Johannes Vuorela

JAVASCRIPT FULL STACK -KEHITYS

Case: Makeko-arviointipalvelu

Opinnäytetyö
Tietojenkäsittelyn koulutusohjelma

Marraskuu 2016

KUVAILULEHTI

MAMK University of Applied Sciences	**Opinnäytetyön päivämäärä** 30.11.2016
Tekijä(t) Johannes Vuorela	**Koulutusohjelma ja suuntautuminen** Tietojenkäsittely

Nimeke

Javascript Full Stack -kehitys, Case: Makeko-arviointipalvelu

Tiivistelmä

Verkkopalveluiden ohjelmointi on lisännyt suosiotaan lähivuosina. Verkkosivujen toteuttaminen koki vallankumouksen, kun kehittäjät ottivat HTML5-kielen laaja-alaisesti käyttöön. Sen rinnalla JavaScript-kieli on tätä nykyä suositumpi kuin koskaan. JavaScript luotiin alun alkaen lisäämään interaktiivisuutta verkkosivujen käyttöliittymäpuolella. Kuitenkin vuonna 2009 Ryan Dahl kehitti Node.js-nimeä kantavan kehitysympäristön, jolla JavaScript-kieli pystyttiin ensimmäistä kertaa historiassa esittelemään myös palvelinpuolelle. Tämä mahdollisti verkkosovellusten sekä käyttöliittymäpuolen että palvelinpääpuolen toteuttamisen kyseisellä kielellä, mikä tarkoittaa myös sitä, että riittävällä JavaScript-kielen osaamisella voi yksittäinen henkilö toteuttaa palvelun molemmat osapuolet, jolloin puhutaan ns. ”Full Stack” -kehitystyöstä.

Tämän työn tavoitteena on tutustua kokonaisen verkkopalvelun toteuttamisen eri vaiheisiin JavaScript-kielellä Node.js-ympäristöä sekä sen tukemia muita moderneja tekniikoita apuna käyttäen sekä toteuttaa kyseisillä tekniikoilla Sanoma Pro Oy:lle toimeksiantona Makeko-verkkopalvelu. Palvelun tarkoitus on mahdollistaa digitaalisten lähtötasotestien järjestäminen kouluissa sekä tilastoida oppilaiden testisuorituksia, joita tultaisiin myöhemmin tilastoimaan ja täten vertailemaan keskenään valtakunnallisella tasolla.

Asiasanat (avainsanat)

JavaScript, Ajax-ohjelmointi, verkko-ohjelmointi, HTML, XML

Sivumäärä 27	**Kieli** Suomi

Huomautus (huomautukset liitteistä)

Ohjaavan opettajan nimi Janne Turunen	**Opinnäytetyön toimeksiantaja** Sanoma Pro Oy

DESCRIPTION

MAMK University of Applied Sciences	**Date of the bachelor's thesis** 30.11.2016
Author(s) Johannes Vuorela	**Degree programme and option** Computer science

Name of the bachelor's thesis

JavaScript Full Stack development, Case: Makeko service

Abstract

Programming web services has increased its popularity in recent years. Website development experienced a revolution when HTML5 was adopted widely among developers. Alongside, the JavaScript language is nowadays more popular than ever. JavaScript was originally created to increase the interacivity of user interfaces on websites. However, in the year 2009 Ryan Dahl developed a development environment by the name Node.js, with which one could introduce JavaScript language also to the serverside, for the first time in history. This made it possible to make both the user interface part of the web service as well as the serverside with that particular language. This means that with enough knowledge of JavaScript one person could develop both sides, in which case we are talking about the so called Full Stack development.

The goal of this thesis was to learn about different parts of developing a whole web service with JavaScript language while making use of the Node.js environment as well as other modern techniques supported by it and to develop the Makeko web service as an assignment for Sanoma Pro Oy with the specified techniques. The purpose of the service was to make it possible to arrange digital starting level tests in schools as well as to compile statistics of pupils' test results that would be later made into statistics and thus compared with the other results in a national scale.

Subject headings, (keywords)

JavaScript, Ajax programming, web programming, HTML, XML

Pages 27	**Language** Finnish

Remarks, notes on appendices

Tutor Janne Turunen	**Bachelor's thesis assigned by** Sanoma Pro Oy

SISÄLTÖ

1 JOHDANTO

Opinnäytetyön tavoitteena on toteuttaa Sanoma Pro Oy:lle Makeko-niminen palvelu, jolla opettajat ympäri Suomen voivat järjestää matematiikan valtakunnallisia lähtötasotestejä oppilailleen, joiden tulokset sitten tilastoidaan. Tässä työssä keskitytään opettajien käyttöliittymän vaatimuksiin.

Sanoma Pro Oy on Sanoma-konserniin kuuluva yhtiö, jonka liiketoiminta perustuu oppimateriaalien tuottamiseen ja jakeluun. Nykyään Sanoma Pron päätehtäviin kuuluu digitaalisen oppimateriaalien tuottaminen sekä jo olemassa fyysisen materiaalin digitalisointi. Sanoma Pron tuotteisiin sekä palveluihin lukeutuu mm. koulukirjat, koemateriaalit, oppimispelit sekä oppimismateriaalin tuottamiseen laaditut työkalut. Sanoma Prolla on toimipisteet Helsingissä sekä Mikkelissä. Mikkelin yksikön vuoropäällikkönä toimii Raija Komppa-Rannaste.

Opinnäytetyöni raportti alkaa palvelun suunnittelulla, jonka käyn läpi luvussa 2. Suunnittelun tavoitteena oli saada toimeksiantajan antamien vaatimusten pohjalta riittävän kattava suunnitelma siitä, millainen lopputuote tulee lopulta olemaan.

Luvussa 3 aloitetaan palvelun toteuttaminen ohjelmointitekniikoin syntyneen suunnitelman pohjalta. Samalla selostan, mitä tekniikoita käytin, ja miten sovelsin niitä työssä. Olen jakanut toteuttamisen tässä työssä 6 osaan. Aloitin työn toteuttamalla siihen etusivun käyttöliittymän sisäänkirjautumisominaisuuksineen. Jatkoin kehitystyötä laatimalla sisään kirjautuneen opettajan etusivunäkymän. Kerron näistä kahdesta sivusta tarkemmin luvussa 3.1. Seuraavaksi kehitin koetehtäväpankin varsinaisen koe-editorinäkymän, jonka toteutuksesta kerron luvuissa 3.2 ja 3.4. Luku 3.3 kertoo siitä, kuinka tein työkalusta astetta tietoturvallisemman. Lopuksi tein tietokantaan liittyvät muokkaustyökalut sekä opettajille että ylläpitäjille, joista kerron luvuissa 3.5 ja 3.6.

Lopuksi käyn aikaansaamani lopputuotoksen kiteytetysti läpi luvussa 4. Samalla kerron, mitä ajatuksia se sekä koko tämän työn tuotteistamisprosessi minussa herätti. Käyn myös läpi, miten palvelua tullaan jatkokehittämään tulevaisuudessa.

2 PALVELUN SUUNNITTELU

Lähtötasotesti on koe, jolla kartoitetaan oppilaan osaamista. Lähtötasotesti poikkeaa tavanomaisesta kokeesta sillä, ettei se ole sidoksissa yhteen ajankohtaiseen kurssiin vaan sillä selvitetään opiskelijan valmius jossakin aineessa yleisellä tasolla. Testejä ei arvostella niinkään perinteisesti pisteyttämällä, vaan niiden lopputuloksessa huomioidaan ne aihealueet, jossa ao. opiskelija tarvitsee tukea ja harjoitusta. Lähtötasotestejä pidetään myös lukuvuosittain ympäri Suomea peruskoululaisille. Näitä testejä pystyi ennen vanhaan tilaamaan Sanoma Prolta Makeko-ohjelman kautta, mutta nyttemmin kyseinen kanava on lakkautettu. Opinnäytetyöni tarkoitus onkin palauttaa Makeko Suomen kartalle virtuaalisena palveluna.

