WEB-SOVELLUKSEN SISÄLLÖNHALLINTAPANEELI

Case Lahti360.fi

LAHDEN AMMATTIKORKEAKOULU
Tekniikan ala
Tietotekniikka
Ohjelmistotekniikka
Opinnäytetyö
Kevät 2011
Henrik Ojanen

Lahden ammattikorkeakoulu
Tietotekniikka

OJANEN, HENRIK:

Web-sovelluksen sisällönhallintapaneeli
Case Lahti360.fi

Ohjelmistotekniikan opinnäytetyö, 51 sivua

Kevät 2011

TIIVISTELMÄ

Opinnäytetyön tarkoituksena on uusia Lahti360.fi-sivusto. Tämän tavoitteen saavuttamiseksi tarkastellaan PHP-ohjelmistokehystä ja JavaScript-kirjastoa nopean web-kehityksen välineinä.

Kehitystyötä varten valittu PHP-ohjelmistokehys on Zend Framework ja sen kanssa käytettävä JavaScript-kirjasto on jQuery. Työssä tutkitaan Zend Framework -ohjelmistokehyksen rakennetta, sen käyttämiä suunnittelumalleja ja työn kannalta keskeisimpiä komponentteja. Tietokanta-, autentikointi-, lomake- ja internationalisointikomponenttien toiminta ja soveltaminen käydään läpi yksityiskohtaisesti. Tämän jälkeen tarkastellaan jQuery-kirjaston toimintaperiaatteita ja sen tarjoamia komponentteja käyttöliittymän toteuttamisen kannalta. Google Maps API -luvussa tutkitaan JavaScript-rajapintaa, jolla voidaan toteuttaa räätälöityjä karttoja sovelluksen tarpeisiin.

Lopuksi tutkittuja tekniikoita hyödynnetään työn tutkimusongelman ratkaisemisessa. PHP-toteutukseen sisältyvät sivuston käyttäjien ja sisällön hallinta, sekä kohteiden näyttäminen etusivulla. jQuerya ja Google Maps API:a käytetään saumattomasti yhdessä kohteiden esittämisessä.

Lopullinen sovellus käyttää tehokkaasti valmiita kirjastoja hyväkseen pienentäen kehitykseen kulunutta aikaa. Toteutus jättää myös oven auki jatkokehitykselle käyttäjäkokemuksen parantamiseksi entisestään.

Avainsanat: web-kehitys, PHP, JavaScript, Zend Framework, jQuery, Google Maps API

Lahti University of Applied Sciences
Degree Programme in Information Technology

OJANEN, HENRIK: Content management panel for a website
Case Lahti360.fi

Bachelor's Thesis in Software Engineering 51 pages

Spring 2011

ABSTRACT

The subject of this study is the renewal of the Lahti360.fi website. A PHP framework and a JavaScript library are examined as means of rapid web development to accomplish this objective.

The PHP framework chosen for the development is Zend Framework and the accompanying JavaScript library is jQuery. The study analyzes the structure of Zend Framework, the design patters it applies, and the most relevant components in terms of the study. The function and application of the database, authentication, form and internationalization components are examined in detail. The operational principles and components of the jQuery library are then analyzed in terms of creating the user interface. The Google Maps API section discusses the JavaScript API, i.e. the means to use custom maps for the application.

Finally the techniques analyzed are used to solve the research problem. The PHP application incorporates the management of users and spots, as well as showing the spots on the front page. jQuery and Google Maps API work seamlessly together to display the spots.

The final application makes efficient use of the ready-made libraries reducing the time spent on the development. The implementation also leaves the door open for further development to enhance the user experience.

Key words: web development, PHP, JavaScript, Zend Framework, jQuery, Google Maps API

SISÄLLYS

1 JOHDANTO

Lahti360.fi on Lahden kaupungin, Lahti travelin ja Korpimedia Oy:n yhteistyönä toteuttama sivusto, jossa esitellään Lahden kohteita ja nähtävyyksiä. Kohteista on saatavilla tietoa ja nähtävillä 360 asteen panoraamakuvia.

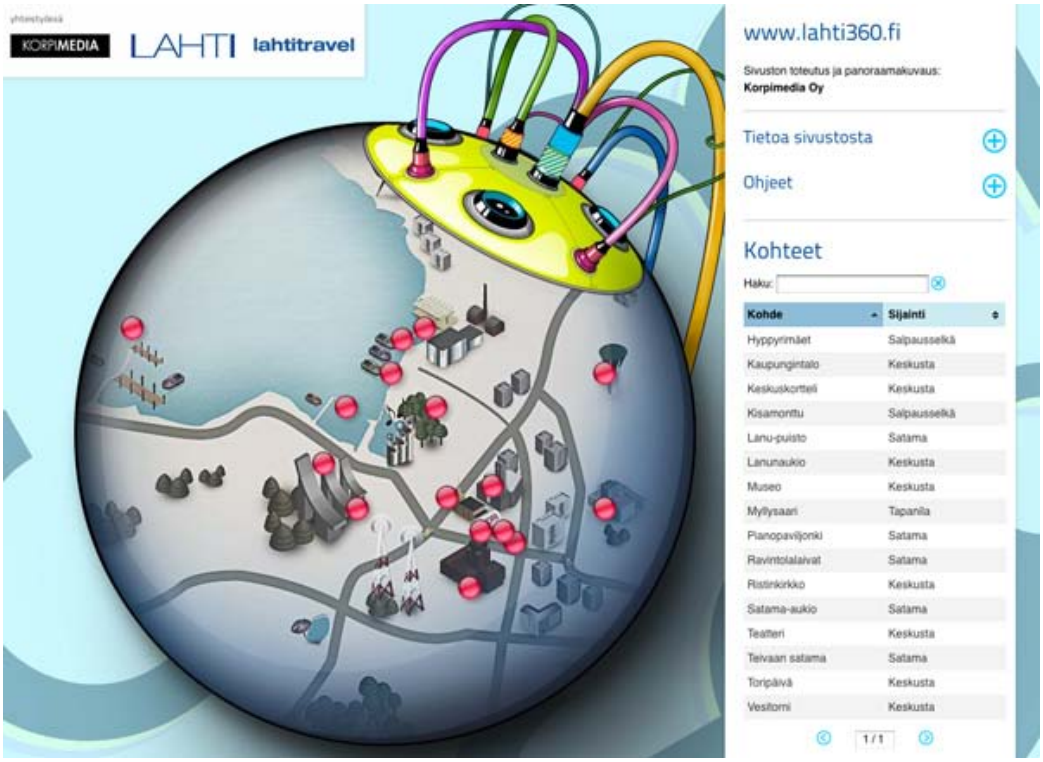
Korpimedia on digimediatoimisto, joka tarjoaa web-sivujen toteutusta ja graafista suunnittelua. Yritys toimii neljän vakituisen työntekijän voimin. Korpimedia ylläpitää Lahti360.fi-sivustoa ja vastasi sen suunnittelusta, toteutuksesta ja panoraamakuvista.

Opinnäytetyön tavoitteena on uusia Lahti360.fi-sivusto niin, että sisällön ylläpitäminen on käyttäjäystävällisempää. Sivustolle toteutetaan kirjautumisen vaativa hallintapaneeli, ja sivuston kartta uudistetaan käyttämään dynaamisempaa Google Maps -karttaa. Sivusto rakennetaan PHP:n päälle ja käyttöliittymässä sovelletaan tehokkaasti JavaScriptiä.

Työn teoriaosuudessa tarkastellaan ohjelmistokehysten ja kirjastojen hyödyntämistä nopean web-kehityksen välineinä. Tutkittaviin tekniikoihin kuuluvat PHP-ohjelmistokehys Zend Framework ja JavaScript-kirjasto jQuery. Teoriaosuudessa käydään yksityiskohtaisesti läpi työssä käytetyt Zend Frameworkin komponentit, tutkitaan hyödynnettyjä jQueryn ominaisuuksia ja tarkastellaan lyhyesti Google Maps API:n soveltamista. Tekniikat käydään läpi käytännönläheisesti esimerkein, joita käytetään myös varsinaisen sivuston toteutuksessa. Varsinaisen toteutuksen kuvaus jää siis lyhyemmäksi, lähinnä sovelluskohtaisten ratkaisujen kuvaamiseksi.

2 TOIMINTAYMPÄRISTÖN KUVAUS

Lahti360.fi on web-sivu, jossa käyttäjät voivat katsella kuvia ja lukea tietoa Lahden kohteista. Kohteet on sijoiteltu kartalle, josta niitä voidaan klikata auki. Vanhassa toteutuksessa kartta oli Flash-sovellus, johon tiedot kohteista ladattiin staattisesta XML-tiedostosta. Sisällön päivittämisen kannalta toteutus oli epäkäytännöllinen. Sivuston toteutuksesta vastaava Korpimedia päätti päivittää sivuston lähes kokonaan niin, että sisältöä voitaisiin päivittää ja lisätä suoraan erillisen hallintapaneelin kautta.



KUVIO 1. Entinen Lahti360.fi

Kuviossa 1 nähdään sivuston ensimmäinen versio, joka koostui pelkästään etusivusta, joka on staattista HTML:ää, ja XML-tiedostosta. Kohteiden valinta toimii uudessa toteutuksessa samalla tavalla kuin vanhassa; kohteen klikkaaminen kartassa tai taulukossa avaa modaalisen ikkunan, jossa voidaan tarkastella kohteen kuvia ja tietoja.

Uuden toteutuksen määrittelyihin kuului hallintapaneelin luominen PHP:llä käyttäen Zend Framework -ohjelmistokehystä. Lisäksi etusivun Flash-kartta vaihdettaisiin dynaamisempaan Google Maps -karttaan, ja sivustolle lisättäisiin käyttöliittymän kielen vaihtamisen mahdollisuus. Määrittelyn mukainen hallintapaneeli sallii käyttäjän, jolla on vaaditut käyttöoikeudet, lisätä sivustolle kohteita ja muokata ja poistaa niitä. Käyttäjän taso määrittelee, voiko tämä ainoastaan muokata sivuston sisältöä, vai myös hallita käyttäjiä. Kartan kohteet tietoineen säilötään MySQL-tietokantaan. Google Maps -kartan avulla kohteiden koordinaatit saadaan vastaamaan täysin reaalimaailman koordinaatteja.

3 WEB-TEKNOLOGIOISTA

Tyypillisesti web-sivujen kehittäminen sisältää asiakas- ja palvelinpuolen ohjelmoinnin. PHP on suosituin vaihtoehto palvelimen ohjelmointiin, eli web-sivuston perustoiminnallisuuden ja taustajärjestelmien toteuttamiseen. JavaScript on tavallisin ja useimpien selainten tukema asiakaspuolen teknologia dynaamisen käyttöliittymän luomiseen. (Wikipedia 2011d.)

Web-sovellusten kehittämisen tukemiseksi ja helpottamiseksi on kehitetty ohjelmistokehyksiä ja kirjastoja. Palvelinohjelmoinnin ohjelmistokehykset tarjoavat kirjastoja tavallisimpien toimintojen (mm. tietokannan käyttö ja istunnon hallinta) toteuttamiseen. JavaScript-kirjastot hoitavat rutiininomaiset asiat (mm. Ajax-kutsujen yksityiskohdat ja sivukohtaisten operaatioiden suorittaminen vasta sivun latauduttua) kehittäjän puolesta ja yksinkertaistavat rutiininomaiset operaatiot. Kirjastot edistävät valmiin koodin uudelleenkäyttöä ja nopeuttavat web-sivustojen kehitystä vapauttamalla kehittäjän aikaa sovelluskohtaisen koodin luomiseen. (Bibeault & Katz 2010, 8; Wikipedia 2011c.)

3.1 PHP

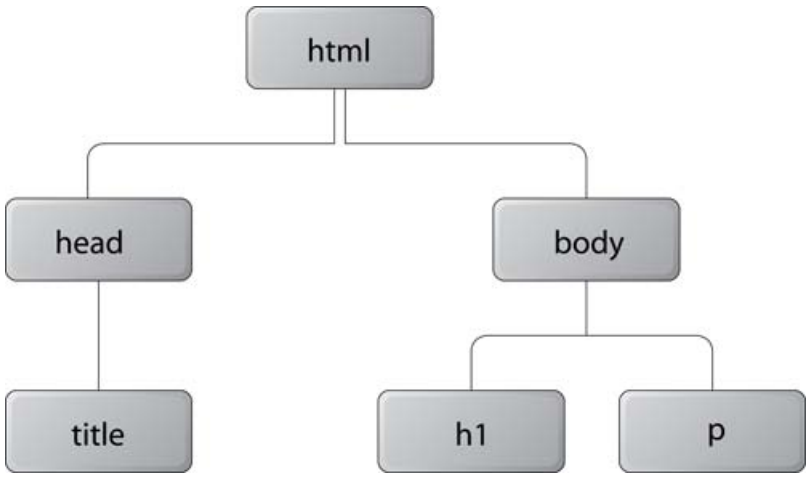
PHP on yleiskäyttöinen oliopohjainen komentosarjakieli palvelinpuolen web-kehitykseen. Se toimii useilla web-palvelimilla, käyttöjärjestelmillä ja alustoilla, ja sitä voidaan käyttää monien tietokannan hallintajärjestelmien kanssa.
75 % kaikista web-palvelimista käyttää PHP:tä. PHP:n suosio on johtanut useiden ohjelmistokehysten kehitykseen, jotka tarjoavat valmiita komponentteja ja hyödyntävät tehokkaita suunnittelumalleja sovellusten nopeaan kehitykseen. Suosittuja PHP-ohjelmistokehyksiä ovat mm. Zend Framework, CodeIgniter, CakePHP ja Symfony. (Wikipedia 2011b.)

PHP:llä voidaan luoda dynaamisia sivuja, mikä parantaa tehokkuutta staattisiin HTML-sivuihin verrattuna. PHP-ohjelmistokehys vie tehokkuuden pidemmälle vähentäen tarvetta kirjoittaa koodia yleisten toimintojen toteuttamiseksi ja tehden sivujen ylläpidosta joustavampaa ja systeemaattisempaa. Ohjelmistokehykset mahdollistavat yksinkertaistetun ja hallitumman ohjelmistokehityksen. PHP-

sovellusta luodessa joudutaan usein luomaan funktioita mm. käyttäjän oikeuksien hallintaan, tietokantaoperaatioihin ja URL-osoitteiden generoimiseen. PHP-ohjelmistokehykset kuitenkin sisältävät valmiita moduuleja tai lisäosia näiden toteuttamiseen, jolloin vaaditaan vain muutaman koodirivin pituista konfigurointia. Suositun ohjelmistokehyksen käyttäminen tarkoittaa, että ohjelmistokehyksen kehitystä tukee laaja käyttäjäyhteisö, lisäosia on saatavilla runsaasti ja turvallisuusaukot paikataan nopeasti. (Stafford 2010.)

