    <h1><?php print $title ?></h1>
    <form action="" method="POST">
      <table>
<?php foreach ($test_questions as $test_question) { ?>
        <tr>
          <td>
            <?php print $test_question['question'] ?>
          </td>
          <td>
            <input type="text"
                   name="answers[]"
                   value="<?php print $test_question['user_answer'] ?>" />
            <?php if (!$test_question['is_correct']) { print "FEL!"; } ?>
          </td>
        </tr>
<? } ?>
      </table>
      <input type="submit" name="submit" value="Rätta!" />
    </form>
```

application/view/edit.php.html

```
    <form action="" method="POST">
      <h1>Titel: <input name="title" type="text" value="<?php print $title ?>" /></h1>
      <table>
        <tr>
          <td>
            <textarea name="questions"><?php print $questions ?></textarea>
          </td>
          <td>
            <textarea name="answers"><?php print $answers ?></textarea>
          </td>
        </tr>
      </table>
      <input type="submit" name="submit" value="Spara" />
    </form>
    <p>
      Editeringslänk: <a href="">http://testa.roboro.se/edit/<?php print $editId ?></a><br />
      Testlänk: <a href="/<?php print $testId ?>">http://testa.roboro.se/<?php print $testId ?></a>
    </p>
```

application/view/error.php.html

```
    <pre><?php print $message ?></pre>
```

10. Design

Det som återstår är mest design. Nu handlar det i princip bara om att lägga till CSS som styr utseendet, gärna som egna filer i `www`-katalogen. Man kan även se till att applikationen ser extra bra ut på mobiler och surfplattor.

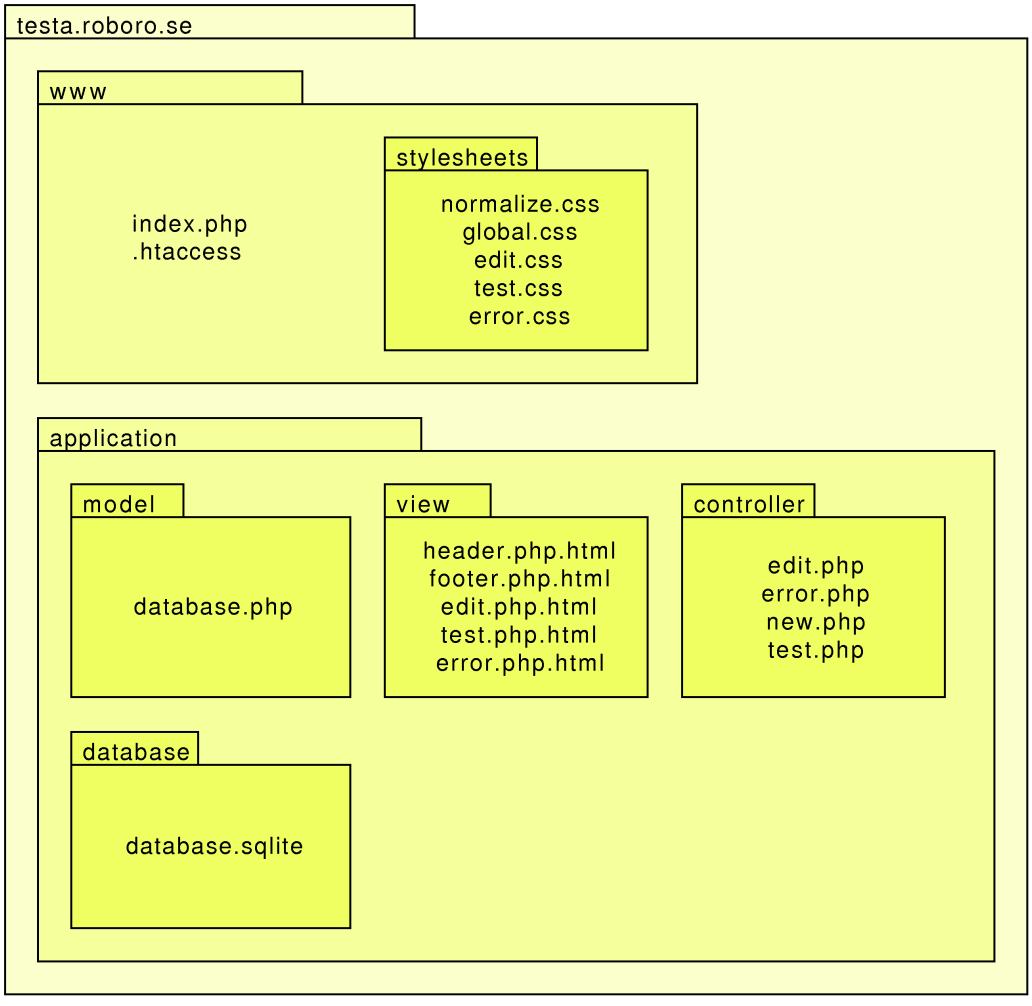
För att lägga till CSS i det här projektet kan vi ändra lite på header-filen, som används i alla vyer. Vi använder en färdig fil som gör att CSS beter sig bättre i olika browsers, som heter `normalize.css`. Den är ett öppenkällkodsprojekt som finns på github. Vi skriver också en fil som innehåller regler som gäller alla vyer, och ger varje vy en egen CSS-fil.

application/view/header_css.php.html