Palvelun toteutus alkoi vaatimusten määrittelyllä. Yrityksen toimituspäällikkö Raija Komppa-Rannaste kävi kanssani keskustelun siitä, mitä kaikkea palvelulla pitäisi voida tehdä. Lopullisella tuotteella pitää minimissään voida tehdä kokonaisia kokeita eli lähtötasotestejä sekä julkaista niitä. Oppilaiden pitäisi myös pystyä suorittamaan niitä asian mukaisissa tiloissa. Ohjelmiston tulee lisäksi tallentaa nämä suoritukset, jotta niitä voitaisiin tutkia kokonaisuutena tilastoista ja mahdollisista diagrammeista opettajan haluamillaan suodattamistavoilla.

Palvelun tärkein käyttötarkoitus on nimensä mukaisesti oppilaiden osaamisen kartoitus. Tällä tarkoitetaan sitä, että opettajan pitäisi voida palvelun avulla helposti tutkia ja analysoida haluamiaan yksityiskohtia oppilaiden tuloksista. Opettaja voi esimerkiksi haluta selvittää, miten hyvin hänen luokkansa kukin oppilas pärjäsi prosenttilaskuissa suhteessa muiden luokkien sekä koulujen opiskelijoihin.

Lopullisen työn toteuttamisen ohella tulen kehittämään luomallani ohjelmistolla esimerkkimateriaalia sekä mahdollisesti järjestämään testin/testejä. Luomissani esimerkeissä tulen painottumaan nimenomaan 3. luokan tehtäviin toimeksiantajan pyynnöstä.

3 PALVELUN OHJELMOINTI

Varsinaisen konkreettisen toteuttamisen aloitin asentamalla tarvittavat ohjelmistot. Näistä ensimmäinen oli Node.js.

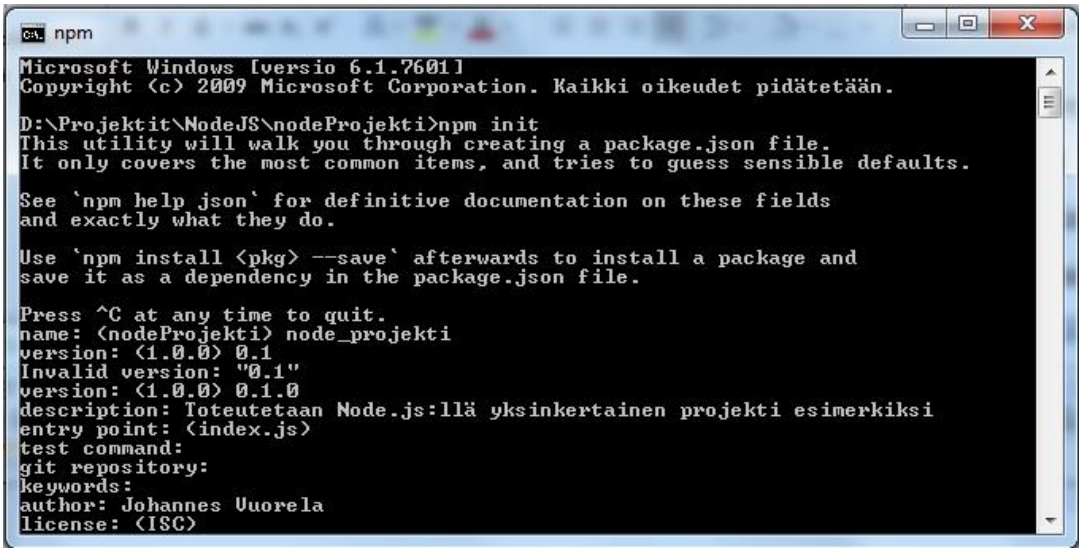
3.1 Node.js ja npm

Node.js on moderni, Googlen Chrome-selaimen Javascript-kieleen pohjautuvan V8-moottorin päälle rakennettu, palvelinpuolen eli ns. Back-endin ohjelmointiympäristö (Node.js 2016). Henkilö nimeltä Ryan Dahl loi Node.js:n nykyaikaisten verkkosovellusten serveriratkaisujen luomista sekä ylläpitoa varten. Syntymästään asti se on kerännyt jatkuvasti kasvavaa suosiota ensinnäkin sen kielivalinnan vuoksi. Javascript on aikanaan luotu antamaan internet-sivuille interaktiivisuutta. Ensimmäistä kertaa kyseistä kieltä voi soveltaa myös verkkosivujen palvelinpuolella, ja sillä itsessään on monta tarkoitusta. Se helpottaa ns. front-end-puoleen eli käyttäjäpuoleen erikoistuneita ohjelmoijia kommunikoimaan edellä mainittuun back-end-puoleen erikoistuneiden kanssa. Vaihtoehtoisesti samat henkilöt voivat toteuttaa verkkopalveluun sekä käyttäjä- että palvelinpuolen, jolloin puhutaan ns. Full stack –kehitystyöstä.

Node.js:n charmikkuus perustuu myös siihen, että se on helposti lähestyttävä. Nodella saa nopeasti yksinkertaisen sovelluksen toteutettua siitä hetkestä, kun sen asentaa, mm. siksi, että sen uudemmat versiot tuovat tullessaan sisään rakennetun apuohjelman nimeltä npm. Tämän ohjelman nimi on lyhenne sanoista Node Package Manager (suomeksi Noden pakettimanageri). Npm nimensä mukaisesti auttaa ohjelmoijaa asentamaan verkkosovellukseensa tarvittavat valmispaketit. Nämä paketit kaikessa yksinkertaisuudessaan sisältää valmista Javascript-koodia, jonka joku muu käyttäjä on omassa projektissaan kirjoittanut ja siten päättänyt jakaa esim. Github-nimisen verkkopalvelun kautta koko maailman kaikkien Node-kehittäjien käytettäväksi. Npm tekee koodin kirjoittamisen helposti jaettavaksi sekä päivitettäväksi. (What is npm? 2016)

Npm perustuu komentorivillä syötettäviin komentoihin. Niistä ensimmäinen komento, jonka opetteleva kehittäjä kirjoittaa, on todennäköisimmin npm init. Tällä komennolla voidaan luoda uusi Node.js:ään pohjautuva projekti. Ohjelma pyytää projektin luojalta tässä vaiheessa muutamia tietoja, esim. projektin nimen, version ynnä muuta. Tietojen

syöttämisen jälkeen kohdekansioon ilmestyy json-muotoinen tiedosto package. Tähän tiedostoon ilmestyy kaikki projektiin liittyvä tieto, joihin lukeutuu se, mitä käyttäjä npm init –komennon ohessa syötti.



KUVA 1. Uuden projektin luominen npm:llä

Tiedostosta löytyy myös dependencies-arvo. Jos tätä arvoa ei löydy tiedostosta, se tulee kirjoittaa itse. Tämän arvon alle voidaan luetella edellä mainittujen valmispakettien nimiä versionumeroineen, jolloin syntyy ns. riippuvuuksia. Jos paketista haluaa uusimman version, versionumeron voi korvata sanalla latest. Nämä riippuvuudet ilmoittavat, mitkä paketit npm asentaa, kun komentoriviin syötetään komento npm install. Paketteja voi vaihtoehtoisesti asentaa muokkaamatta tiedostoa komennolla npm install ”paketin nimi.” (esim. npm install express) Komennon perään voidaan vielä lisätä esim. save-attribuutti (--save), joka luo kyseiselle paketille paketille automaattisesti riippuvuuden samalla, kun asentaa sen projektiin, jolloin sitä ei tarvitse kirjoittaa erikseen itse. Huomataan, että pakettia näin asentaessa ei tarvita versionumeroa eikä sanaa latest. Versionumeron voi lisätä paketin nimen jälkeen ennen attribuutteja, jos haluaa muun kuin viimeisimmän version. (Using a ’package.json’ 2016)

```
{
  "name": "node_projekti",
  "version": "0.1.0",
  "description": "Toteutetaan Node.js:llä yksinkertainen projekti esimerkiksi",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Johannes Vuorela",
  "license": "ISC",
  "dependencies": {
    "express": "^4.13.4",
    "jade": "^1.11.0",
    "mysql": "^2.11.1",
    "morgan": "latest"
  }
}
```

KUVA 2. Esimerkki package.json-tiedostosta

Varsinainen ohjelmointityö on Javascript-tiedostojen kirjoittamista ja niiden ajamista npm-ympäristössä. Ensimmäinen tiedosto, joka yleensä kirjoitetaan, on package.json-tiedostossa mainittu ”päätiedosto”, joka mainitaan muuttujassa ”main”. Kuvan 2 esimerkissä tämä tiedosto olisi nimeltään ”index.js”.