3.2 DOM

HTML määrittelee sivun staattisena dokumenttina, joka ei sellaisenaan sisällä mitään vuorovaikutteista. Selaimen skriptikielet eivät muokkaakaan HTML:ää suoraan, vaan kaikki käsittely tapahtuu DOM-mallin kautta. DOM (Document Object Model) eli dokumenttioliomalli on selaimen käyttämä rajapinta, jonka avulla skriptit voivat dynaamisesti käsitellä XML- tai HTML-dokumentin rakennetta ja sisältöä. DOM esittää HTML-dokumentin hierarkkisena puurakenteena, jossa jokainen dokumentin olio on solmu. Solmuja ovat siis niin HTML-elementit kuin niiden sisältämät tekstitkin. Puun huipulla on dokumentin juurisolmu, joka HTML-dokumentissa on <html>-elementti. Tällä solmulla on kaksi alielementtiä, <head> ja <body>, jotka haarautuvat muihin alielementteihin. Kuviossa 2 esitetään yksinkertainen HTML-dokumentti puurakenteena. Tekstisolmut on jätetty pois selvyyden takia. (Häkkinen 2009; Olsson & O'Brien 2011.)



KUVIO 2. DOM-puu (Olsson & O'Brien 2011)

DOM:n avulla voidaan lukea HTML-tagien tekstisisältöä ja attribuutteja, poistaa elementtejä tai niiden sisältöjä, ja luoda uusia elementtejä ja lisätä ne dokumenttiin. HTML-sivu on näin ollen muokattavissa dynaamisesti ja web-sivuista voidaan tehdä vuorovaikutteisia ilman, että tarvitaan jatkuvaa palvelinyhteyttä. (Koch 2007.)

3.3 JavaScript

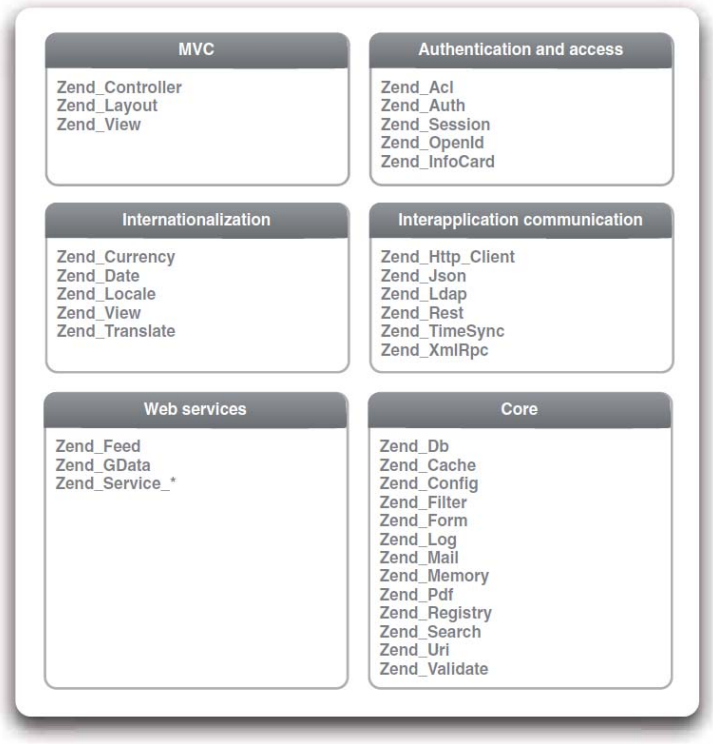
JavaScriptin keskeisin käyttötarkoitus on sisällyttää HTML-sivulle toiminnallisuutta, jolla voidaan olla vuorovaikutuksessa sivun DOM:n kanssa. JavaScriptillä voidaan esimerkiksi validoida web-lomakkeiden syötteitä ja vaihtaa kuvia hiiren siirtyessä niiden päälle. JavaScript suoritetaan paikallisesti käyttäjän selaimessa, jolloin käyttöliittymä voi reagoida käyttäjän syötteisiin nopeasti. Suurin osa yleisimmistä selaimista tukee JavaScriptiä. Vaikka siihen liittyy tehokkuusrajoituksia sen tulkattavuuden vuoksi, JavaScript-moottoreiden kasvava nopeus on tehnyt JavaScriptistä suositun kielen interaktiivisten, uuden sukupolven web-sovellusten toteuttamiseen. Esimerkkejä tästä ovat lukuisat JavaScript-kirjastot, -sovellukset ja -pelit. Yleisesti käytettyjä JavaScript-kirjastoja ovat mm. YUI, jQuery, MooTools, Prototype ja Dojo. (Wikipedia 2011a.)

JavaScriptin yleistyminen on tehnyt kielen käytöstä myös sivistyneempää kehittäjien ottaessa käyttöön täysimittaisia JavaScript-kirjastoja. Kirjastot ratkaisevat selainriippuvaisiin toimintoihin liittyvät ongelmat ja tarjoavat uusia, edistyneitä malleja web-kehitykseen. Dynaamisen toiminnallisuuden lisääminen sivulle tarkoittaa elementtien valitsemista ja niille jonkin operaation suorittamista; raa'an JavaScriptin käyttäminen voi johtaa kymmeniin koodiriveihin jokaista tällaista kaavamaista operaatiota kohden. Useimmat JavaScript-kirjastot on suunniteltu tekemään rutiininomaisista tehtävistä triviaaleja, käytännön sovelluksissa usein tarvittu koodi on tehty lyhyeksi ja yksinkertaiseksi käyttää. Toiminnallisuutta voidaan myös laajentaa helposti lisäosien avulla. (Bibeault & Katz 2010, 3—4.)

4 ZEND FRAMEWORK

Zend Framework on avoimen lähdekoodin ohjelmistokehys web-sovellusten kehittämiseen. Se on kirjoitettu PHP5:llä ja on täysin oliopohjainen. Zend Framework sisältää kattavan kirjaston komponentteja, jotka muodostavat yhdessä tehokkaan ja laajennettavan ohjelmistokehyksen. Zend Framework sisältää mm. MVC-järjestelmän, tietokantasovittimen ja lomakekomponentin, joka toteuttaa HTML-lomakkeiden renderoinnin, validoinnin ja suodatuksen. Muut komponentit, kuten Zend_Auth ja Zend_Acl, tarjoavat käyttäjien identifioinnin ja pääsynvalvonnan. (Programmer's Reference Guide 2011, Introduction to Zend Framework.)

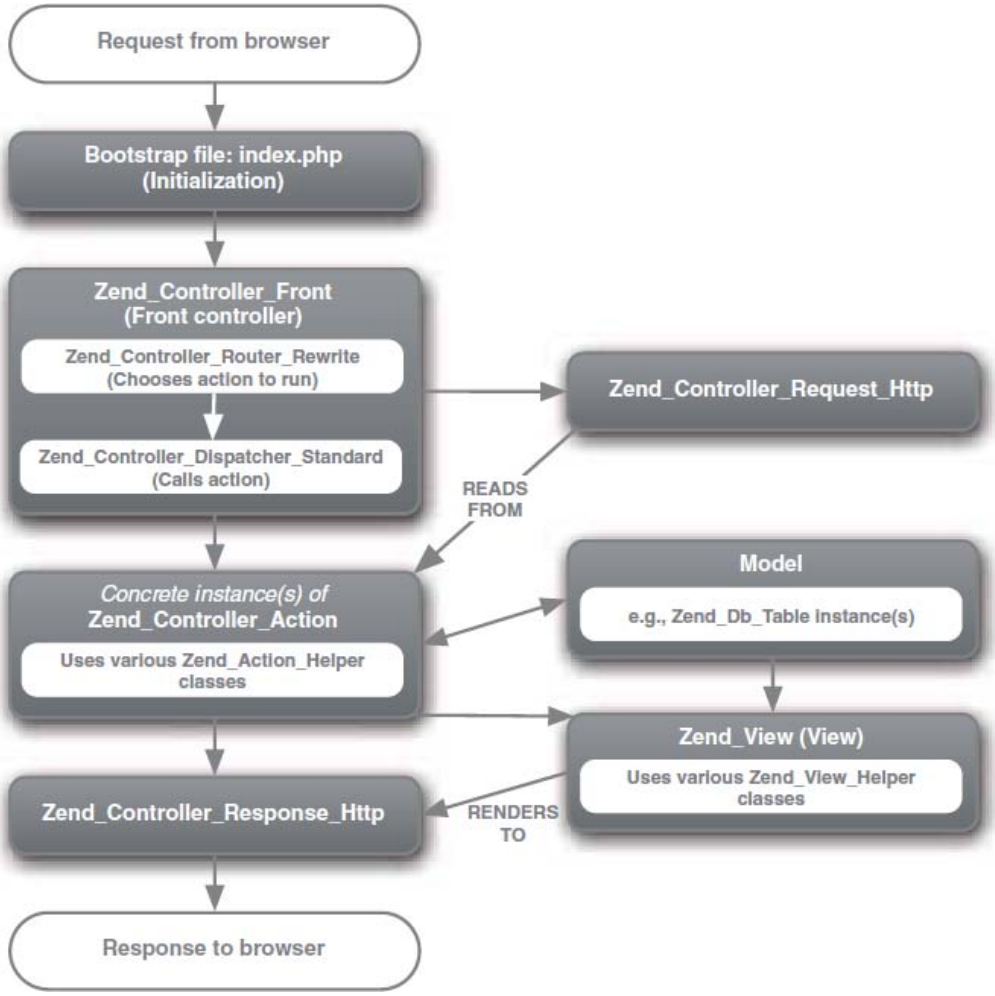
Zend Frameworkin komponentit voidaan jakaa kuuteen kategoriaan (kuvio 3) (Allen, Lo & Brown 2009, 9). Tämän opinnäytetyön kannalta olennaisia kategorioita ovat MVC, Authentication and access, Internationalization ja Core, joiden komponenteista kerrotaan tarkemmin seuraavissa luvuissa.



KUVIO 3. Zend Frameworkin komponentit (Allen, Lo & Brown 2009, 10)

4.1 Sovelluksen rakenne

Zend Framework soveltaa MVC-suunnittelumallia yhdistettynä Front Controller -suunnittelumalliin, joka keskittää sovelluksen sisäänkäyntikohdan. Pyyntöjen käsittelijä, eli web-palvelimen juuren index.php, vastaanottaa kaikki pyynnöt, päättelee suoritettavan toiminnon ja lähettää pyynnön oikealle toiminnolle. (Allen, Lo & Brown 2009, 19.) Kuvio 4 esittää järjestelmän toimintaperiaatteen.



KUVIO 4. Front Controllerin toiminta (Allen, Lo & Brown 2009, 11)

Pyyntö saapuu index.php-tiedostoon, joka käynnistää Front Controllerin. Reititin (router) ja lähettäjä (dispatcher) toimivat yhdessä määrittääkseen URL:n sisällön perusteella, mikä ohjain suoritetaan. Ohjain toimii mallin ja näkymän kanssa luodakseen lopullisen web-sivun, joka lähetetään takaisin selaimelle. (Allen, Lo &

Brown 2009, 19.) Ohjaimen toiminta on helposti mukautettavissa ja laajennettavissa kaikilla pyynnön käsittelyn tasoilla (Allen, Lo & Brown 2009, 10).

4.1.1 MVC

MVC (model-view-controller) -malli tuo sovellukseen selvän perusrakenteen määrittäen erilliset mallit (model), näkymät (view) ja ohjaimet (controller). MVC-periaate tuottaa enemmän tiedostoja, mutta jokaisella on oma selvästi rajattu tehtävänsä, jolloin ylläpito on paljon helpompaa. (Allen, Lo & Brown 2009, 4—5.) Ohjaimen toteuttaa Zend_Controller ja näkymän Zend_View, mallin muodostaa mm. Zend_Db- komponentti (Allen, Lo & Brown 2009, 10).

Malli sisältää toimintalogiikan koodia, joka toimii sovelluksen kulissien takana. Esimerkiksi kaikki tietokantaoperaatioiden suoritus on rajattu malliluokkiin. Zend_Db_Table tarjoaa taulutason pääsyn tietokantoihin ja tekee kirjoitus- ja lukuoperaatiot yksinkertaiseksi. (Allen, Lo & Brown 2009, 20.)

Näkymä toteuttaa visuaalisen logiikan, mikä tarkoittaa tavallisesti web-sivujen HTML-koodia. Se voi kuitenkin olla myös esimerkiksi XML:ää, joka muodostaa RSS-syötteen. Myös kaikki muu loppukäyttäjälle näytettävän datan generointi kuuluu näkymän tehtäviin. Näkymiin liittyvä monimutkaisempi ja toistuvasti tarvittava koodi sijoitetaan tavallisesti funktioihin, joita kutsutaan näkymäapureiksi (view helper). (Allen, Lo & Brown 2009, 20.)

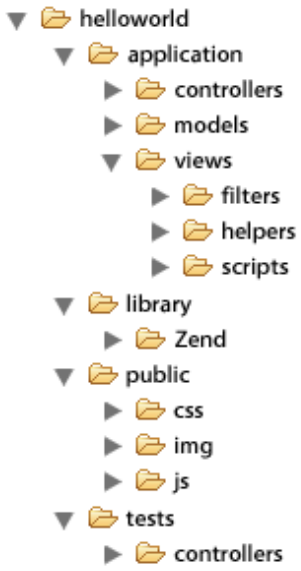
Ohjain kattaa loput sovelluksen koodista, esimerkiksi mallien ja näkymien väliset yhteydet. Ohjain päättää, mitä tehdään vastauksena web-pyyntöön. (Allen, Lo & Brown 2009, 20.)

Zend Frameworkin tiedostojen nimeämiskäytännön mukaisesti kaikkien PHP-koodia sisältävien tiedostojen nimien tulee päättyä .php-tunnisteeseen - poikkeuksena näkymätiedostot, jotka merkitään erityisellä .phtml-päätteellä. Luokkien nimien tulee vastata tiedostojen nimiä. (Programmer's Reference Guide 2011, Zend Framework Coding Standard for PHP.) EsimerkkiController-ohjainluokka esimerkiksi sijaitsee EsimerkkiController.php-tiedostossa. Tätä ohjainta vastaavat

näkymät sijoitetaan näkymäskriptien hakemiston esimerkki-kansioon, jossa index-toiminnon näkymän toteuttaa index.phtml-tiedosto.

4.1.2 Hakemistorakenne

Zend Framework -sovellus koostuu monesta hakemistosta, mikä edesauttaa osa-kokonaisuuksien erottelua. Komentorivityökalun luoman sovelluksen kansio sisältää neljä ylimmän tason hakemistoa: application, library, public ja tests. Kolmea ensimmäistä käytetään käyttäjän pyynnön käsittelyssä, tests-hakemiston Zend Framework luo yksikkötestaustiedostojen säilyttämiseen, joita voidaan käyttää koodin toiminnan testaamiseen. (Allen, Lo & Brown 2009, 21.)