```
<!DOCTYPE html>
<html lang="sv">
  <head>
    <meta charset="utf-8" />
    <title><?= $title ?></title>
    <link rel="stylesheet" type="text/css" href="/stylesheets/normalize.css" />
    <link rel="stylesheet" type="text/css" href="/stylesheets/global.css" />
    <link rel="stylesheet" type="text/css" href="/stylesheets/<?= $stylesheet ?>.css" />
  </head>
  <body>
```

12. Sammanfattning

Så här ser hela katalogstrukturen ut:



13. Byta databas

Om vi vill byta till en annan databasmotor så behöver vi bara skriva en ny `database.php`. I det här fallet byter vi till en CSV-databas, vilket skulle kunna vara bra om man vill integrera den med andra system. Eftersom `Database`-klassen har samma funktioner påverkas ingen annan kod.

application/model/database_csv.php

```
<?php
  class Database
  {
    private $db;
    private static $instance = null;

    public static function get_instance()
    {
      if (self::$instance == null)
        self::$instance = new Database();

      return self::$instance;
    }

    private function Database()
    {
      $this->dbfile = 'database/database.csv';
    }

    function update_test($editId, $title, $questions, $answers)
    {
      $db = fopen($this->dbfile, 'r');

      $new_dbfile = tempnam('/tmp', 'testadb');
      $newdb = fopen($new_dbfile, 'a');

      while ($test = fgetcsv($db))
      {
        if ($test[0] == $editId)
          fputcsv($newdb, array($editId, $test[1], $title, $questions, $answers));
        else
          fputcsv($newdb, $test);
      }

      fclose($db);
      fclose($newdb);

      rename($new_dbfile, $this->dbfile);
    }

    function get_test_by_editId($id)
    {
      $db = fopen($this->dbfile, 'r');

      while (list($editId, $testId, $title, $questions, $answers) = fgetcsv($db))
      {
        if ($editId == $id)
        {
          $test = array('editId' => $editId,
                        'testId' => $testId,
                        'title' => $title,
                        'questions' => $questions,
                        'answers' => $answers);
          return $test;
        }
      }
    }
```

```
    function get_test_by_testId($id)
    {
      $db = fopen($this->dbfile, 'r');

      while (list($editId, $testId, $title, $questions, $answers) = fgetcsv($db))
      {
        if ($testId == $id)
        {
          $test = array('editId' => $editId,
                        'testId' => $testId,
                        'title' => $title,
                        'questions' => $questions,
                        'answers' => $answers);
          return $test;
        }
      }
    }

    function create_test()
    {
      do
      {
        $editId = rand(10000, 99999);
      } while ($this->get_test_by_editId($editId));

      do
      {
        $testId = rand(10000, 99999);
      } while ($this->get_test_by_testId($testId));

      $db = fopen($this->dbfile, 'a');
      fputcsv($db, array($editId, $testId, '', '', ''));

      return $editId;
    }
  }
?>
```