Node-ympäristössä kirjoitettu javascript-tiedosto alkaa tyypillisesti sillä, että määritellään kaikki projektiin asennetut paketit, joita tarvitaan kyseisessä tiedostossa. Tyypillisesti kullekin näistä paketeista luodaan oma muuttuja, joka yksinkertaisesti nimetään omalla nimellään, esim. ”var http = require(’http’);”. Lisäksi tämän muuttujan avulla voidaan määritellä erilaisia ns. ”apumuuttujia”, joihin palaan tässä työssä hieman myöhemmin. Luodun tiedoston ainut varsinainen virka on ohjata ylläpidettävän palvelimen logiikkaa, eli millaisiin http-pyyntöihin vastataan ja millä tavalla. Loin esimerkiksi yksinkertaisen Hello World –henkisen ohjelman, joka palauttaa pääsivulle tultaessa yksinkertaisesti h1-elementillä kirjoitetun otsikon ”Heippa maailma!”.

```
var http = require('http');

var palvelin = http.createServer(function(pyynto, vastaus) {
  vastaus.writeHead(200, {"Content-Type": "text/html"});
  vastaus.end("<h1>Heippa maailma!</h1>");
});

palvelin.listen(3000);
console.log("Palvelin on päällä");
```

KUVA 3. Yksinkertainen Node.js palvelimen tiedosto

Kun koodi on kirjoitettu käyttökelpoiseen kuntoon, eli kun mahdollisia syntaksivirheitä ei löydy, palvelin voidaan käynnistää komennolla node ”tiedoston nimi, joka ajetaan”, esimerkkitapauksessa ”node index.js”. (Herron 2013, 31)

Esimerkkipalvelimemme koodi koostuu kolmesta eri osasta. Ensin on määritelty http-nimiselle paketille oma muuttuja, luonnollisesti nimeltään http. Huomaa, että http-paketti on aina Node:ssa mukana, eikä täten tarvitse asentaa manuaalisesti erikseen. Toisekseen luomme palvelimelle ns. ”kuuntelija” (engl. listener), joka vastaanottaa selaimen käyttäjältä erikseen määritellyn pyynnön. Pyynnön saatuaan kuuntelija suorittaa funktion, jonka sisällä voidaan tehdä lukuisia eri asioita. Mitä pääasiassa tahdotaan tehdä, on vastauksen palauttaminen takaisin pyynnön lähettäjälle, joka tässä tapauksessa on se kävijä, joka selaimellaan navigoi meidän palvelimen sivulle. Lähetetty vastaus esimerkissämme on yksinkertainen HTML-kielen h1-elementti ”Heippa maailma!”, jonka aikaisemmin asetin tavoitteeksi. Kehittäjä käyttää ohjelman testaamiseen palvelimen osoitteena ”http://localhost”. Osoitteen perään lisätään kaksoispisteen kera jokin porttinumero, joka yleensä määritellään erikseen koodin perässä, kuten esimerkkimme koodissa on tehty.

Kuvan 3 koodi antaa osviittaa sille, miten pienellä määrällä tekstiä voidaan pystyttää yksinkertainen Node.js-pohjainen moderni palvelin. Tähän työhön sellainen ei kuitenkaan riitä. Tulen tarvitsemaan paljon enemmän komponentteja sekä muita keinoja toteuttamaan kokonainen Full Stack -periaatteella toteutettu palvelu.

3.2 Express.js

Olemassa olevista Noden paketeista ensimmäinen ja tärkein on lähes kaikissa Nodella toteutetuissa verkkopalveluissa mukana. Sen nimi on Express ja se on niinkin tärkeä, että se mainitaan yhtenä neljästä komponentista MEAN-arkkitehtuurin nimessä Node.js:n rinnalla. Kirjaimet M ja A tulevat sanoista MongoDB ja AngularJS (MEAN.IO, 2016). Express helpottaa oman verkkosovelluspalvelimen ylläpitoa huomattavasti esim. helpottamalla syntaksia ja on siksi suosittu. Syntaksin helpottamisella tarkoitan tässä sitä, että pyyntöjen vastaanottamiseen luodut erilaiset kuuntelijat on selvästi helpompaa toteuttaa Expressin kanssa. Expressillä luodulle, projektin alussa määritellylle muuttujalle, joka olkoon tässä esimerkissä app, voidaan määritellä funktioita, ikään kuin jQueryssä, jotka ovat muotoa: ”app.get(’/’, function(parametrit) { tee jotain

});" Oheisen funktion sisään voidaan kirjoittaa koodia, jossa mm. palautetaan vastaanotetun pyynnön mukaisesti määrittyvä vastaus loppukäyttäjälle. Pyyntöjä, joita nämä kuuntelijat vastaanottavat, voidaan karsia ja parsia jatkamalla kyseisen kuuntelijan ensimmäistä parametria. Jos esimerkkinä halutaan lähettää erilainen viesti riippuen siitä, tullaanko pääsivulle vaiko "moikka"-nimiselle alasivulle, voidaan näille luoda omat kuuntelijat, joista ensimmäinen noudattaa edellä mainittua kaavaa, ja toinen puolestaan muotoa: "app.get('/moikka', function(parametrit) { tee jotain });". Tällä tavoin voimme manipuloida monipuolisemmin sitä, miten palvelin käsittelee osoiteriviä edukseen. Näillä funktioilla voidaan myös esim. vastaanottaa parametreja suoraan osoiterivistä. Lisäämällä kaksoispiste jonkin osoitehaaran eteen voidaan määritellä kyseessä olevan muuttuja, joka voidaan tallettaa suoraan kuuntelijan käynnistämän funktion sisälle. Esimerkiksi tapauksessa "app.get('/moikka/:nimi')" huomataan, että nimi on määritelty muuttujaksi. Jos käyttäjä menee selaimellaan osoitteeseen "http://osoite:portti/moikka/Jussi", voimme varastoida nimen Jussi oheiseen nimi-arvoon ja tehdä sillä jotakin, esim. tulostaa sen osana sivulla ilmenevää otsikkoa (Express routing 2016).

```
1  var express = require('express');
2  var router = express.Router();
3
4  /* GET home page. */
5  router.get('/', function(req, res, next) {
6    res.send('<h1>Heippa, maailma!</h1>');
7  });
8
9  router.get('/moikka', function(req, res, next) {
10   res.send('<h1>Moikka, maailma!</h1>');
11 });
12
13 router.get('/moikka/:nimi', function(req, res, next) {
14   res.send('<h1>Moikka, ' + req.params.nimi + '!</h1>');
15 });
16
17 module.exports = router;
18
```

KUVA 4. Esimerkki Express-ympäristön tiedostosta

Expressin toinen hyvä puoli on se, että siihen on sisäisesti rakennettu valmis työkalu "Express generator", jolla kokonaisia projekteja voi luoda selvästi helpommin ja nopeammin kuin pelkällä Node.js:llä. Express generatorin käyttö on samanlaista kuin esim. Phonegap:issä. Nykyään Cordovana tunnettu Phonegap mahdollistaa verkkosivujen kääntämisen ns. hybridisovelluksiksi älypuhelimille komentorivillä samaan tapaan kuin

Express generatorilla. Tyypillisen komennon ”npm init” sijaan komennolla express "projektin nimi" (esim. tässä tapauksessa express makeko) voidaan luoda valmis monikansioinen rakenne, joka sisältää kaikki olennaiset tiedostot valmiina helposti ymmärrettävässä kansiorakenteessa sekä sen lisäksi asentaa automaattisesti monia yleiskäytännöllisiä paketteja. Tässä asennusprosessissa ei siis tarvitse syöttää itse tietoja projektista samalla tavalla kuin ”npm init” -komennon seurauksena. (Express application generator 2016)

3.3 Jade ja Bootstrap

Express generatorin automaattisesti asennettujen pakettien joukossa on Jade-niminen paketti, joka nyttemmin tunnetaan tekijänoikeussyistä nimellä Pug. Se on yksi lukuisista ns. HTML:n puskurointimoottoreista (engl. rendering engine), joiden tehtävä on tehdä HTML-koodin kirjoittaminen nopeammaksi ja kivuttomammaksi, etenkin, jos käytetään erikseen saatavia tyylipaketteja, kuten Bootstrapiä.