KUVIO 5. Tyypillisen Zend Framework -sovelluksen hakemistorakenne (Allen, Lo & Brown 2009, 21)

Application-hakemisto sisältää kaiken sovelluksen suorittamiseen vaadittavan koodin, eikä web-palvelin pääse siihen suoraan käsiksi. Hakemisto sisältää malli-, näkymä- ja ohjaintiedostot omissa hakemistoissaan. Tarvittaessa voidaan luoda muita hakemistoja esimerkiksi konfigurointi- ja lomaketiedostoille. (Allen, Lo & Brown 2009, 21.)

Library-hakemistossa sijaitsevat kaikki Zend Frameworkin kirjastot. Samaan hakemistoon sijoitetaan myös muut kirjastot. (Allen, Lo & Brown 2009, 21.)

Public-hakemisto sisältää index.php-tiedoston, joka on Front Controller -mallin mukaisesti ainoa web-palvelimen saatavissa oleva php-tiedosto. Muita tavallisia suoraan käsiteltäviä tiedostoja ovat kuvat, CSS-tyylitiedostot ja JavaScript-tiedostot, joten ne sijoitetaan omiin alahakemistoihinsa public-hakemistoon. (Allen, Lo & Brown 2009, 21.)

Esilataustiedosto (index.php) alustaa sovelluksen. Yksinkertaisimmillaan se lisää PHP:n include-polkuun sovelluksen library-hakemiston ja käynnistää Front Controllerin. Kaikki pyynnöt, jotka eivät koske kuvia, skriptejä tai tyylitiedostoja, ohjataan esilataustiedostoon, mikä tapahtuu käyttämällä Apachen mod_rewrite-moduulia. Public-hakemistoon sijoitetaan kuvion 6 mukainen .htaccess-tiedosto. Tämä ohjaa kaikki pyynnöt index.php-tiedostoon, ellei pyyntö vastaa olemassaolevaa tiedostoa. Rivillä 2 URI-tunnisteiden uudelleenkirjoitus asetetaan käyttöön. Rivin 3 ehdon mukaan pyynnöt, jotka eivät koske olemassa olevia tiedostoja, palauttavat rivin 4 mukaisesti käyttäjälle index.php-tiedoston. (Allen, Lo & Brown 2009, 24.)

```
1  # Rewrite rules for Zend Framework
2  RewriteEngine on
3  RewriteCond %{REQUEST_FILENAME} !-f
4  RewriteRule .* index.php
```

KUVIO 6. .htaccess-tiedosto (Allen, Lo & Brown 2009, 24)

4.2 Tietokanta

Zend Frameworkin tietokantasovitinluokat mahdollistavat tietokantojen kanssa työskentelyn yhdenmukaisesti tietokannan tyypistä riippumatta. Tietokantakomponenttien ydin on Zend_Db_Adapter-luokassa, joka kommunikoi tietokannan kanssa. Taulukohtaisiin operaatioihin Zend Framework sisältää Zend_Db_Table-

komponentin, joka käyttää sisäisesti Zend_Db_Adapter-oliota. (Allen, Lo & Brown 2009, 107—108.)

4.2.1 Zend_Db_Adapter

Zend_Db_Adapter-luokasta luodaan instanssi kuvion 7 mukaisesti Zend_Db::factory()-funktiolla. Tietokantaan ei kuitenkaan muodosteta yhteyttä ennen kuin sovelluksessa suoritetaan jokin yhteyttä vaativa operaatio. (Allen, Lo & Brown 2009, 108—109.)

```
2  $params = array ('host'     => 'localhost',
3                   'username' => 'rob',
4                   'password' => 'password',
5                   'dbname'   => 'places');
6
7  $db = Zend_Db::factory('PDO_MYSQL', $params);
```

KUVIO 7. Zend_Db_Adapter-olion luominen (Allen, Lo & Brown 2009, 108)

Tietokantaan voidaan suorittaa SQL-kyselyjä sovittimen query()-metodilla. Metodi sallii ”?”-merkkien sijoittamisen SQL-lauseeseen muuttujien tilalle, jolloin sovitin varmistaa SQL:n oikeellisuuden muodostaessaan kyselyn. (Allen, Lo & Brown 2009, 109.) Kuviossa 8 on esimerkki kyselyn suorittamisesta Zend_Db_Adapter::query()-funktiolla.

```
2  $sql = 'SELECT * FROM users WHERE date_of_birth > ?';
3  $result = $db->query($sql, array('1980-01-01'));
```

KUVIO 8. SQL-kyselyn suorittaminen (Allen, Lo & Brown 2009, 109)

Toinen tapa on käyttää Zend_Db_Select-luokkaa SQL-kyselyjen generoimiseen. Suurimmat hyödyt ovat kyselyn syntaksin automaattinen varmistaminen, oliopohjaisen rajapinnan helppokäyttöisyys ja riippumattomuus tietokantamoottorista. Esimerkki yksinkertaisesta kyselystä Zend_Db_Select-luokan avulla nähdään kuviossa 9. (Allen, Lo & Brown 2009, 109—110.)

```
$select = new Zend_Db_Select($db);
$select->from('users');
       ->where('date_of_birth > ?', '1980-01-01');
$result = $select->query();
```

KUVIO 9. SQL-kyselyn generointi (Allen, Lo & Brown 2009, 110)

Tietueiden lisäämiseen, päivittämiseen ja poistamiseen Zend_Db_Adapter sisältää insert()-, update()- ja delete()-metodit. Lisäämisessä ja päivittämisessä metodille välitetään assosiatiivinen taulukko kuten kuviossa 10. insert()-metodi varmistaa automaattisesti, että data on ympäröity lainausmerkeillä asianmukaisesti. (Allen, Lo & Brown 2009, 110.)

```
// insert a user
$data = array(
    'date_created' => date('Y-m-d'),
    'date_updated' => date('Y-m-d'),
    'first_name' => 'Ben',
    'surname' => 'Ramsey'
);

$table = 'users';
$rows_affected = $db->insert($table, $data);
$last_insert_id = $db->lastInsertId();
```

KUVIO 10. Rivin lisääminen käyttäen Zend_Db_Adapter:ia (Allen, Lo & Brown 2009, 110)

Tietueen päivittäminen toimii muuten samoin, mutta metodille on välitettävä SQL-ehto rajaamaan päivitys haluttuihin riveihin. quoteInto() varmistaa, että syöte on turvallinen tietokannan kanssa käytettäväksi. (Allen, Lo & Brown 2009, 110.) Esimerkki päivittämisestä nähdään kuviossa 11.

```
// update a user
$data = array(
    'country' => 'United Kingdom'
);

$table = 'users';
$where = $db->quoteInto('country = ?', 'UK');
$db->update($table, $data, $where);
```

KUVIO 11. Rivien päivittäminen Zend_Db_Adapter:n avulla (Allen, Lo & Brown 2009, 111)

Tietueen poistamiseen delete()-metodilla tarvitaan taulun nimi ja where-lause. insert()-, update()- ja delete()-metodit palauttavat aina muuttuneiden rivien lukumäärän. (Allen, Lo & Brown 2009, 111.)

4.2.2 Zend_Db_Table

Zend_Db_Table sisältää kolme pääkomponenttia: Zend_Db_Table_Abstract, Zend_Db_Table_Rowset ja Zend_Db_Table_Row. Jokaista taulua kohden luodaan Zend_Db_Table_Abstract-luokan aliluokka. Kun taulusta valitaan useita tietueita, palautetaan Zend_Db_Table_Rowset-luokan instanssi, jota voidaan iteroida yksittäisten rivien käsittelemiseksi. Jokainen rivi on Zend_Db_Table_Row-luokan instanssi. Kuviossa 12 nähdään esimerkki periytetystä luokasta. (Allen, Lo & Brown 2009, 112—113.)

```
2  class Users extends Zend_Db_Table_Abstract
3  {
4      protected $_name = 'users';
5  }
```

KUVIO 12. Zend_Db_Table_Abstract-luokan laajentaminen (Allen, Lo & Brown 2009, 113)

$_name-jäsenmuuttuja viittaa taulun nimeen tietokannassa. users-taulusta voidaan nyt noutaa tietoja Users-luokan fetchAll()-metodilla, joka ottaa vapaaehtoisina parametreina SQL-muotoiset where-, order-, count- ja offset-lauseet. (Allen, Lo & Brown 2009, 113—114.)

Tietueiden lisääminen ja päivittäminen Zend_Db_Table:n avulla toimii lähes samoin kuin Zend_Db_Adapter:n kanssa. Ainoina eroina taulun nimeä ei tarvitse välittää metodille ja uuden rivin ID saadaan suoraan. (Allen, Lo & Brown 2009, 114.) Kuvioissa 13 ja 14 verrataan näitä operaatioita.

Zend_Db_Adapter	Zend_Db_Table

`// insert a user`
`$table = 'users';`
 	
`$data = array(`
`'date_created' => date('Y-m-d'),`
`'date_updated' => date('Y-m-d'),`
`'first_name' => 'Ben',`
`'surname' => 'Ramsey'`
`);`
 	
`$db->insert($table, $data);`
`$id = $db->lastInsertId();`
	`// insert a user`
	`$users = new Users();`
	
	`$data = array(`
	`'date_created' => date('Y-m-d'),`
	`'date_updated' => date('Y-m-d'),`
	`'first_name' => 'Ben',`
	`'surname' => 'Ramsey'`
	`);`
	
	`$id = $users->insert($data);`

KUVIO 13. Lisääminen Zend_Db_Table- ja Zend_Db_Adapter-luokilla (Allen, Lo & Brown 2009, 115)

Zend_Db_Adapter	Zend_Db_Table

`// update a user`
`$table = 'users';`
 	
`$data = array(`
`'date_updated' => date('Y-m-d'),`
`'first_name' => 'Ben',`
`'surname' => 'Ramsey'`
`);`
 	
`$where = 'id=2';`
`$db->update($table, $data,`
`$where);`
	`// update a user`
	`$users = new Users();`
	
	`$data = array(`
	`'date_updated' => date('Y-m-d'),`
	`'first_name' => 'Ben',`
	`'surname' => 'Ramsey'`
	`);`
	
	`$where = 'id=2';`
	`$users->update($data, $where);`

KUVIO 14. Päivittäminen Zend_Db_Table- ja Zend_Db_Adapter-luokilla (Allen, Lo & Brown 2009, 115)

Kuviossa 15 havainnollistetaan Zend_Db_Table-luokan näkyvintä hyötyä. Zend_Db_Table_Row-luokka piilottaa rivin lisäämisen ja päivittämisen erot, ja save()-metodia voidaan käyttää molempien operaatioiden suorittamiseen. (Allen, Lo & Brown 2009, 115—116.)

Inserting a new record	Updating a record

`$users = new Users();`
`$row = $users->fetchNew();`
 	
`$row->first_name = 'Chris';`
`$row->surname = 'Shiflett';`
 	
`$row->save();`
	`$users = new Users();`
	`$row = $users->fetchRow('id=2');`
	
	`$row->first_name = 'Chris';`
	`$row->surname = 'Shiflett';`
	
	`$row->save();`

KUVIO 15. Rivin lisääminen ja päivittäminen (Allen, Lo & Brown 2009, 115)

4.3 Autentikointi ja ACL

Autentikointi tarkoittaa käyttäjän tunnistamista, yleensä käyttäjänimi/salasana -yhdistelmän avulla. Access control -prosessi päättää, mihin tunnistetulla käyttäjällä on pääsyoikeus tai mitä tällä on oikeus muokata. Zend Frameworkissa on näiden toteuttamiseksi kaksi komponenttia: Zend_Auth ja Zend_Acl. Käyttäjän tunnistamiseen käytettävää Zend_Auth-komponenttia käytetään tavallisesti Zend_Session-komponentin kanssa, joka tallettaa ja säilyttää tiedon tunnistautuneesta käyttäjästä. Zend_Acl käyttää sitten tunnusta tarkistaakseen käyttäjän pääsyoikeudet käyttäen roolipohjaista käyttöoikeuksien hallintajärjestelmää. (Allen, Lo & Brown 2009, 12.)

4.3.1 Zend_Auth

Zend_Auth käyttää Singleton-suunnittelumallia, joten tunnistautumistiedot voidaan noutaa aina tarvittaessa. Tietokantapohjaisen autentikoinnin tapauksessa käytetään Zend_Auth_Adapter_DbTable-sovitinta. (Allen, Lo & Brown 2009, 130.)

Kuvio 16 esittää esimerkin web-sivun sisäänkirjautumisprosessin käsittelystä. Käyttäjätunnuksen ja salasanan oikeellisuus tarkistetaan Zend_Auth-luokan avulla. (Allen, Lo & Brown 2009, 134.)

```
2   $authAdapter = $this->_getAuthAdapter($formData);
3   $auth = Zend_Auth::getInstance();
4   $result = $auth->authenticate($authAdapter);
5
6   if (!$result->isValid()) {
7       //login failed...
8   } else {
9       $data = $authAdapter->getResultRowObject(null,'password');
10      $auth->getStorage()->write($data);
11
12      $this->_redirect($this->_redirectUrl);
13      return;
14  }
```

KUVIO 16. Käyttäjänimen ja salasanan validointi (Allen, Lo & Brown 2009, 134)

$formData-taulukkoon on aiemmin asetettu ”username”- ja ”password”-elementit käyttäjän syötteiden pohjalta. Zend_Auth_Adapter_DbTable-instanssin määrittäminen on sijoitettu _getAuthAdapter()-metodiin (esitetään kuviossa 17). Autentikointi suoritetaan authenticate()-metodilla, jonka palauttaman olion isValid()-metodilla testataan kirjautumisen onnistuminen. Rivillä 10 käyttäjän identiteetti varastoidaan istuntoon. (Allen, Lo & Brown 2009, 135.)

```
2   protected function _getAuthAdapter($formData)
3   {
4       $dbAdapter = Zend_Registry::get('db');
5
6       $authAdapter = new Zend_Auth_Adapter_DbTable($dbAdapter);
7       $authAdapter->setTableName('users')
8               ->setIdentityColumn('username')
9               ->setCredentialColumn('password')
10              ->setCredentialTreatment('SHA1(?)');
11
12      $authAdapter->setIdentity($formData['username']);
13      $authAdapter->setCredential($formData['password']);
14
15      return $authAdapter;
16  }
```

KUVIO 17. Autentikointisovittimen määrittely (Allen, Lo & Brown 2009, 136)

Kuvion 17 rivillä 4 noudetaan sovelluksen esilatauksessa rekisteriin varastoitu tietokantasovitin. Riveillä 7—9 asetetaan käyttäjätunnustaulu ja ne taulun sarakkeet, jotka sisältävät tunnukset ja salasanat. Rivillä 10 asetetaan tiivistealgoritmi, jota käytetään salasanojen tiivistämiseen. Riveillä 12 ja 13 asetetaan tunnistautumistiedot sovittimeen. (Allen, Lo & Brown 2009, 136.)