Bootstrap on erillinen skriptikirjasto, jonka saa verkosta ilmaiseksi, ja joka sujuvoittaa huomattavassa määrin oman verkkosivuston tai palvelun visuaalista ilmettä, kuin myös responsiivisuutta. (Bootstrap 2016) Nykypäivänä sivuston tulee ottaa huomioon pöytätietokonenäyttöjen lisäksi myös pienempien älylaitteiden, kuten puhelinten, näyttökoot. Bootstrap on omiaan ottamaan nämä huomioon skaalautuvuudellaan. Sen käyttö omalla sivustolla on pitkälti ylimääräisten Div-elementtien sijoittelua sivustolle varsinaisen sisällön ympärille, sekä lukemattomien CSS-kielellä kirjoitettujen erilaisten luokkien käyttöä. Huonona puolena tässä on se, että Bootstrap tekee helposti HTML-koodista runsasta ja paikoin jopa vaikealukuista. Tätä tasapainoittamaan tulen käyttämään Jaden/Pugin kaltaista moottoria.

```
doctype html
html
  head
    title Makeko Koe-editori
    link(rel='stylesheet', href='https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css')
    link(rel='stylesheet', href='//maxcdn.bootstrapcdn.com/font-awesome/4.5.0/css/font-awesome.min.css')
    script(src='https://code.jquery.com/jquery-1.12.3.min.js')
    script(src='https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/js/bootstrap.min.js')
    script(src='/javascripts/koeeditor.js')
    link(rel='stylesheet', href='/stylesheets/koeeditor.css')
    meta(name='viewport', content='width=device-width, initial-scale=1')
    meta(charset='utf-8')
  body
    .container-fluid
      h1 Makeko Kokeet - #{ muokattavaKoe }
        span#kokeenId (#{ muokattavaId })
      a.btn.btn-primary.btn-lg(href='http://localhost:3000') Takaisin
      a.btn.btn-warning.btn-lg(type='button', data-toggle='modal', data-target='#releaseDialog',
      | style='float: right;') Julkaise koe
      .row
        .col-md-4
          h2(align='center') Tehtävät
          ul#tehtavat.nav.nav-pills.nav-stacked
        .col-md-4
          h2(align='center') Esikatselu
          #esikatselu.tab-content
        .col-md-4
          h2(align='center') Kokeen sisältö
          ul#kokeenRakenne.nav.nav-pills.nav-stacked

    #releaseDialog.modal.fade(role='dialog')
      .modal-dialog.modal-sm
        .modal-content
          .modal-header
            button.close(type='button', data-dismiss='modal') &times;
            h4.modal-title Julkaise koe
          .modal-body
            select
              option Luokka 1
              option Luokka 2
          .modal-footer
            button.btn.btn-warning(type='button', data-dismiss='modal') Julkaise Koe
            button.btn.btn-danger(type='button', data-dismiss='modal') Peruuta
```

KUVA 5. Esimerkki Jade-tiedostosta

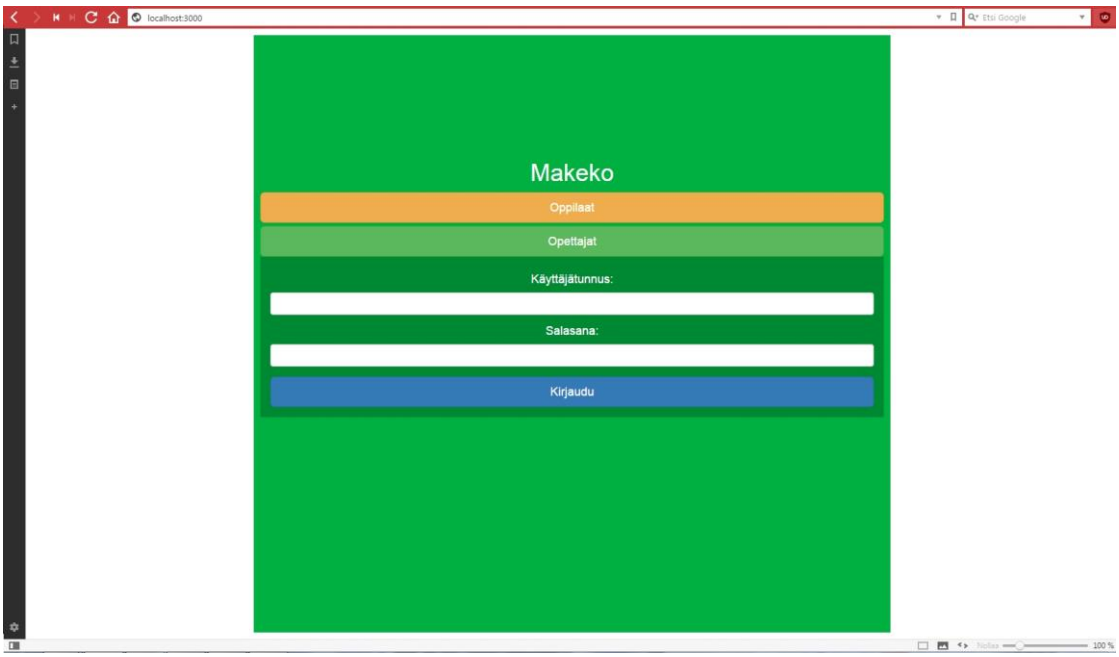
Jaden käyttö perustuu pääasiassa siihen, että kyseiseen työkaluun on liitetty oma skriptikieli, joka muistuttaa erehdyttävästi luonnollista HTML-koodia. Suurimmat erot ovat siinä, että elementtejä ei sisällytetä tagien sisälle, vaan ne kirjoitetaan sellaisenaan omalle rivilleen, jonka jälkeen elementin sisältö voidaan kirjoittaa välilyönnillä erotettuna itse elementin perään. Ainut elementti, jota ei tarvitse kirjoittaa, on Div-elementti, ja tämä on se, missä Bootstrapin liittäminen projektiin mukaan tulee sujuvammaksi. Elementille voidaan antaa attribuutteja, kuten HTML-kielessä, kun ne kirjoitetaan sulkujen sisään pilkulla eroteltuna. Kahteen attribuuteista pätee poikkeus. Samaan tapaan kuin CSS:ssä, elementille voidaan antaa id, kun se liitetään elementin perään risuaidan kanssa. Sama pätee luokkiin, ne tosin liitetään perään pisteillä eroteltuna risuaidan sijasta. (Learn Jade 2016)

Jade-tiedosto saadaan muutettua HTML-koodiksi ja sitten lähetettyä käyttäjälle Expressin avulla, kun tavanomaisen kuuntelijan sisään syötetään metodi ”res.render(’jade-tiedoston nimi ilman päätettä’);”. Kyseinen metodi hyväksyy sekä yhden että kaksi attribuuttia. Pelkkä tiedosto voidaan ajaa ja lähettää itsessään, tai siihen voidaan myös lisätä erilaisia muuttujia mukaan. Muuttujat lisätään render-funktion toisessa attribuutissa JSON-olion sisään yksittäisinä arvoina. (Using template engines with Express 2016)

3.4 Sisäänkirjautumisen toteuttaminen

Projekti lähti liikkeelle ensimmäisestä sivun näkymästä, joka oli luonnollisesti sisäänkirjautuminen. Tähän tarvitsin keskitetyn, responsiivisen ruudun, jossa on kaksi erillistä kirjautumislomaketta; toinen oppilaille, toinen opettajille. Näistä kahdesta lomakkeesta todennäköisesti vain toinen on se, mitä kukin käyttäjä tulee aina tarvitsemaan, joten lähdin siitä liikkeelle, että nämä lomakkeet voisi liittää Bootstrapin ns. haitariratkaisuun eli Accordioniin. Lisäksi viimeistään tässä vaiheessa valita meidän tietokantaratkaisu monista olemassa olevista. Yksi modernimpi vaihtoehto olisi asentaa MongoDB. Kokemuksesta voin kuitenkin suositella MySQL:n käyttöä vielä tänäkin päivänä mm. sen helppokäyttöisyyden takia, etenkin Windows-ympäristössä. Molemmille näille sekä pitkälti jokaiselle muulle olemassa olevalle tietokantakielelle on oma helposti lähestyttävä pakettinsa Node:n ympäristössä valmiina asennettavaksi, joten loppujen lopuksi tällä valinnalla ei ole niin suurta merkitystä, kuin voisi alkujaan ymmärtää. MySQL yhdistetään Nodeen paketilla nimeltä mysql, ja se asennetaan luonnollisesti komennolla npm install mysql (GitHub – mysqljs/mysql 2016). Paketin asennuttua voimme ottaa Mysql-paketin käyttöön palvelimen lähdekoodissa.

Näkymän olisi voinut toteuttaa perinteisesti yksittäisellä HTML-näkymällä, mutta asentamamme Jade-paketti mahdollistaa paljon sujuvamman sivujen kokoamisen, kuten edellä mainittu. Tein juuri näin, eli vastaavanlaisessa funktiossa määrään ohjelman lähettämään ensimmäisen sivun käyttäjälle, joka ei tässä tapauksessa ole ”index.html”, vaan ”layout.jade”-tiedosto. Tässä tiedostossa on Jade-kielellä kirjoitettu etusivu, joka näyttää paljon kompaktimmalta ja yksinkertaisemmalta, kuin vastaava HTML-sivu. Tämä sivu voidaan kääntää HTML-kieleksi ja sitten lähettää käyttäjälle komennolla ”res.render(’layout’)”. Huomaa, että tässä ei tarvita tiedoston päätettä eikä minkäänlaista kansiopolkua, pelkkä tiedoston nimi riittää.



KUVA 6. Makeko-palvelun sisäänkirjautumisnäkymä

Sisään kirjautumisen tapahtumassa on useita vaiheita. Ensimmäisenä käyttäjä yrittää kirjautua sisään eli syöttää tilinsä tiedot asian omaisiin tekstikenttiin. Seuraavassa vaiheessa käyttäjä lähettää lomakkeen tiedot palvelimelle erillistä nappia painamalla, joka tässä tapauksessa lähettää ”get”-tyyppisen Ajax-komennon. Tämä komento määrittäisi sen, mitä palvelin palauttaa vastaukseksi. Komento sisältää kirjautujan syöttämän tunnuksen ja salasanan. Olisi tosin epäammattimaista olla käyttämättä minkään sortin kryptaus- tai salausmenetelmää, kun käsitellään arkaluontoista tietoa. Siksi toinen paketti, joka asennetaan tässä vaiheessa mysql:n lisäksi on jotakin näistä menetelmistä käyttävä lisuke. Valitsin niistä sen, jota opin nopeitten käyttämään, eli crypton. Crypton avulla voin laatia vähän tilaa vievän funktion, joka vastaanottaa merkkijonon ja palauttaa sen tietyllä logiikalla salattuna.

```
205
206 function salaa(merkkijono) {
207     var cipher = crypto.createCipher("aes-256-ctr", "Saapasjalkakissa");
208     return cipher.update(merkkijono, "utf8", "hex") + cipher.final("hex");
209 }
```

KUVA 7. Esimerkki funktiosta, joka salaa tietoa

Salauksen luoma sillisalaatti on se merkkijono, jota ohjelma tulee vertaamaan tietokannasta löytyviin, samalla algoritmilla salattuihin tietoihin aina sisään kirjautuessa. Minkäänlaista purkausmetodia ei tässä tapauksessa tarvita.

Lähetetty lomake salattuine tietoineen käsitellään erillisessä palvelimen ”get”-metodissa, joka on tässä tapauksessa router.get(’/kirjaudu/:tunnus/:salasana’, …), josta näemme suoraan, mihin osoitteeseen ajax-komento tulee lähettää. Aikaisemman esimerkin mukaisesti, jos makeko.sanomapro.fi olisi palvelun osoite, kyseinen tapahtuman osoite olisi makeko.sanomapro.fi/kirjaudu/:tunnus/:salasana. Tässä huomataan, että kyseinen kuuntelija vastaanottaa kaksi parametria, tunnuksen ja salasanan. ”get”-komennon sisällä voidaan purkaa saadusta osoitteesta kyseen omaiset parametrit ja tallentaa erillisiin muuttujiin. Esim. jos kirjautuisin tunnuksella ”erkki.erilainen@eposti.fi” ja salasanalla ”ronttionkissa”, ajax-komennon osoite olisi: ”makeko.sanomapro.fi/kirjaudu/erkki.erilainen@eposti.fi/ronttionkissa”, jolloin koko ajax-komento olisi jQueryllä kirjoitettuna ”$.get(”makeko.sanomapro.fi/kirjaudu/erkki.erilainen@eposti.fi/ronttionkissa”, function(data) {});” (jQuery API documentation 2016) Edellyttäen, että saadut tiedot täsmäävät niitä, jotka löytyvät tietokannasta, voimme palauttaa selaimelle renderöidyn sivun opettajanäkymästä, joka sitten korvaisi nykyisen näkymän.

Opettajanäkymä koostuu kolmesta osasta: yläpalkista, koeosuudesta sekä tilastonäkymästä. Yläpalkkiin sijoitetaan mm. projektin logo, tervetuliaistoivotus kirjautuneelle sekä uloskirjautumisnappi. Koenäkymään haetaan tietokannasta kaikki kirjautuneen opettajan tekemät kokeet ja niistä laaditaan lista. Jokainen koe, joka löytyy, muutetaan linkiksi, joka vie koe-editorinäkymään kyseisen kokeen kohdalle. Editorinäkymä on erillinen työkalu, jolla kokeet varsinaisesti tehdään, eli täytetään tehtävistä.

Opettajansivulta löytyy toistaiseksi kaksi nappulaa, toinen uloskirjautumiselle, toinen uuden kokeen luomista varten. Molempien olisi tarkoitus avata oma modaalinsa eli popup-tyyppinen dialogi-ikkuna, jossa kysytään tarvittavia tietoja ennen varsinaista toimenpidettä. Uloskirjautuminen lähinnä vain vahvistetaan, jotta mahdolliset virheklikkaukset voitaisiin välttää, ja kokeenluomisdialogi kysyy, minkä vuosiluokan kokeesta on kyse ja mikä sen nimeksi annetaan.

KUVA 8. Opettajanäkymän ensimmäinen versio

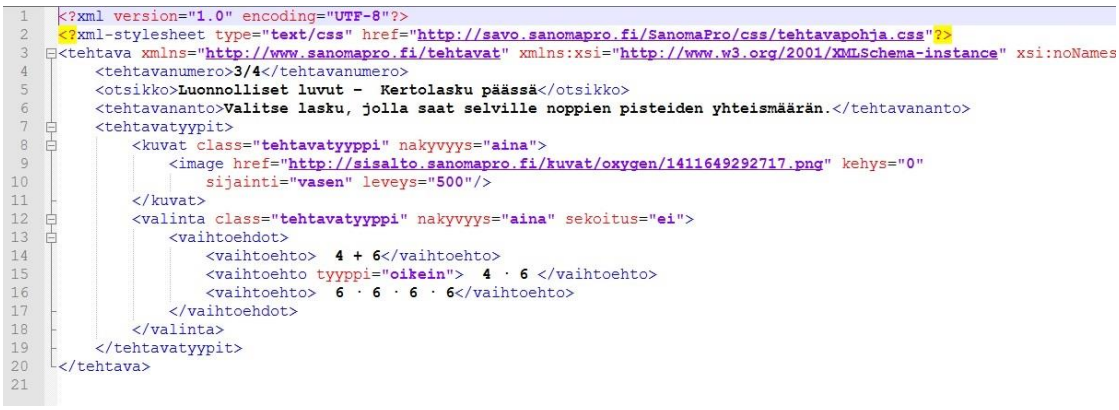
Kun ensimmäinen versio opettajan näkymästä oli luotu, lähdin selvittämään helpointa tapaa lisätä sivustolle sessionhallinta. Tämän voi tehdä joko selainpuolella tai palvelinpuolella. Valitsin näistä selainpuolen lähestymistavan, jolla saadaan tehtyä selaimelle yksinkertainen muistinhallinta, jonka avulla käyttäjä voi sivun päivittyessä pysyä sivulla, kunhan ei sulje välissä selainta. Tässä ratkaisussa on toisaalta vakava tietoturva-aukko.