Uloskirjautuminen puolestaa on paljon yksinkertaisempaa. Käyttäjän tunniste poistetaan istunnosta Zend_Auth-luokan clearIdentity()-metodilla. (Allen, Lo & Brown 2009, 138.)

4.3.2 Zend_Acl

Zend_Acl käyttää roolipohjaista käyttöoikeuksien hallintaa, jossa käytetään termejä roolit ja resurssit. Web-sovelluksessa rooli tarkoittaa yleisesti Zend_Authin identifioimaa käyttäjäryhmää. Resurssi on mikä tahansa suojattu kohde, esimerkiksi tietokannan tietue tai tiedosto. (Allen, Lo & Brown 2009, 12.)
Roolin toteuttaa Zend_Acl_Role-luokka, joka sisältää ainoastaan roolin nimen. Rooli lisätään Zend_Acl-instanssiin sen addRole()-metodilla. Rooli voi periä toisen roolin oikeudet. Kuviossa 18 esitetään esimerkkitilanne, jossa moderator-rooli perii member-roolin. (Allen, Lo & Brown 2009, 139—140.)

```
2  $acl = new Zend_Acl();
3  $acl->addRole(new Zend_Acl_Role('member'));
4  $acl->addRole(new Zend_Acl_Role('moderator'), 'member');
```

KUVIO 18. Roolien luominen ja perintä (Allen, Lo & Brown 2009, 140)

Myös Zend_Acl_Resource:n luomiseen vaaditaan vain resurssin nimi. Resurssi liitetään Zend_Acl-instanssiin add()-metodilla. Resurssi voidaan periä toisesta resurssista. Kuvion 19 esimerkissä luodaan posts- ja threads-nimiset resurssit, jotka periytetään forum-resurssista. Resurssit viittaavat tyypillisesti ohjaimiin tai toimintoihin, tässä tapauksessa posts ja threads olisivat ohjaimia forum-moduulissa. (Allen, Lo & Brown 2009, 140.)

```
2  $acl = new Zend_Acl();
3  $acl->addRole(new Zend_Acl_Role('member'));
4  $acl->addRole(new Zend_Acl_Role('moderator'), 'member');
5  $acl->add(new Zend_Acl_Resource('forum'));
6  $acl->add(new Zend_Acl_Resource('posts'), 'forum');
7  $acl->add(new Zend_Acl_Resource('threads'), 'forum');
```

KUVIO 19. Resurssien luominen ja periyttäminen (Allen, Lo & Brown 2009, 140)

Lopuksi Zend_Acl-olioon asetetaan, mitä oikeuksia rooleilla on resursseihin. Lupien asettamiseen on kaksi metodia: allow() ja deny(). Aluksi pääsy on kielletty kaikilta rooleilta kaikkiin resursseihin. allow() antaa roolille luvan päästä tiettyyn resurssiin, deny() poistaa luvan. Kuvion 20 esimerkissä member-roolille annetaan view-oikeus forum-resurssiin, ja moderator-roolille sen lisäksi moderate- ja blacklist-oikeudet. (Allen, Lo & Brown 2009, 140—141.)

```
2  $acl->allow('member', 'forum', 'view');
3  $acl->allow('moderator', 'forum', array('moderate', 'blacklist');
```

KUVIO 20. Roolien oikeuksien asettaminen (Allen, Lo & Brown 2009, 141)

Hyvä tapa implementoida ACL-tarkistus on käyttää avustajaluokkaa. Avustajaluokka rekisteröidään järjestelmään, ja se mahdollistaa käyttäjän oikeuksien tarkistamisen ennen pyydetyn toiminnon suorittamista. Oikeuksien tarkastaminen suoritetaan preDispatch()-metodissa, jota Front Controller kutsuu automaattisesti ennen pyynnön lähettämistä edelleen. (Allen, Lo & Brown 2009, 142.) Kuvion 21 esimerkissä tarkastellaan avustajaluokan olennaisinta metodia.

```
2   public function preDispatch()
3   {
4       $role = 'guest';
5       if ($this->_auth->hasIdentity()) {
6           $user = $this->_auth->getIdentity();
7           if(is_object($user)) {
8               $role = $this->_auth->getIdentity()->role;
9           }
10      }
11
12      $request = $this->_action->getRequest();
13      $controller = $request->getControllerName();
14      $action = $request->getActionName();
15      $module = $request->getModuleName();
16      $this->_controllerName = $controller;
17
18      $resource = $controller;
19      $privilege = $action;
20
21      if (!$this->_acl->has($resource)) {
22          $resource = null;
23      }
24
25      if (!$this->_acl->isAllowed($role, $resource,
26              $privilege)) {
27          $request->setModuleName('default');
28          $request->setControllerName('auth');
29          $request->setActionName('login');
30          $request->setDispatched(false);
31      }
32  }
```

KUVIO 21. ACL-avustajaluokan preDispatch()-metodi (Allen, Lo & Brown 2009, 144)

Rivillä 4 oletusrooliksi asetetaan guest. Rivillä 5 tarkistetaan, onko käyttäjä kirjautunut sisään, ja rivillä 8 noudetaan tunnistautuneen käyttäjän rooli. Riveillä 18 ja 19 asetetaan testattava resurssi ja oikeus. Rivillä 21 varmistetaan, että resurssi on olemassa. Rivillä 25 testataan, onko roolilla tarvittavat oikeudet. Jos ei, pyyntö asetetaan osoittamaan kirjautumissivulle. Jos käyttäjän roolilla on vaaditut oikeudet, pyyntöön ei tehdä muutoksia ja käyttäjä ohjautuu haluamalleen sivulle. (Allen, Lo & Brown 2009, 145.)

4.4 Lomakkeet

Lomakkeiden toteuttamiseen Zend Framework sisältää Zend_Form-komponentin. Zend_Form:n kanssa käytettävät Zend_Filter- ja Zend_Validate-komponentit toimivat käyttäjän syötteiden suodattamisessa ja validoinnissa. (Allen, Lo & Brown

2009, 147—148.) Lomakkeen muotoiluun käytetään kuorruttajia (decorator), jotka määräävät, miten lomake ja sen elementit esitetään HTML-koodissa.

Esimerkkinä kuviossa 22 on toteutettu minimalistinen sisäänkirjautumislomake, jossa on käyttäjänimi- ja salasanakentät sekä lähetyspainike. Zend_Form-luokan muodostinfunktio kutsuu init()-metodia, jossa määritetään lomakkeen komponentit. Elementit luodaan ja lisätään lomakkeeseen addElement()-metodilla, jossa voidaan myös välittää elementille parametreja. (Allen, Lo & Brown 2009, 155—156.)

```
2   class LoginForm extends Zend_Form
3   {
4       public function init($action)
5       {
6           $this->setAction($action)
7               ->setMethod('post')
8               ->setAttrib('id', 'loginForm');
9
10          $username = $this->addElement('text', 'username',
11              array('label' => 'Username'));
12
13          $password = $this->addElement('password', 'password',
14              array('label' => 'Password'));
15
16          $submit = $this->addElement('submit', 'Login');
17      }
18  }
```

KUVIO 22. Tyypillinen Zend_Form-kirjautumislomake (Allen, Lo & Brown 2009, 156)

Toimiva lomake siistii käyttäjän syötteet ja suodattaa datan niin, että se on hyväksyttävää sen käyttöyhteydessä. Zend_Filter-luokka sisältää kokoelman suodattimia, jotka poistavat tai muuntavat ei-toivotun datan syötteestä. Zend_Filter_Alnum-suodatin esimerkiksi poistaa merkkijonosta kaikki muut merkit kuin aakkosnumeeriset, Zend_Filter_StringToLower muuntaa kirjaimet pienaakkosiksi, ja Zend_Filter_HtmlEntities muuttaa erikoismerkit (esim. ”<”) merkkiviittauksiksi (”<”). (Allen, Lo & Brown 2009, 11—12, 148.)

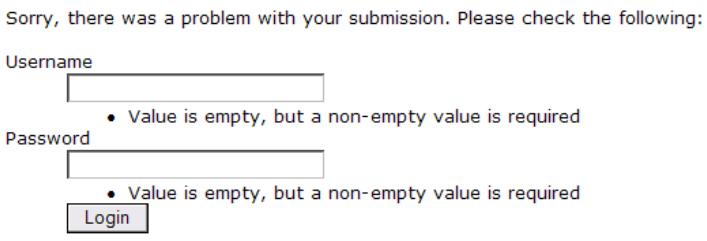
Zend_Validate puolestaan validoi syötteen, eli tarkistaa, että syöte vastaa asetettuja rajoituksia. Virheellisen syötteen tapauksessa voidaan käyttäjän antaa muokata

syötettä. Esimerkkeinä Zend_Validate_StringLength tarkistaa, onko syötteen pituus annettujen rajojen sisällä, ja Zend_Validate_Alnum-validaattoria käytetään hylkäämään syötteet, jotka sisältävät muita kuin kirjaimia ja numeroita. (Allen, Lo & Brown 2009, 150.)

Kuviossa 23 käyttäjänimi- ja salasanaelementteihin liitetään suodatus ja validointi addFilter()- ja addValidator()-metodeilla. Rivien 6 ja 13 setRequired()-metodi käyttää Zend_Validate_NotEmpty-validaattoria, joka hylkää tyhjät syötteet. (Allen, Lo & Brown 2009, 156—157.) Kuvio 24 näyttää virheviestit virheellisten syötteiden lähettämisen jälkeen. Virheviestien sisältö on vaihdettavissa validaattorin setMessage()-metodilla. (Allen, Lo & Brown 2009, 157—158.)

```
2   $this->addElement('text', 'username',
3       array('label' => 'Username'));
4   $username = $this->getElement('username')
5           ->addValidator('alnum')
6           ->setRequired(true)
7           ->addFilter('StringTrim');
8
9   $this->addElement('password', 'password',
10      array('label' => 'Password'));
11  $password = $this->getElement('password')
12          ->addValidator('stringLength', true, array(6))
13          ->setRequired(true)
14          ->addFilter('StringTrim');
```

KUVIO 23. Suodatuksen ja validoinnin lisääminen lomake-elementteihin (Allen, Lo & Brown 2009, 157)



KUVIO 24. Tyhjän lomakkeen lähettämisen tulos (Allen, Lo & Brown 2009, 157)

Kuorruttajien avulla voidaan lomake-elementit ympäröidä halutuilla HTML-tageilla ja esimerkiksi esittää lomakkeen virheviestit elementtien yhteydessä (Al-

len, Lo & Brown 2009, 151). Zend_Form_Element ja Zend_Form lataavat tietyt oletuskuorruttimet, kuvio 24 havainnollistaa asiaa käytännössä. Zend_Formin clearDecorators()-metodi poistaa oletuskuorruttimet, minkä jälkeen voidaan asettaa oma kuorrutinkokoonpano. (Allen, Lo & Brown 2009, 163.)

4.5 Internationalisointi

Zend Framework sisältää komponentteja sovelluksen lokalisointiin. Tähän sisältyy pienempiä asioita, kuten oikean valuuttasymbolin käyttö, sekä täysi tuki koko sivun tekstien vaihtamiseen halutulle kielelle. Zend_Locale-, Zend_Currency- ja Zend_Measure-luokat vastaavat siitä, että sovelluksessa käytetään oikeaa kieltä ja oikeita ilmauksia. Zend_Translate hoitaa web-sivun tekstisisällön varsinaisen kääntämisen. (Allen, Lo & Brown 2009, 13.)

Web-sivun tekstien kääntäminen tarkoittaa jokaisen sivulla näkyvän merkkijonon mappaamista kohdekieliseen merkkijonoon. Jokaiselle näytettävälle merkkijonolle on siis oltava käännetty versio. Zend_Translate tukee käännöksien säilömiseen mm. taulukoita, CSV-tiedostoja ja INI-tiedostoja. Kuviossa 25 käytetään Zend_Translate-komponentin taulukkosovitinta tulosteiden kääntämiseen. (Allen, Lo & Brown 2009, 300, 303.)

```
2   $data = array();
3   $data['hello'] = 'WELCOME';
4   $data['today %1$s'] = 'TODAY\'S DATE IS %1$s';
5
6   $translate = new Zend_Translate('array', $data, 'en');
7
8   print $translate->_("hello")."\n";
9   print "=======\n";
10  printf($translate->_('today %1$s')."\n",
11      date('d/m/Y'));
```

KUVIO 25. Sisällön kääntäminen Zend_Translate-komponentilla (Allen, Lo & Brown 2009, 304)

Rivillä 2 luodaan käännöstaulukko. Rivillä 6 luodaan Zend_Translate-luokan instanssi, parametreina annetaan käännössovittimen tyyppi, käännösdata ja kieli-

alue. Muita kieliä voitaisiin lisätä Zend_Translate-luokan addTranslation()-metodilla. Rivillä 8 käytetään _()-funktiota kääntämisen suorittamiseksi. ”hello”-avainta käytetään oikean merkkijonon noutamiseksi. Formatoidun tekstin tulostaminen printf()-funktiolla toimii myös moitteetta _()-funktion kanssa. (Allen, Lo & Brown 2009, 304.)

Käyttäjän valitsema kieli voidaan välittää esimerkiksi URL-osoitteessa, jolloin saksankielinen sivu olisi muotoa www.example.com/de. Sovelluksen reititysformaattia on muokattava, jotta kielikoodia voidaan käyttää osoitteessa. Oletuksena tulkittava polku on muotoa ”/{moduuli}/{ohjain}/{toiminto}/{muita parametreja}”, missä ”{moduuli}” on vapaavalintainen. Tarkoituksenmukainen polku olisi muotoa ”/{kieli}/{ohjain}/{toiminto}/{muita parametreja}”. Tämän toteuttamiseksi luodaan uusi reitityssääntö Bootstrap-luokkaan (esilataaja) kuvion 26 mukaisesti. (Allen, Lo & Brown 2009, 306—307.)

```
2   $zl = new Zend_Locale();
3   $lang = in_array($zl->getLanguage(), $languages)
4           ? $zl->getLanguage() : 'en';
5
6   // add language to default route
7   $route = new Zend_Controller_Router_Route(
8       ':lang/:controller/:action/*',
9       array('controller'=>'index',
10          'action' => 'index',
11          'module'=>'default',
12          'lang'=>$lang));
13
14  $router = $frontController->getRouter();
15  $router->addRoute('default', $route);
16  $frontController->setRouter($router);
```

KUVIO 26. Reitityssäännön asettaminen (Allen, Lo & Brown 2009, 307)

$languages on taulukko, johon on ladattu saatavilla olevien kielten lyhenteet. Rivillä 3 käytetään Zend_Locale::getLanguage()-funktiota käyttäjän selaimen oletuskielen noutamiseksi, jos se on käytettävissä olevien kielten listassa. Rivillä 7 määritellään uusi reitti Zend_Controller_Router_Route-luokan avulla. Reitin muuttujaosat määritellään kaksoispisteellä, asteriski tarkoittaa muita parametreja. Jokaiselle reitin osalle määritellään oletusarvo, jota käytetään osan puuttuessa. (Allen, Lo & Brown 2009, 308.)