Aikaiseksi saatu sessionhallinta tulisi liittää etusivulle siten, että ohjelma tarkistaisi aina sille tullessa tai selaimen päivittyessä, onko käyttäjä kirjautunut sivustolle kyseisen istunnon aikana. Jos näin on, lataa palvelin suoraan opettajan näkymän. Muussa tapauksessa selainnäkymään latautuu sisäänkirjautumisruutu.

Seuraavaksi lähdin kehittämään koe-editorin näkymää.

3.5 Koe-editorin toteutus

Seuraava vaihe työssäni oli toteuttaa käyttöliittymä, jossa opettaja voisi lisätä, katsella, poistaa sekä muokata omia kokeitaan, eli käytännössä kokonainen CRUD niille kokeille/testeille, joita oppilaat tekisivät. Tulin siihen johtopäätökseen, että tämä editori tulisi vaatimaan oman osoitteensa, johon pääsee käsiksi opettajanäkymän etusivulta linkin välityksellä. Tämä edellyttäisi sitä, että sivua varten tehtäisiin kokonaan oma jade-tiedosto, joka sisältäisi sekä head- että body-elementin sisällöt.



```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css" href="http://savo.sanomapro.fi/SanomaPro/css/tehtavapohja.css"?>
<tehtava xmlns="http://www.sanomapro.fi/tehtavat" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNames
    <tehtavanumero>3/4</tehtavanumero>
    <otsikko>Luonnolliset luvut -  Kertolasku päässä</otsikko>
    <tehtavananto>Valitse lasku, jolla saat selville noppien pisteiden yhteismäärän.</tehtavananto>
    <tehtavatyypit>
        <kuvat class="tehtavatyyppi" nakyvyys="aina">
            <image href="http://sisalto.sanomapro.fi/kuvat/oxygen/1411649292717.png" kehys="0"
                sijainti="vasen" leveys="500"/>
        </kuvat>
        <valinta class="tehtavatyyppi" nakyvyys="aina" sekoitus="ei">
            <vaihtoehdot>
                <vaihtoehto>  4 + 6</vaihtoehto>
                <vaihtoehto tyyppi="oikein">  4 · 6 </vaihtoehto>
                <vaihtoehto>  6 · 6 · 6 · 6</vaihtoehto>
            </vaihtoehdot>
        </valinta>
    </tehtavatyypit>
</tehtava>
```

KUVA 9. Esimerkki Xml-muotoisesta Makeko-tehtävästä

Editorin käyttöliittymään haluttaisiin näkyviin sekä lista kokeeseen lisättävistä tehtävistä sekä toinen lista jo lisätyistä tehtävistä. Tehtäviä olisi lisäksi hyvä voida esikatsella ennen kuin niitä lisää kokeeseen. Näiden vaatimusten pohjalle perustin kolmijakoisen käyttöliittymän, jossa opettaja valitsee vasemmalta palstalta tehtävän, joka ilmestyy keskimmäiseen palstaan esikatseltavaksi. Halutessaan opettaja voi lisätä ko. tehtävän kokeeseen napista, joka joko lisää tai poistaa tehtävän kokeesta riippuen siitä, onko se jo siellä. Koetehtävien lista päivittyy reaaliajassa oikealle palstalle. Käytännön toteutuksessa tehtävien lista haetaan tietokannasta yhtenä tauluna, jonka jokaista tehtävää kohti laaditaan editorin sivulle oma välilehti. Tähän välilehteen kuuluu kyseisen tehtävän lisäys- ja poistonappi sekä niiden lisäksi itse tehtävän rakenne, jonka olen muuntanut xml-kielisestä sisällöstä html-muotoiseksi div-elementiksi. Tämä oli helpointa toteuttaa Javascriptin tukemilla RegExp-tekniikoilla.

RegExp eli Regular Expression on tekniikka, jolla esim. Javascriptissä voidaan eritellä pitkästäkin tekstistä loogisten lausekkeiden perusteella kaikki toisiaan muistuttavat osat. Käytännön esimerkkinä voimme ottaa jonkin Makeko-tehtävän sisällön tarkasteluun ja poimia sieltä kaikki samankaltaiset tagit. Esimerkkinä voimme etsiä rakenteesta kaikki <lihavointi> -tagit lausekkeella /<lihavointi>/gm. Haluamme muuntaa kaikki vastaavat tagit HTML-kieltä lukevien selainten ymmärtämään muotoon, joka tässä tapauksessa tarkoittaisi niiden muuntamista -tageiksi. Tämä tehdään Javascript-kielellä komennolla ”lauseke.replace(vanha teksti, uusi teksti);”. Tässä käytännön esimerkissä kirjoittaisimme siis ”tehtava.replace(/<lihavointi>/gm, ””);”. Tässä on kuitenkin vielä yksi mutta: Kyseinen komento korvaisi kaikki <lihavointi>-tagit, eli toisin sanoen ainoastaan lihavoidun tekstin aloittavat tagit. Meidän tulisi korvata samalla tavalla kaikki sulkevat tagit, esim. lausekkeella ”replace(/<\/lihavointi>/gm,

””);”, mutta syventämällä vähän Regular Expressionin teoriaa voimme kirjoittaa em. kaksi lauseketta paljon tiiviimpään muotoon. (Regular Expression tutorial 2016)

Edellä mainitut lausekkeet poikkeavat toisistaan ainoastaan yhdellä merkillä, joka joko lisätään väliin tai ei lisätä. RegExp-kielellä voi kirjoittaa lauseita, jotka ottavat huomioon ko. tapaukset. Lausekkeeseen voidaan spesifioida erillinen ”vapaaehtoinen” osa liittämällä sen perään kysymysmerkki. Tässä esimerkissä voisimme esim. poimia sekä kaikki <lihavointi>-kohdat, että kaikki </lihavointi>-kohdat, kirjoittamalla lauseke muotoon ”/<\/?lihavointi>/gm”. Oheisessa lausekkeessa huomaamme, että kauttaviiva ei voi sellaisenaan sisällyttää lausekkeeseen, koska se on erikoismerkki, vaan tarvitsemme erikoismerkeille oman erikoismerkin, joka on sen edessä nähtävä toisinpäin kirjoitettu kenoviiva. (Regular Expression tutorial 2016)