Aiemmin luotu Zend_Translate-olio lisätään rekisteriin kutsumalla Zend_Registry::set('Zend_Translate', $translate). ”Zend_Translate”-avainsanan takia käännösolio on kaikkien näkymien käytettävissä translate()-avustajafunktion avulla. Esimerkiksi käännöslistan ”hello”-avainta vastaava merkkijono tulostetaan nyt kutsumalla $this->translate('hello'). (Allen, Lo & Brown 2009, 310.)

Varsinainen kielen vaihtaminen tapahtuu uudessa avustajaluokassa, jonka dispatchLoopStartup()-metodissa asetetaan käyttäjän pyytämä kieli (kuvio 27). (Allen, Lo & Brown 2009, 309.) Kuvion rivillä 2 noudetaan pyynnön lang-parametri, ja rivillä 4 kieli asetetaan käytettäväksi sovelluksessa.

```
2  $lang = $this->getRequest()->getParam('lang');
3  $translate = Zend_Registry::get('Zend_Translate');
4  $translate->setLocale($lang);
```

KUVIO 27. Kielen vaihtaminen Zend_Translate-oliolla

5 JQUERY

JavaScript-kirjasto jQuery suunniteltiin tekemään rutiininomaisista tehtävistä triviaaleja. Se yksinkertaistaa HTML-dokumenttien läpikäymisen, tapahtumien käsittelyn, käyttöliittymän animoinnin ja Ajaxin käytön nopeuttaen ohjelmistokehitystä. (Bibeault & Katz 2010, 4; jQuery 2011.)

5.1 Ydintoiminnallisuus

Tässä alaluvussa tarkastellaan jQuery-kirjaston ydintoimintoja. Näihin lukeutuvat perustavanlaatuisimmat menetelmät DOM-elementtien käsittelyyn dynaamisen käyttöliittymän luomiseksi.

5.1.1 Elementtien valitseminen

jQuery käyttää CSS-tyylisiä valitsimia, jotka kuvaavat elementtejä niiden tyypin, attribuuttien tai sijainnin perusteella. Esimerkiksi valitsin muotoa ”p a” viittaa kaikkiin linkkeihin (<a>-elementti) <p>-elementin sisällä. Valitsin välitetään jQuerylle yksinkertaisella syntaksilla: $("p a"). $()-funktio (alias jQuery()-funktiolle) palauttaa JavaScript-olion, joka sisältää taulukkona valitsinta vastaavat DOM-elementit. Näille elementeille voidaan suorittaa jQueryssa määriteltyjä operaatioita. (Bibeault & Katz 2010, 8—9.)

Jos esimerkiksi halutaan piilottaa kaikki -elementit, joilla on kohde-luokka, voidaan käyttää hide()-metodia: $("div.kohde").hide(). Huomioitavaa näissä funktioissa on se, että niitä voidaan ketjuttaa, koska funktio palauttaa aina alkuperäisen elementtijoukon. Seuraava esimerkki piilottaa elementit ja lisää niihin poistettu-nimisen luokan: $("div.kohde").hide().addClass("poistettu"). (Bibeault & Katz 2010, 9.) jQuery tukee myös monimutkaisempia CSS-valitsimia, muutamia esimerkkejä taulukossa 1.

TAULUKKO 1. Esimerkkejä jQueryn valitsimista (Bibeault & Katz 2010, 10)

Valitsin	Tulos
$("p:even")	Valitsee kaikki parilliset <p>-elementit
$("tr:nth-child(1)")	Valitsee jokaisen taulun ensimmäisen rivin
$("body > div")	Valitsee <body>:n välittömät -alielementit
$("a[href$= 'pdf ']")	Valitsee PDF-tiedostolinkit

Hyvin rakennetussa web-sivussa JavaScript-koodi pidetään erillään HTML-rakenteesta. Tällöin koodin on tiedettävä, milloin operoitavat elementit ovat latautuneet. Tätä varten käytetään jQueryn ready()-metodia kuten kuvion 28 rivillä 1, lyhennetty muoto tästä nähdään rivillä 7. (Bibeault & Katz 2010, 12.)

```
2  $(document).ready(function(){
3      //dokumentin latautumisen jälkeen suoritettava koodi
4  });
5
6  //vaihtoehtoisesti:
7  $(function(){
8      //lisää dokumentin latautumisen jälkeen suoritettavaa koodia
9  });
```

KUVIO 28. jQueryn ready()-metodi

$()-funktiolla voidaan myös luoda DOM-elementtejä. Esimerkkinä kuvion 29 rivillä 2 luodaan tavallinen <img>-elementti, riveillä 4—6 määritellään sille attribuutteja, rivillä 7 asetetaan klikkauksen käsittelijäfunktio, rivillä 11 kuvaan lisätään CSS-tyylejä, ja rivillä 17 elementti liitetään dokumenttiin. (Bibeault & Katz 2010, 33.)

```
$('<img>',
    {
        src: 'images/little.bear.png',
        alt: 'Little Bear',
        title:'I woof in your general direction',
        click: function(){
            alert($(this).attr('title'));
        }
    })
    .css({
        cursor: 'pointer',
        border: '1px solid black',
        padding: '12px 12px 20px 12px',
        backgroundColor: 'white'
    })
    .appendTo('body');
```

KUVIO 29. <img>-elementin luominen dynaamisesti kaikilla ominaisuuksilla (Bibeault & Katz 2010, 33)

Jos valitun joukon elementeille halutaan suorittaa monimutkaisempia operaatioita, joukkoa voidaan iteroida each()-metodilla, joka suorittaa määrätyn funktion jokaisen elementin kohdalla. Funktiolle välitetään kaksi parametria: elementin indeksi ja elementti itse. Kuviossa 30 muokataan jokaisen kuvaelementin alt-ominaisuutta. (Bibeault & Katz 2010, 49.)

```
$('img').each(function(n){
    this.alt='This is image['+n+'] with an id of '+this.id;
});
```

KUVIO 30. Elementtijoukon iterointi each()-metodilla (Bibeault & Katz 2010, 49)

5.1.2 Tapahtumankäsittely

HTML:llä ja JavaScriptillä toteutettu käyttöliittymä on asynkroninen ja tapahtumapohjainen; se odottaa, että jotain kiinnostavaa tapahtuu ja reagoi tapahtumaan määrätyllä tavalla. Tapahtumankäsittelijä määritellään asettamalla elementin ominaisuuksiin viittaus funktioon. Esimerkiksi onclick käsittelee elementin klikkauk-

sesta syntyvän tapahtuman, ja onmouseover-käsittelijän funktiota kutsutaan käyttäjän viedessä hiiren elementin päälle. (Bibeault & Katz 2010, 93—95.)

jQueryn tapahtumien käsittely mahdollistaa mm. yhtenäisen tapahtumankäsittelijöiden käytön eri selainten kanssa ja sallii useamman käsittelijän asettamisen elementin samalle tapahtumatyypille. DOM-elementteihin lisätään tapahtumankäsittelijöitä bind()-metodilla. Kuviossa 31 sidotaan klikkaustapahtuman käsittelijä jokaiselle sivun kuvaelementille. (Bibeault & Katz 2010, 106—107.)

```
2  $('img').bind('click',function(event){
3      alert('Hi there!');
4  });
```

KUVIO 31. Tapahtumankäsittelijän määrittäminen ilman selainsidonnaista koodia (Bibeault & Katz 2010, 107)

bind()-metodin lisäksi useat tapahtumankäsittelijät voidaan asettaa lyhyesti erityisillä metodeilla, kuten click(), load() tai mouseenter(), jotka ottavat parametrinaan käsittelijäfunktion (Bibeault & Katz 2010, 110). Tapahtumankäsittelijä poistetaan unbind()-metodilla (Bibeault & Katz 2010, 111).

5.1.3 Elementtien animointi

On yleensä visuaalisesti miellyttävämpää ja havainnollisempaa katsella elementtien siirtyvän tilasta toiseen asteittaisesti. jQueryssa on kolmenlaisia tehosteita: näyttäminen ja piilottaminen, sisään ja ulos häivyttäminen, ja ylös ja alas liu'uttaminen. (Bibeault & Katz 2010, 144.)

Elementin näyttäminen ja piilottaminen show()- ja hide()-metodeilla voidaan animoida välittämällä niille parametreja. hide(kesto, callback) piilottaa elementin säätämällä sen leveyttä, korkeutta ja läpinäkyvyyttä nollaa kohden kesto-parametrin ajan (millisekunteina). Lopuksi elementille kutsutaan callback-funktiota. show(kesto, callback) toimii vastaavasti tuoden elementin näkyviin asteittaisesti. (Bibeault & Katz 2010, 144—145.)

fadeIn(kesto, callback)-kutsu häivyttää elementin näkyviin annetun ajan kuluessa muuttamalla sen läpinäkyvyyttä. fadeOut() häivyttää elementin näkymättömiin vastaavasti. (Bibeault & Katz 2010, 150.) slideDown()- ja slideUp()-metodit piilottavat ja näyttävät elementin muuttamalla sen korkeutta (Bibeault & Katz 2010, 152).

Näiden lisäksi animate()-metodilla voidaan itse asettaa CSS-ominaisuudet, joita kohti elementin vastaavat ominaisuudet animoidaan (Bibeault & Katz 2010, 154). Kuvion 32 esimerkki muuttaa valitun elementin taustavärin punaiseksi puolen sekunnin aikana.

```
2  $("#esimerkki").animate({backgroundColor: "red"}, 500);
```

KUVIO 32. Elementin animointi

5.1.4 Ajax

Ajax (Asynchronous JavaScript and XML) mahdollistaa asynkronisten pyyntöjen lähettämisen palvelimelle tarvitsematta ladata koko sivua uudelleen. Tyypillisen Ajax-kutsun toteuttaminen tarkoittaa tavallisesti jopa 20 riviä koodia. Vastaava koodi voidaan jQuerylla toteuttaa paljon lyhyemmin. (Bibeault & Katz 2010, 242—243.) Kuviossa 33 noudetaan dataa palvelimen resurssi.php:ltä ja asetetaan se esimerkki-elementin sisällöksi.

```
2  $('#esimerkki').load('resurssi.php');
```

KUVIO 33. Datan lataaminen palvelimelta

load()-metodin parametreina voidaan määrittää URL:n lisäksi palvelimelle lähetettävä data ja takaisinkutsufunktio. load()-metodi suorittaa joko GET- tai POST-pyynnön riippuen lähetettävän datan muodosta. Joissain tapauksissa käytettävään

HTTP-metodiin halutaan kuitenkin vaikuttaa. Tyypillisesti saman GET-pyynnön odotetaan palauttavan aina saman tuloksen. POST-pyyntöä puolestaan voidaan käyttää esimerkiksi tietojen muokkaamiseen tai poistamiseen tietokannasta. (Bibeault & Katz 2010, 250.)

Tätä erottelua varten jQueryssa on $.get()- ja $.post()-funktiot. $.post(url, parametrit, callback) suorittaa POST-pyynnön URL:n osoittamalle palvelimen resurssille annetuilla parametreilla (Bibeault & Katz 2010, 255). Edellä mainittu esimerkki voidaan toteuttaa $.post()-funktiolla kuvion 34 mukaisesti.

```
2  $.post('resurssi.php', function(data) {
3      $('#esimerkki').html(data);
4  });
```

KUVIO 34. POST-pyynnön suorittaminen palvelimelle

$.ajax() on yleiskäyttöinen funktio Ajax-kutsujen toteuttamiseen. Sen avulla kutsulle voidaan määritellä paljon yksityiskohtaisemmat parametrit. Kaikki jQueryn Ajax-ominaisuudet käyttävät tätä funktiota. (Bibeault & Katz 2010, 261—278.)

5.2 jQuery UI

jQuery UI on jQueryn virallinen käyttöliittymäkirjasto. jQuery UI laajentaa käyttöliittymän mahdollisuuksia tuoden toimintoja, kuten kyvyn raahata ja pudottaa elementtejä sekä järjestää niitä ja muuttaa niiden kokoa. Kirjasto sisältää myös tyypillisiä käyttöliittymäkomponentteja, kuten välilehtiä, edistymispalkkeja, dialogilaatikoita ja päivämäärävalitsimia. (Bibeault & Katz 2010, 279, 282.)

CSS:llä on suuri painoarvo käyttöliittymäkomponenttien ja -toimintojen toteuttamisessa. jQuery UI:n mukana tulee valmis CSS-tiedosto, joka muodostaa kaikkien komponenttien ja niiden eri tilojen ulkoasun. (Bibeault & Katz 2010, 282.) Komponenttien ulkoasu on näin muokattavissa web-sovelluksen ulkoasuun sopivaksi.

5.2.1 Elementtien järjestely

Vuorovaikutus hiiren osoittimen kanssa on olennainen osa mitä tahansa graafista käyttöliittymää. Elementtien järjestely on yksi hyödyllisimmistä raahausta ja pudotusta hyödyntävistä toiminnoista. jQuery UI:n sortable()-metodi tekee HTML-listaelementistä järjesteltävän. Metodille välitetään lista asetuksista, joihin kuuluu mm. tapahtumankäsittelijöitä. Järjestelyoperaation käynnistämiin tapahtumiin kuuluvat mm. start, joka käynnistyy järjestelyn alkaessa, ja update, joka käynnistyy, kun järjestely on päättynyt ja elementtien järjestys on muuttunut. (Bibeault & Katz 2010, 322, 327—328.)

update-tapahtuman kohdalla halutaan usein tietää listan uusi järjestys. sortable('serialize') palauttaa elementtilistan id-arvoista serialisoidun merkkijonon, joka voidaan lähettää Ajaxilla. Listan jokaisen -elementin id-arvon on oltava muotoa ”etuliite_numero”, jossa etuliite voi olla mitä tahansa ja numero on elementin järjestysluku. Serialisoituun merkkijonoon säilötään tällöin etuliite-niminen taulukko, joka sisältää elementtien järjestysluvut uudessa järjestyksessä. (Bibeault & Katz 2010, 329.)