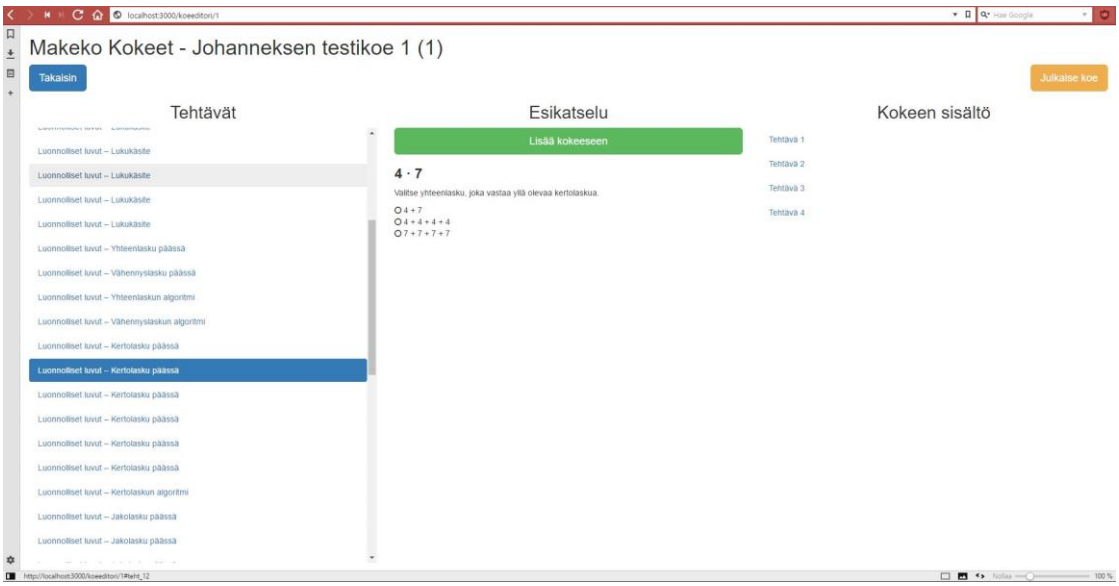
Toinen RegExp:in ominaisuus, jota voimme soveltaa tässä esimerkissä, on muuttuja, joka toteutetaan kirjoittamalla se osa, joka halutaan tallentaa muistiin, sulkujen sisään. Kyseinen osa tallentuu numerojärjestyksessä ensimmäiseen muuttujaan, jota voidaan kutsua korvaavassa lauseessa dollarimerkillä sekä muuttujan omalla numerolla, tässä tapauksessa ”$1”. Kaikki tähänastinen teoria yhdistettynä toisiinsa voimme siis kirjoittaa lausekkeen ”/<(\/?)lihavointi>/gm”. Funktio, jolla lopullinen korvaamisprosessi tehdään, on ”string.replace(/<(\/?)lihavointi>/gm, ”<$1strong>”);”. (Regular Expression tutorial 2016)

```
function xmlToHtml($xml) {
  var $html = $("<div />").addClass("esikatseltava");

  $html.append($("<h3 />").html($xml.find("tehtavananto").html()
    .replace(/ xmlns=\".*?\"/gm, "")
    .replace(/<(\/?)lihavointi>/gm, "<$1strong>")));
  $xml.find(".tehtavatyyppi").each(function() {
    switch(this.tagName) {
      case "aukkotaydennys":
        var rivinKorkeus = $(this).find("syottoruutu").length > 0 ? "300%" : "auto";
        var $form = $("<form />").addClass("form-inline").attr("role", "form")
          .append($("<p />").css("line-height", rivinKorkeus)
          .append($(this).find("taydennysteksti").html()
            .replace(/ xmlns=\".*?\"/gm, "")
            .replace(/<(\/?)kursivointi>/gm, "<$1em>")
            .replace(/<(\/?)lihavointi>/gm, "<$1strong>")
            .replace(/<(\/?)ylaindeksi>/gm, "<$1sup>")
            .replace(/<(\/?)alaindeksi>/gm, "<$1sub>")
            .replace(/<(\/?)alleviivaus>/gm, "<$1u>")
            .replace(/<valintaruutu>(.*?)<\/valintaruutu>/gm,
                "<div class='dropdown' style='float:none;display:inline;'><button class='btn "
              + "btn-default dropdown-toggle' type='button' data-toggle='dropdown' "
              + "aria-haspopup='true' aria-expanded='true'>Valitse "
              + "<span class='caret'></span></button><ul class='dropdown-menu'>$1</ul></div>")
            .replace(/<murtoluku>(.*?)\/(.*?)<\/murtoluku>/gi,
                "<div class='fraction'><span class='fup'>"
              + "$1</span><span class='bar'>/</span>"
              + "<span class='fdn'>$2</span></div>")
            .replace(/=\s<syottoruutu.*?>.*?<\/syottoruutu>/gm,
                "<div class='form-group'><input type='text'"
              + "class='form-control input-sm'></div><br />")
            .replace(/<syottoruutu.*?>.*?<\/syottoruutu>/gm,
                "<div class='form-group'><input type='text'"
              + "class='form-control input-sm'></div>")));
        $html.append($("<div />").append($form));
        break;
      default:
        $html.append($("<div />").append($(this).html().replace(/<kuvat.*?>/gm, "<div>")
            .replace(/<valinta.*?>/gm, "<div>")
            .replace(/<otsikko>(.*?)<\/otsikko>/gm, "<h3>$1</h3>")
```

KUVA 10. Ote funktiosta, jolla xml muutetaan html-muotoon

Muita editoriin haluttavia ominaisuuksia oli luonnollisesti paluulinkki takaisin opettajanäkymän etusivulle sekä linkki, josta päästään syöttämään tarvittavat tiedot kokeen julkaisua varten. Editorista pitäisi saada myös luonnollisesti responsiivinen, eli edellä mainittu ”kolmipalstajako” voi koitua mobiilinäytöillä ongelmaksi. Todennäköisesti editorille tulisi luoda pienemmille resoluutioille kokonaan oma käyttöliittymä (kuten mahdollista myös muissa sivunäkymissä), mutta tähän minä palaan lähempänä työn valmistumisajankohtaa.



KUVA 11. Koe-editori

Olen tehnyt juuri sen, mitä suunnittelin edellä. Huomioitava tosiseikka on tässä vaiheessa vain se, että tähän näkymään ei pääse vielä mitään kautta. Ensin pitää voida luoda koe, sekä lisätä ominaisuus järjestelmään, joka lisää automaattisesti linkin tähän näkymään ja vieläpä siten, että siinä voi muokata vain yhtä, ennalta määrättyä koetta, jonka sisään kirjautunut opettaja on nimenomaan itse luonut, ja johon täten vain hänellä olisi oikeudet.

Koelistan päivittäminen edellyttää uusien ajax-komentojen lisäämistä opettajan etusivunäkymään. Aloitin luomalla oman nappulan kokeen luomista varten. Kyseinen nappi avaa erillisen modal-ikkunan, jossa syötetään vähimmät tarvittavat tiedot: kokeen nimi sekä vuosiluokka, jolle koe on tarkoitettu. Perään tulee lisätä varsinainen tallennuspainike, jota painamalla koe lisätään kantaan ja jota voi täten muokata. Ylimääräinen ominaisuus, jonka lisäsin, on se, että tallennuksen ohessa sivusto ohjaa käyttäjän automaattisesti koe-editoritilaan, jossa juuri luotua koetta voi muokata edellä mainituilla ominaisuksilla.

Nyt kun kokeita voi luoda, tulee kokeen luonnin ohessa etusivulla oleva koelista myös päivittää reaaliajassa. Loin tätä varten tavanomaisen listan, jonka jokainen elementti, tässä tapauksessa yksittäinen koe, sisältää sekä kokeen nimen, että painikkeet kokeen muokkausta ja poistamista varten.

KUVA 12. Opettajanäkymän etusivu koelistalla

Tässä vaiheessa aloin päivittää tietokantaa uusilla attribuuteilla sekä sisällöllä, jota tulisin tarvitsemaan lopullisessa esityksessä. Ennen kuin koetta voi julkaista, tarvitaan koe, jonka on luonut joku opettaja, jossa on joitain tehtäviä, ja johon osallistuu yksi tai useampi oppilas ennalta määrättyyn aikaan. Lähdin luomaan erinäisiä fiktiivisiä tietoja, joita tarvittaisiin tietokantaan ennen myöhempää ohjelmistonkehitystä, kuten koulu, luokka sekä luokkaan kuuluvat oppilaat.

3.6 Turvallisemman sessionhallinnan toteutus

Seuraavaksi lähdin laatimaan järjestelmälle uutta, entistä tietoturvallisempaa sessionhallintaa. Uusi versio tulisi tallentamaan välimuistin tarvitseman tiedon palvelinpuolelle suojauksen kera, jotta järjestelmään murtautuminen jonkin opettajan tunnuksilla tulisi mahdollisimman hankalaksi, ellei mahdottomaksi. Mikä tekee tästä ratkaisusta turvallisemman edelliseen nähden, on se, että ulkopuolinen henkilö ei voisi väärentää sessioita selainpuolella.

Toinen huomioitava seikka nykyisessä sessionhallinnassa on se, että kellä tahansa on pääsy mahdollisesti jopa arkaluonteisiin tietoihin osoiterivin kautta. Kuten jo Expressiä esitellessä on tullut esille, palvelinta voi navigoida kirjoittamalla erilaisia osoitteita tietyllä logiikalla. Esim. ”http://localhost:3000/opettaja/1” palauttaa JSON-objektin, joka sisältää tiedot opettajatunnuksesta, joka tällä hetkellä edustaa id:tä 1, joka on tässä tapauksessa oma testitunnukseni. Näihin tietoihin pääsisi käsiksi kuka tahansa, jos erillistä sessiotestiä ei suoriteta. Toisin sanoen palvelusta tulee tehdä sellainen, että kaikki muutkin kuin pelkkä etusivu tarkistavat, löytyykö välimuistista jo sisään kirjautunut henkilö. Henkilökohtaisesti miellyttävin lähestymistapa oli toteuttaa sisäänkirjautumisruutu, jota kysyttäisiin aina, kun tarvittava tieto puuttuu välimuistista, sivulla kuin sivulla.