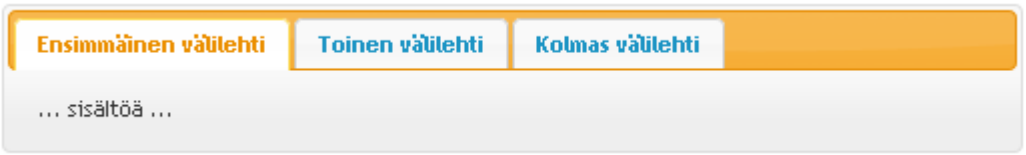
5.2.2 Välilehtikomponentti

Välilehdet ovat suosittu navigointimetodi, joiden avulla voidaan nopeasti selata erilaisia sisältökokonaisuuksia. Välilehtijoukon HTML-rakenne näyttää tyypillisesti kuvion 35 mukaiselta. Riviltä 1 alkava elementti sisältää välilehdet ja välilehtipaneelit. -elementin -elementit määrittävät välilehdet, ja seuraavat -elementit toteuttavat vastaavat paneelit. (Bibeault & Katz 2010, 389—390.)

```
<div id="tabset">
    <ul>
        <li><a href="#panel1">Ensimmäinen välilehti</a></li>
        <li><a href="#panel2">Toinen välilehti</a></li>
        <li><a href="#panel3">Kolmas välilehti</a></li>
    </ul>
    <div id="panel1">
        ... sisältöä ...
    </div>
    <div id="panel2">
        ... sisältöä ...
    </div>
    <div id="panel3">
        ... sisältöä ...
    </div>
</div>
```

KUVIO 35. Välilehtijoukon HTML-rakenne (Bibeault & Katz 2010, 390)

Välilehtijoukko luodaan tabs()-metodilla, joka kohdistetaan uloimpaan -elementtiin (Bibeault & Katz 2010, 391). Selaimessa välilehtijoukko näyttää muotoilusta riippuen esimerkiksi kuvion 36 mukaiselta.



KUVIO 36. Välilehtijoukko

Halutessa tabs()-metodille voidaan välittää esimerkiksi URL, josta välilehtien sisällöt ladataan dynaamisesti Ajaxilla (Bibeault & Katz 2010, 390). Tässä opinnäytetyössä hyödynnetään kuitenkin vain välilehtien tuomaa navigoinnin kätevyyttä ilman lisätoimintoja.

6 GOOGLE MAPS API

Google Maps JavaScript API -rajapinta mahdollistaa Google Maps -kartan upottamisen omalle web-sivulle. Se sisältää työkalut karttojen monipuoliseen käyttämiseen ja sisällön lisäämiseen niihin. (Google 2011.)

6.1 Toimintaperiaate

Google Maps V3 API -sovelluksen keskeisin elementti on itse kartta, jonka toteuttaa google.maps.Map-olio. Jotta kartta voidaan näyttää sivulla, sitä varten luodaan elementti, joka on tyypillisesti muotoa .

```
2  $(document).ready(function() {
3      var latlng = new google.maps.LatLng(60.983, 25.651);
4      var myOptions = {
5          zoom: 8,
6          center: latlng,
7          mapTypeId: google.maps.MapTypeId.ROADMAP
8      };
9      var map = new google.maps.Map($("#map_canvas").get(0),
10         myOptions);
11 }
```

KUVIO 37. Google Maps -kartan määrittäminen

Kuvion 37 rivillä 3 luodaan latlng-arvo, joka määrittää koordinaatit, joihin kartta keskitetään. Rivillä 4 asetetaan kartan asetukset: zoomauksen taso, kartan keskikohta ja kartan tyyppi. ROADMAP näyttää tavallisen kaksiulotteisen kartan; toinen vaihtoehto on esimerkiksi satelliittikuva. Rivillä 9 luodaan kartta, jonka muodostimelle välitetään HTML-elementti ja asetukset. (Google 2011, Tutorial.)

google.maps.LatLng-olio on tapa viitata sijainteihin kartalla. Sille välitetään leveys- ja pituusaste. LatLng-olioita käytetään esimerkiksi merkitsimien ja muiden kuvien sijoittamiseen kartalle haluttuun maantienteelliseen sijaintiin. (Google 2011, Tutorial.)

6.2 Tapahtumat

Google Maps -rajapinnan tapahtumamalli sisältää kahdenlaisia tapahtumia: käyttäjätapahtumat, kuten hiiren klikkaus, ja kartan tilan muutokset, kuten zoomausasteen muutos. Tapahtumankäsittelijä rekisteröidään addListener()-metodilla kuvion 38 mukaisesti. (Google 2011, Events.)

```
12      google.maps.event.addListener(map, 'zoom_changed', function() {
13          var darwin = new google.maps.LatLng(-12.461334, 130.841904);
14          map.setCenter(darwin);
15      });
16
17      var marker = new google.maps.Marker({
18          position: latlng,
19          map: map,
20          title: "Hello World!"
21      });
22      google.maps.event.addListener(marker, 'click', function() {
23          map.setZoom(8);
24      });
```

KUVIO 38. Tapahtumankäsittelijöiden rekisteröiminen (Google 2011, Events)

Rivillä 12 asetetaan tapahtumankäsittelijä reagoimaan kartan zoom-ominaisuuden muutoksiin; tapahtuman seurauksena kartta keskitetään Darwiniin, Australiaan. Rivillä 22 lisätään luotuun merkitsimeen tapahtumankäsittelijä, joka asettaa klikkauksesta kartan zoomauksen. (Google 2011, Events.)

6.3 Peittokuvat

Peittokuvat ovat kartan päällä näytettäviä olioita, jotka on sidottu tiettyihin leveys- ja pituusasteisiin. Tyypillisin peittokuva on merkitsin, joka edustaa yhtä sijaintia kartalla. Edellä olevassa kuviossa 38 luotiin rivillä 17 merkitsin oletuskuvakkeella. Oma kuvake määritetään asettamalla merkitsimen icon-ominaisuuden arvoksi kuvan URL. Monimutkaisempi kuvake voidaan luoda MarkerImage-oliolla, jolloin voidaan asettaa esimerkiksi kuvakkeen ankkurikohta. (Google 2011, Overlays.) Kuviossa 38 nähdään esimerkki monimutkaisemman kuvakkeen määrittämisestä.

```
    var icon = new google.maps.MarkerImage(
        'images/icon.png',
        new google.maps.Size(31, 32), //mittasuhteet
        new google.maps.Point(0,0), //origo
        new google.maps.Point(12, 12) //ankkuri
    );
```

KUVIO 39. Merkitsimen kuvakkeen määrittäminen

On myös mahdollista luoda mukautettuja peittokuvia OverlayView-rajapinnan avulla. Kuvion 40 esimerkkitoteutus piirtää annetun kuvan annettujen koordinaattien välille.

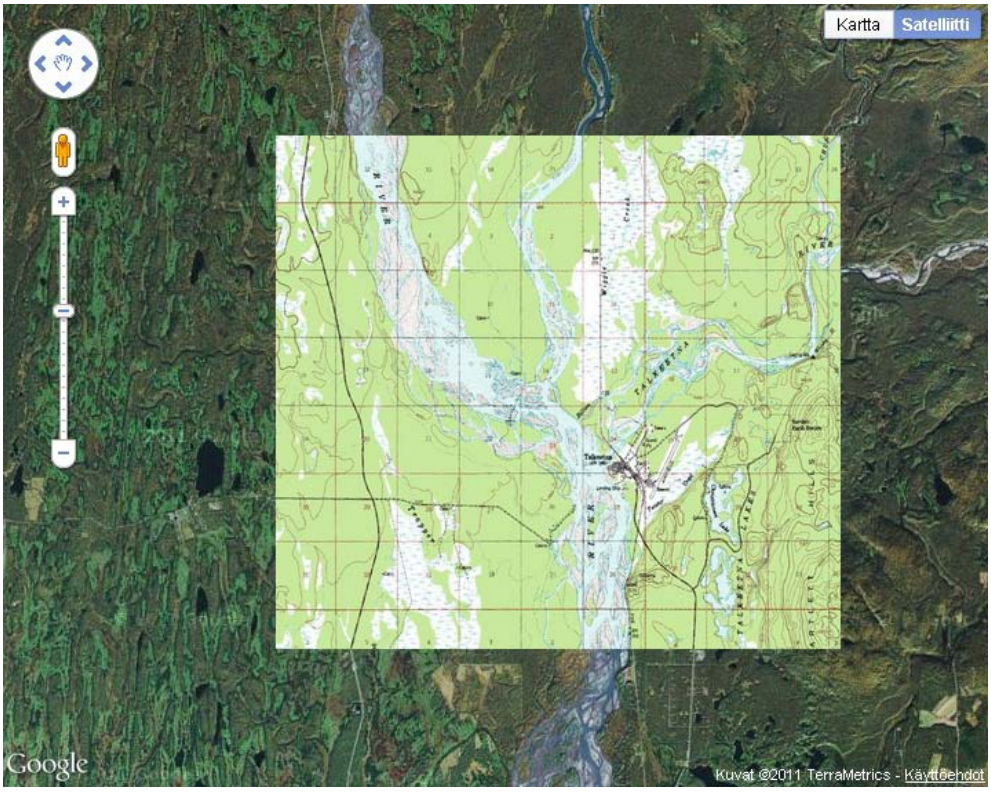
```
function USGSOverlay(bounds, image, map) {
    this.bounds_ = bounds;
    this.image_ = image;
    this.map_ = map;

    this.div_ = null;

    this.setMap(map);
}

USGSOverlay.prototype = new google.maps.OverlayView();

USGSOverlay.prototype.onAdd = function() {
    var div = document.createElement('DIV');
    div.style.position = "absolute";

    var img = document.createElement("img");
    img.src = this.image_;
    img.style.width = "100%";
    img.style.height = "100%";
    div.appendChild(img);

    this.div_ = div;

    var panes = this.getPanes();
    panes.overlayLayer.appendChild(div);
}

USGSOverlay.prototype.draw = function() {
    var overlayProjection = this.getProjection();

    var sw = overlayProjection.fromLatLngToDivPixel(this.bounds_.getSouthWest());
    var ne = overlayProjection.fromLatLngToDivPixel(this.bounds_.getNorthEast());

    var div = this.div_;
    div.style.left = sw.x + 'px';
    div.style.top = ne.y + 'px';
    div.style.width = (ne.x - sw.x) + 'px';
    div.style.height = (sw.y - ne.y) + 'px';
}
```

KUVIO 40. Mukautettu peittokuva (Google 2011, Overlays)

Ensimmäisellä rivillä määritetään USGSOverlay-nimellä nimetyn luokan muodostin. Muodostimelle välitetyt parametrit asetetaan olion ominaisuuksiksi. Rivillä 11 omasta luokasta tehdään OverlayView-luokan aliluokka. Kun peittokuva on lisätty karttaan, kutsutaan automaattisesti onAdd()-metodia. Rivillä 14 luodaan -elementti, johon kuva asetetaan. Tämän jälkeen luodaan <img>-elementti, joka liitetään -elementtiin. Rivillä 26 peittokuva liitetään karttaan.

Google Maps API kutsuu peittokuvan draw()-metodia, kun peittokuva on piirrettävä karttaan. Riveillä 32 ja 33 noudetaan peittokuvan maantieteelliset koordinaatit ja muutetaan ne vastaaviksi kuvapistekoordinaateiksi. Kuvan -elementin suhteita muutetaan tämän jälkeen vastaamaan saatuja arvoja. (Google 2011, Overlays.)

Kuviossa 41 nähdään, miten peittokuva renderoituu kartan päälle. Esimerkissä erillinen alueen kartta ankkuroituu asetettuihin koordinaatteihin ja muuttaa kokoaan ja sijaintiaan Google Maps -kartan mukana.



KUVIO 41. Peittokuva käytännössä (Google 2011, Overlays)

6.4 Kartan muotoilu

Google Maps -kartan perusmallin ulkoasua voidaan mukauttaa StyledMapType-luokan avulla. Kartta koostuu joukosta ominaisuuksia, kuten tiet, puistot ja vesistöt. Ominaisuustyypit muodostavat kategoriapuun, jolloin kategorian tyylin muokkaaminen vaikuttaa myös kaikkiin alakategorioihin.

Kaikkiin elementteihin liittyy piirrettävä geometrinen kuvio (geometry), ja esimerkiksi teihin myös tien nimi (labels). Kartan elementtien muotoilussa voidaan asettaa mm. elementin värisävy, kirkkaus, värikylläisyys ja se, näytetäänkö elementti kartalla.

```
1  var stylez = [
2      {
3          featureType: "road.local",
4          elementType: "geometry",
5          stylers: [
6              { hue: "#00ff00" },
7              { saturation:100 }
8          ]
9      },
10     {
11         featureType: "landscape",
12         elementType: "geometry",
13         stylers: [
14             { lightness: -100 }
15         ]
16     }
17 ];
18
19 var styledMapOptions = {
20     name: "Hip-Hop"
21 }
22
23 var jayzMapType = new google.maps.StyledMapType(
24     stylez, styledMapOptions);
25
26 map.mapTypes.set('hiphop', jayzMapType);
27 map.setMapTypeId('hiphop');
```

KUVIO 42. Kartan ulkonäön muotoilu (Google 2011, Map Types)

Kuvion 42 koodi muuttaa kartan tiet kirkkaan vihreäksi ja asuma-alueet mustaksi. Riveillä 3 ja 4 valitaan paikallisten teiden geometria, ja riveillä 5—8 asetetaan valitun kohteen värisävy ja värikylläisyys. Luotu muotoilu lisätään karttaan rivillä 26, ja otetaan käyttöön rivillä 27. (Google 2011, Map Types.)

7 LAHTI360.FI

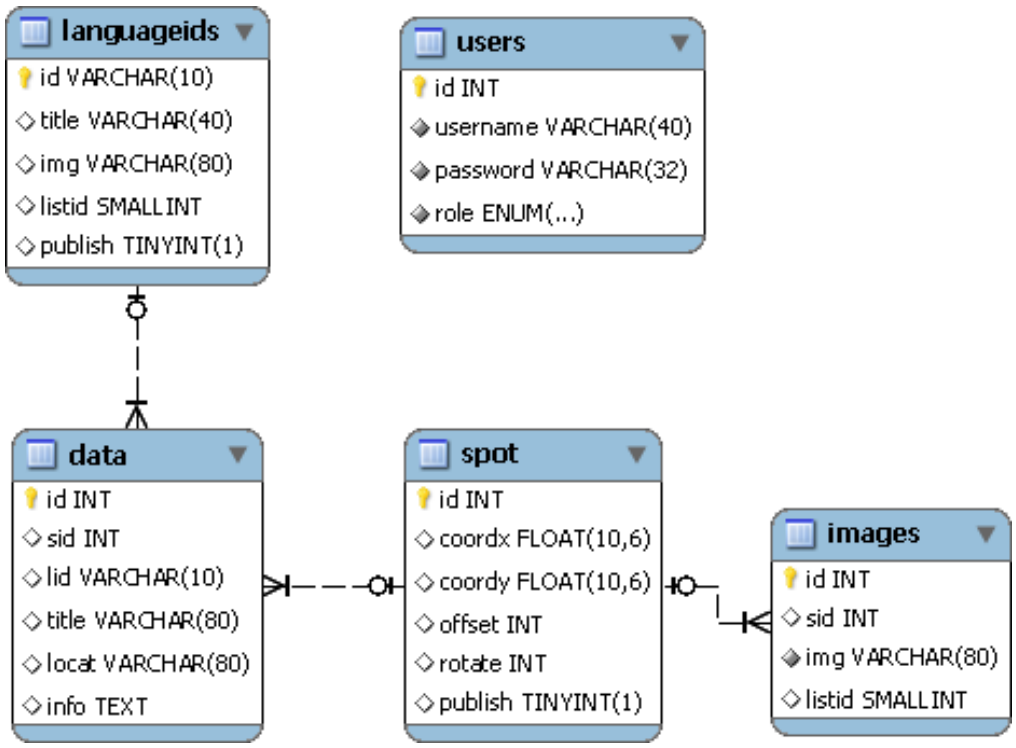
7.1 Sivuston rakenne

Sivusto muodostuu seuraavista alasivuista:

- Etusivu
- Sisäänkirjautuminen
- Kohteiden hallinta
    - Kohteen lisääminen/muokkaaminen
    - Kohteen tarkastelu
- Käyttäjien hallinta
    - Käyttäjän lisääminen/muokkaaminen

7.2 Tietokanta

Tietokannan hyvä toteutus on olennainen osa web-sivuston toimintaa. Tässä työssä käytetty tietokantamalli ja taulujen väliset relaatiot on kuvattu kuviossa 43.

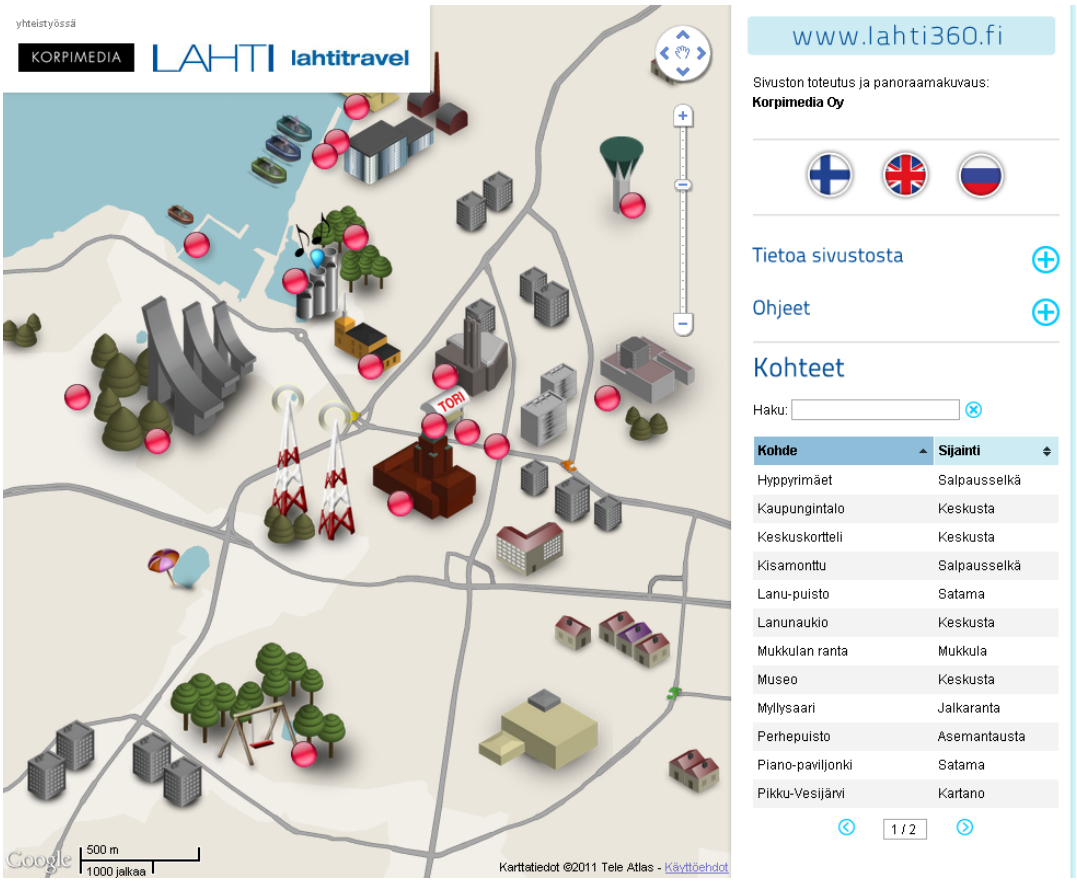


KUVIO 43. Tietokannan rakenne

Sovelluksen kannalta keskeisin taulu on spot, joka sisältää sivuston kohteet. data-taulu sisältää kohteiden tekstit eri kielillä; yhdellä kohteella voi siis olla useita tekstejä, mutta teksti voi kuulua vain yhteen kohteeseen. Kohteeseen voi myös kuulua 0—n kuvaa. Sivuston kielet on tallennettu languageids-tauluun, joka sisältää kielten nimet ja kuvakkeiden URL:t. spot- ja languageids-taulujen publish-kenttä osoittaa, onko kyseinen tietue saatavilla julkisella sivulla. Näin tietyt kohteet ja kielet voidaan piilottaa käyttäjiltä. listid-kenttä määrää kuvan tai kielen järjestysnumeron listauksessa.

7.3 Toteutus

Lahti360.fi-sivuston etusivu nähdään kuviossa 44. Tämä on ainoa sivu, joka kiinnostaa tavallista kävijää. Käyttäjä, jolla on tunnukset järjestelmään, pääsee hallintapaneeliin login-sivun kautta (kuvio 45).



KUVIO 44. Lahti360.fi-etusivu

KUVIO 45. /login-sivu

7.3.1 Käyttöoikeudet

Kirjautumissivun toteuttaa Authentication-ohjaimen login-toiminto sitä vastaavan näkymän kanssa. Esilataajaluokassa asetetaan reitityssääntö, joka lyhentää URI:n ”/authentication/login” muotoon ”/login”. Vastaavasti logout-toimintoon päästään lyhennetysti osoitteesta ”/logout”.

Kun käyttäjä lähettää lomakkeen, tunnusten oikeellisuus varmistetaan Zend_Auth_Adapter_DbTable-oliolla. Mikäli tietokannan users-taulussa ei ole annettuja tunnuksia, lomakkeen kuvaukseksi asetetaan virheviesti lomakkeen setDescription()-metodilla. Muussa tapauksessa käyttäjän id, käyttäjänimi ja rooli varastoidaan istuntoon, ja käyttäjä ohjataan hallintapaneelin sisällön muokkaussivulle.

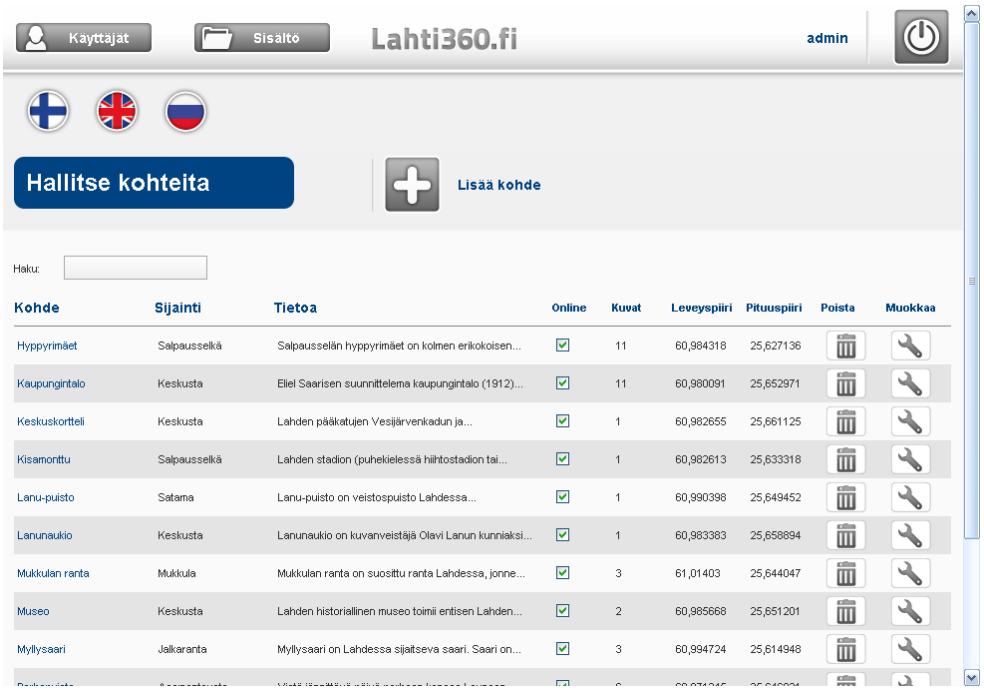
Käyttäjärooleja ovat ylläpitäjä, muokkaaja ja vieras. Roolit, resurssit ja käyttöoikeudet asetetaan esilataajassa (kuvio 46). Vieraalla on oikeus vain etusivun käyttämiseen ja kielen vaihtamiseen, muokkaaja pääsee lisäksi sisällön hallintasivulle, ja ylläpitäjällä on oikeudet kaikkialle.

```
42          $acl->add(new Zend_Acl_Resource('content'));
43          $acl->add(new Zend_Acl_Resource('authentication'));
44          $acl->add(new Zend_Acl_Resource('admin'));
45          $acl->add(new Zend_Acl_Resource('language'));
46
47          $acl->addRole(new Zend_Acl_Role('guest'));
48          $acl->addRole(new Zend_Acl_Role('editor'), 'guest');
49          $acl->addRole(new Zend_Acl_Role('admin'), 'editor');
50
51          $acl->allow('guest', array('authentication', 'language'));
52          $acl->allow('editor', 'content');
53          $acl->allow('admin');
```

KUVIO 46. ACL-asetukset

7.3.2 Sisällön hallinta

Sisällön hallintasivun toteuttaa Content-ohjain, jonka index-toiminnon näkymä nähdään kuviossa 47. Tietokantaoperaatiot suoritetaan ContentMapper-nimisessä mallissa, jonka fetchAll()-metodia ohjain käyttää hakeakseen kaikkien kohteiden tiedot ja niihin liittyvät käännöstekstit ja kuvat. Metodi luo jokaisesta kohteesta edellä mainitut tiedot sisältävän Content-olion ja palauttaa oliojoukon taulukossa. Taulukko välitetään näkymälle, joka muodostaa tiedoista HTML-taulukon.

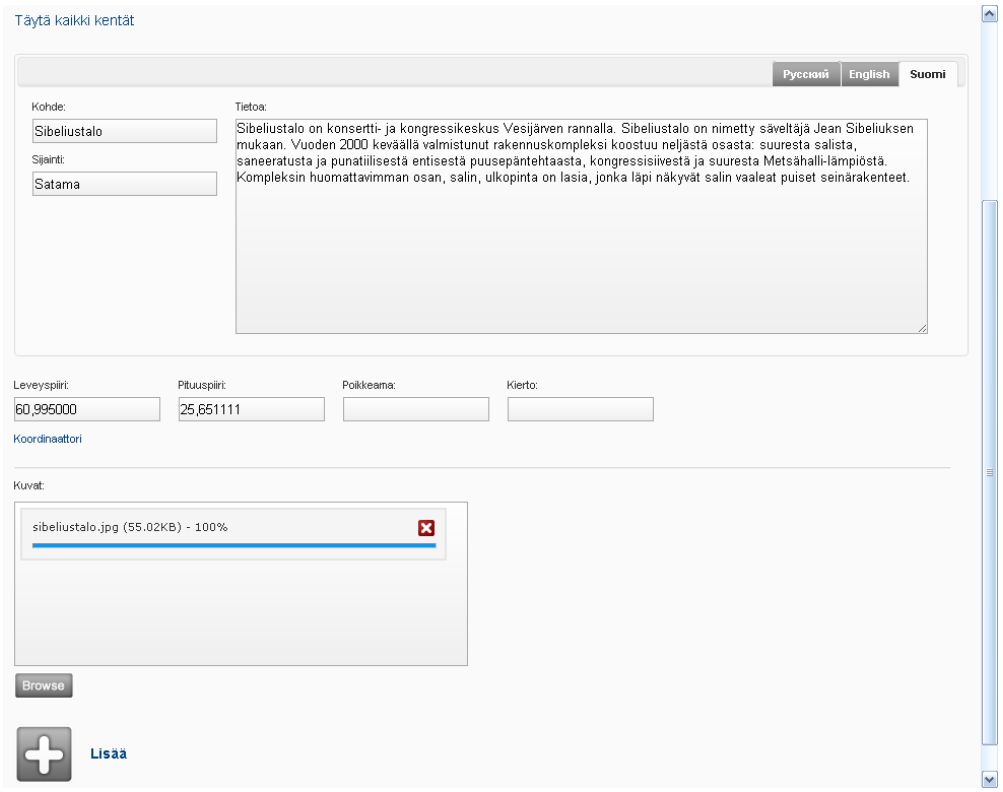


KUVIO 47. Kohteiden listaus ja niiden hallinta (/content)

HTML-taulukko muutetaan dynaamiseksi jQueryn DataTables-lisäosalla, minkä jälkeen taulukko voidaan järjestää aakkosjärjestykseen minkä tahansa sarakkeen mukaan klikkaamalla sarakkeen otsikkoa. DataTables myös jakaa taulukon tarvittaessa useammalle sivulle ja mahdollistaa näkyvien rivien suodattamisen Hakukentän avulla. Kohteen poisto tapahtuu Ajax-kutsulla, jossa yksinkertaisesti välitetään halutun kohteen id ohjaimen delete-toiminnolle, joka pyytää ContentMapper-mallia poistamaan kohteen tietokannasta. Ajax-kutsun palatessa kyseessä oleva rivi häivytetään näkymättömiin ja sitten poistetaan taulukosta. Kohteen online-valintaruudun klikkaaminen muokkaa niin ikään Ajaxin välityksellä kohteen näkyvyysarvoa tietokannassa.

Kohteen lisäämissivu on kuvattu kuviossa 48. Uuden kohteen lisääminen määritetään URL:ssa mode-parametrin arvolla ”add”. Lomake on periytetty ZendX_JQuery_Form-luokasta, joka tavallisen lomakkeen toiminnallisuuden lisäksi sisältää jQuery-käyttöliittymäkomponentteja, kuten välilehdet. Kielikohtaiset kentät on asetettu välilehtien sisään. Kohde- ja sijainti-kentille on asetettu 80 merkin maksimipituus StringLength-validaattorilla, ja leveys- ja pituuspiirien on oltava liukulukuja. Syötteistä siistitään turhat välilyönnit StringTrim-suodattimella. Koordinaattori-linkki avaa modaalisen ikkunan (absoluuttisesti sivun päälle sijoitettu -elementti), jossa Google Maps -kartan avulla voidaan helposti valita halutut koordinaatit.

Kuvatiedostojen lähettämistä varten lomakkeessa on neljä tavallista <input type="file">-elementtiä. Ne kuitenkin korvataan JavaScriptillä sivun latautuessa Flash-pohjaisella lataajalla, jonka toteuttaa Uploadify-niminen jQuery-lisäosa. Flash-sovellus mahdollistaa useamman tiedoston valitsemisen ladattavaksi samanaikaisesti, ja sillä lataus palvelimelle tapahtuu heti, jo ennen lomakkeen lähettämistä. Ladattujen kuvatiedostojen nimet lisätään lomakkeeseen muiden tietojen kanssa lähetettäväksi.



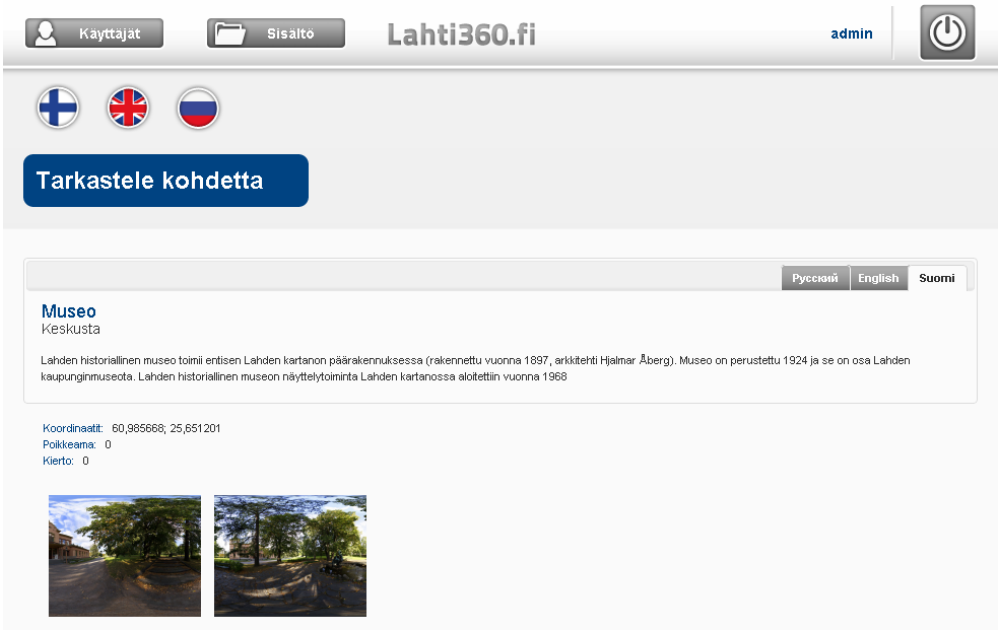
KUVIO 48. Kohteen lisääminen (/content/update/mode/add)

Ohjaimen PHP-skripti luo lähetetyistä arvoista Content-olion ja välittää sen ContentMapper-mallin save()-metodille. save()-metodi lisää ensin kohteen koordinaatit ja panoraamakuvan poikkeaman ja kierron spot-tauluun ja käyttää saatua id:tä lisätessään käännöstekstit ja kuvat data- ja images-tauluihin. Ladatut kuvat nimetään uudelleen kohteen mukaan uniikilla nimellä ja siirretään väliaikaishakemistosta web-palvelimen juuren photos-hakemistoon.

Olemassaolevan kohteen muokkaaminen vastaa muuten kohteen lisäystä, mutta tällöin URL:ssa välitetään mode-parametrina ”edit” ja id-parametrina kohteen id. Muokattavan kohteen tiedot ladataan ja täytetään lomakkeen kenttiin. Lisäksi sivulle listataan kohteen kuvien esikatselukuvat (kuvio 49), joiden järjestys on muokattavissa ja uusi järjestys päivitetään Ajaxilla tietokantaan. Kuvia voidaan myös poistaa samalla periaatteella kuin kohteita.

KUVIO 49. Kohteen kuvien muokkaus

Sisällön hallintasivulla kohteen nimeä klikkaamalla päästään tarkastelemaan kohdetta (kuvio 50). Kielikohtaiset tiedot esitetään jälleen välilehtikomponentissa. Kohteen kuvat näytetään esikatselukuvina, joita klikkaamalla aukeaa suurempi kuva jQueryn Fancy Zoom -lisäosaa hyväksikäyttäen.



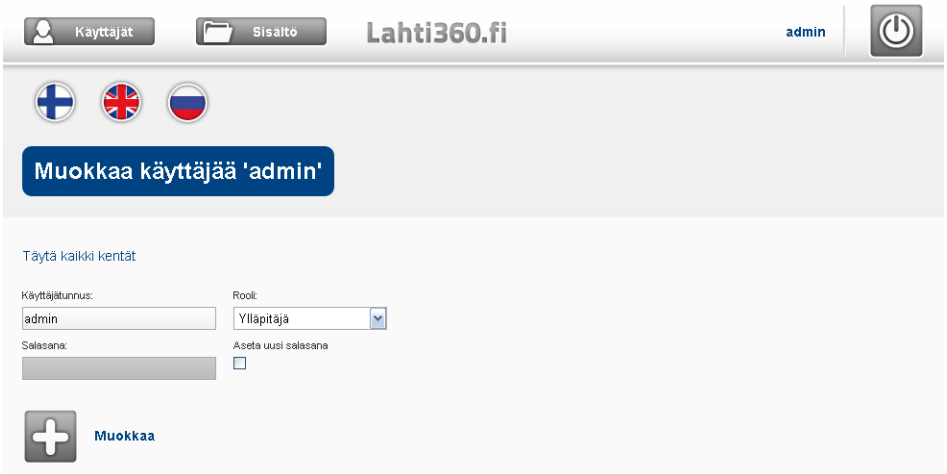
KUVIO 50. Kohteen tarkasteleminen (/content/view/id/10)

7.3.3 Käyttäjien hallinta

Ylläpitäjät hallitsevat käyttäjiä Admin-ohjaimen toteuttamalla sivulla (kuvio 51). Sivulla on sama periaate kuin sisällön hallinnan sivulla; ohjain hakee tietokannasta listan käyttäjistä UsersMapper-mallin avulla ja välittää sen näkymälle.

KUVIO 51. Käyttäjien listaus ja niiden hallinta (/admin)

Käyttäjän lisääminen ja muokkaus erotetaan Admin-ohjaimen user-toiminnossa toisistaan mode-parametrin arvoilla ”add” ja ”edit”. Lomakkeessa (kuvio 52) asetetaan käyttäjän tunnus, salasana ja rooli. Käyttäjän muokkauksessa salasana voidaan jättää huomiotta ja muokata vain käyttäjätunnusta tai roolia.



KUVIO 52. Käyttäjän muokkaaminen (/admin/user/mode/edit/id/1)

7.3.4 Internationalisointi

Sivun kääntäminen muille kielille on toteutettu, kuten on kuvattu Zend Framework -luvussa. Sivun käännökset on varastoitu CSV-tiedostoihin web-palvelimen juuren ulkopuolelle languages-hakemistoon, josta ne ladataan esilataajassa. Aluksi kielen vaihtaminen toteutettiin erillisen Language-ohjaimen avulla, joka talletti valitun kielen istuntoon. Myöhemmin tämä muutettiin niin, että kieli välitetään URL:ssa. Näin sivuston ulkopuolella voidaan linkittää suoraan halutun kieliseen versioon.

7.3.5 Etusivu

Etusivulla luodaan Google Maps -kartta, joka keskitetään Lahden keskustaan. Kartta muotoillaan niin, että mm. teiden ja kaupunginosien nimet ja pienet tiet piilotetaan, ja värit muutetaan hillitymmäksi. Karttaan lisätään 63 Monument-oliota, jotka ovat google.maps.OverlayView-rajapinnan toteuttavia peittokuvia. Karttaan asetetaan myös bounds_changed-tapahtumankuuntelija, joka reagoi sijainnin muutoksiin eikä anna käyttäjän poistua Lahden alueelta, ja zoom_changed-kuuntelija, joka pitää zoomausasteen aina 12 ja 16 välillä.

Index-ohjaimen xml-toiminto generoi XML-tiedoston, joka sisältää kaikki kohteet teksteineen ja linkkeine kuviinsa. Etusivun JavaScriptissä XML-tiedosto ladataan jQueryn post()-funktiolla (kuvio 53).

```
130     $.post("/index/xml", {}, function(xml){
131         $(xml).find("spot").each(function() {
132             var spot = $(this);
133 
134             var title = spot.find('title').text();
135             var location = spot.find('locat').text();
136             var info = spot.find('info').text();
137             var position = new google.maps.LatLng(
138                 parseFloat(spot.find("lat").text()),
139                 parseFloat(spot.find("lng").text())
140             );
141             var images = new Array();
142             spot.find('image').each(function() {
143                 images.push($(this).text());
144             });
145             var rotate = spot.find('rotate').text();
```

KUVIO 53. Kohteiden lataaminen dynaamisesti

Jokaiselle kohteelle luodaan karttaan merkitsin, joka asetetaan kuvion 53 rivillä 137 määritettyihin position-koordinaatteihin. Merkitsimelle asetetaan kuvake ja nk. työkaluvihje, johon asetetaan kohteen nimi ja sijainti ja joka näytetään käyttäjän viedessä hiiren merkitsimen päälle. Merkitsimeen määritetään click-tapahtumankuuntelija, joka klikkauksesta avaa jQueryn jqModal-lisäosalla toteutetun modaalisen ikkunan, johon se asettaa kohteen nimen ja kuvauksen. Samassa ikkunassa käytetään krpano-nimistä panoraamakuvien katseluohjelmaa kohteen kuvien esittämiseen.

8 YHTEENVETO

Työn tarkoituksena oli uusia Lahti360.fi-sivuston taustajärjestelmä täysin PHP-pohjaiseksi ja tuoda MySQL-tietokanta tietojen varastoimiseen. Toteutukseen käytettiin Zend Framework -ohjelmistokehystä. Se tuo sovellukseen vankan, toimivan pohjan, jonka päälle voidaan rakentaa näyttävä käyttöliittymä jQueryn kirjastoilla. Sivuston dynaamisuudella oli painoarvoa, joten suuressa osassa toiminnoista hyödynnettiin mm. Ajaxia.

Käytetty ohjelmistokehys oli sovelluksen kehitysvaiheessa vielä uutta, joten osa asioista toteutettiin vaikeamman kautta ja kaikkien suunnittelumallien täyttä potentiaalia ei hyödynnetty. Esimerkiksi käyttöliittymän internationalisointiin on Zend Frameworkissa näkymäapuri, joka tuo käännökset näkymien käyttöön automaattisesti ja jota ei käytetty. Sen sijaan käännökset ladattiin rekisteristä ohjaimen kautta näkymään. Lisäksi kaikki tulostus tulisi suorittaa optimaalisesti näkymissä ja kaikki tiedon prosessointi malliluokissa, tässä työssä näitä toimintoja suoritettiin paikoin ohjainluokassa.

Opinnäytetyön toteuttamiseen tutkimustyö mukaan luettuna kului aikaa hieman yli kuukausi, jonka jälkeen sivusto siirrettiin kehitysympäristöstä tuotantopalvelimelle. Pieniin korjauksiin ja viilauksiin kului tämän jälkeen noin kaksi viikkoa. Kaikki tarvittava toiminnallisuus saatiin implementoitua.

Lahti360.fi-sivuston seuraavan version kehityskohteisiin voisi kuulua käyttöliittymän muokkaaminen entistä dynaamisemmaksi mm. muuttamalla kaikki lomakkeet Ajax-pohjaisiksi. Lisäksi käyttäjien hallinnan kehittämisessä on potentiaalia, nykyisellään se on toteutettu erittäin yksinkertaisesti. Pidemmälle kehitetyssä järjestelmässä käyttäjät voisivat vaihtaa oman salasanansa, käyttäjien tekemät sisältömuutokset ja mahdolliset väärinkäytökset olisivat jäljitettävissä, ja käyttäjistä voitaisiin tallettaa joitakin yhteystietoja.

LÄHTEET

Allen, R., Lo, N. & Brown S. 2009. Zend Framework in Action. Greenwich, CT, Yhdysvallat: Manning Publications.

Bibeault, B. & Katz, Y. 2010. jQuery in Action. 2. uudistettu painos. Stamford, CT, Yhdysvallat: Manning Publications.

Google. 2011. Google Maps JavaScript API V3 [viitattu: 13.3.2011]. Saatavissa: ://code.google.com/intl/fi-FI/apis/maps/documentation/javascript/.

Häkkinen, P. 2009. Selainohjelmointi: DOM ja JavaScript. Tampereen teknillinen yliopisto [viitattu 3.4.2011]. Saatavissa: ://matriisi.ee.tut.fi/hmopetus/johd-hm/2009/luennot/johd-hm2009-6-dom-ja-javascript.pdf.

jQuery. 2011. jQuery [viitattu 12.3.2011]. Saatavissa: ://jquery.com/.

Koch, P. 2007. W3C DOM - Introduction. QuirksMode.org [viitattu: 3.4.2011]. Saatavissa: ://www.quirksmode.org/dom/intro.html.

Olsson, T. & O'Brien, P. 2011. CSS Layout and Formatting. SitePoint Pty. Ltd. [viitattu 3.4.2011]. Saatavissa: ://reference.sitepoint.com/css/csslayout.

Stafford, L. 2010. Rapid Web Development Using PHP Frameworks. EzineArticles.com [viitattu 28.2.2011]. Saatavissa: ://ezinearticles.com/?Rapid-Web-Development-Using-PHP-Frameworks&id=5499562.

Wikipedia. 2011a. JavaScript [viitattu 28.2.2011]. Saatavissa: ://en.wikipedia.org/wiki/JavaScript.

Wikipedia. 2011b. PHP [viitattu 1.3.2011]. Saatavissa: ://en.wikipedia.org/wiki/PHP.

Wikipedia. 2011c. Web application framework [viitattu 1.3.2011]. Saatavissa:

://en.wikipedia.org/wiki/Web_application_framework.

Wikipedia. 2011d. Web development [viitattu 1.3.2011]. Saatavissa: ://en.wikipedia.org/wiki/Web_development.

Zend Technologies Ltd. 2011. Programmer's Reference Guide [viitattu 4.3.2011]. Saatavissa: ://framework.zend.com/manual/en/manual.html.