Uuden sessionhallinnan toteuttamiseen tarvittiin uutta komponenttia, express-session-komponenttia, joka oli valmiiksi mukaan pakattuna vanhemmissa Expressin versioissa, viimeistä kertaa versiossa 3.0, mutta joka käyttämässämme versiossa 4 joudutaan ensimmäistä kertaa asentamaan erikseen, jälleennimetyn komponentin muodossa. (Migrating to Express 4 2016)

3.7 Kokeen julkaisutyökalun toteutus

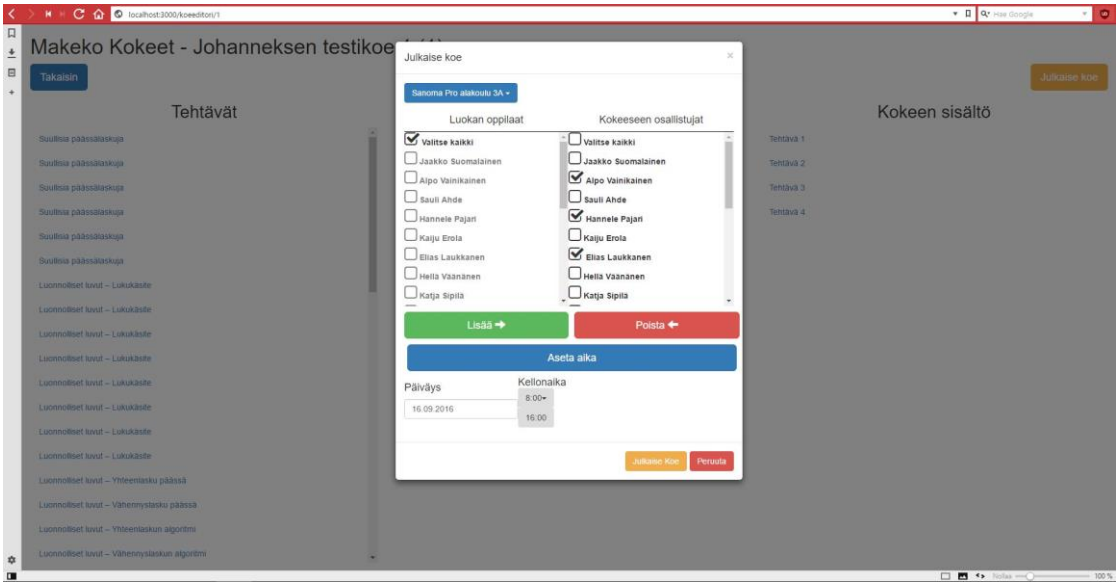
Kun tietokantaan oli lisätty joitain esimerkkitietueita ja sessionhallinta oli tehty turvallisemmaksi, lähdin kehittämään edellä mainittua kokeenjulkaisunäkymää, joka siis olisi koe-editoriin sisältyvä modal-ikkuna. Tässä ikkunassa kokeelle voitaisiin lisätä ja poistaa osallistuvia opiskelijoita samalla periaatteella kuin tehtäviä itse editorissa. Lisäksi kokeelle ominainen tieto on ajankohta, jolloin se pidetään. Näitä kahta tietoa varten oman ikkunan luominen koitui yllättävän haasteelliseksi lähinnä niitä varten tarvittavan tilan määrän vuoksi.

Päätyin lopulta ratkaisuun, jossa kehittäisin modal-ikkunan sisälle taulukon. Keskeltä kahteen palstaan jaettu näkymä oli itselle tuttavallisinta tehdä taulukko-periaatteella, toisin sanoen table-elementillä.

Kuten näemme kuvasta 10, julkaisunäkymän ensimmäiset tiedot ovat otsikko ”Julkaise koe”, sekä alavetolaatikko, johon ajax-komennolla haetaan kaikki ne luokat, joita opettaja sillä hetkellä opettaa, ja jotka edustavat sitä vuosiluokkaa, jolle suunnatusta kokeesta on kyse. Jonkin ryhmän valitseminen tästä valikosta avaa alapuolella olevaan ensimmäiseen laatikkoon vasemmalle palstalle kyseisen luokan oppilaat, joista voidaan valita kaikki ne, joiden halutaan osallistuvan kokeeseen, joko yksitellen poimimalla, tai jos haluaa koko luokan osallistuvan kerralla samaan kohteeseen, voi myös käyttää ensimmäisenä listasta löytyvää ”Valitse kaikki” painiketta. Vihreä lisää-painike lisää valitut oppilaat oikean puoleiselle palstalle, toisin sanoen kokeeseen osallistuvien opiskelijoiden listaan. Samalla periaatteella, tältä listalta voidaan poimia yksittäisiä oppilaita, jotka halutaan poistaa ko. listasta.

Oppilasvalintaruudun perään samaan ikkunaan on lisätty ajansäätönäkymä, jolla asetetaan varsinainen kokeen suoritusaika. Käytettävyyden mukavuuden kannalta päätin piilottaa kyseisen näkymän collapse-paneelin alle. Toisin sanoen, pelkkä ”Aseta aika”-

painike, joka nähdään kuvassa, näkyy ensin pelkästään. Sitä painamalla, yleensä vasta sitten, kun oppilaat on valittu, saadaan varsinainen aikanäkymä näkyviin.



KUVA 13. Koe-editorin julkaisuikkuna

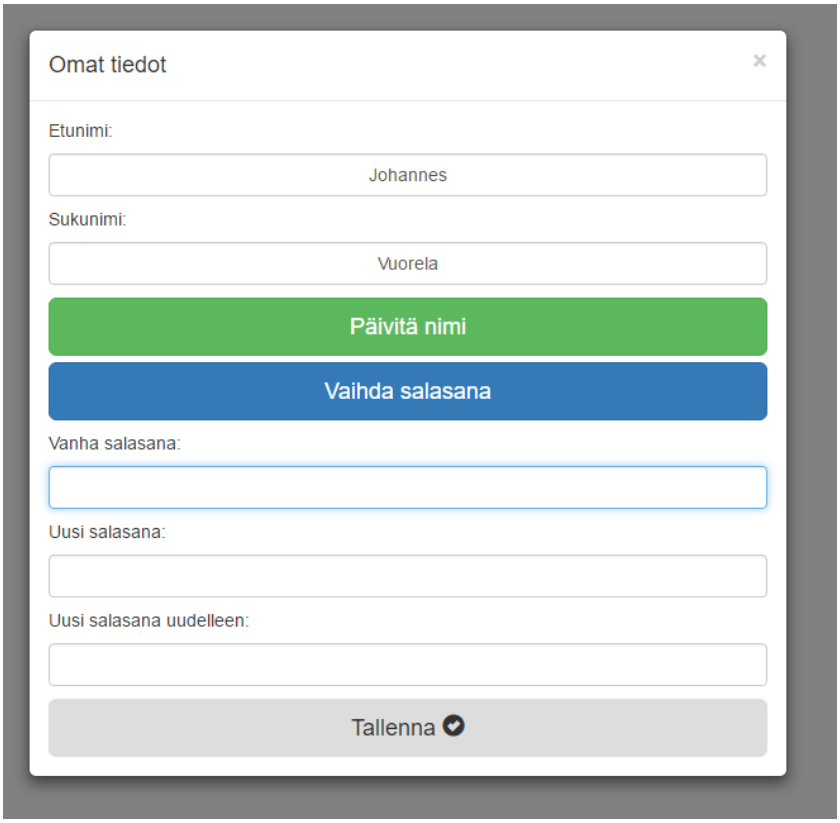
Kun kaikki tiedot on syötetty, voidaan Julkaise koe -painikkeella tallentaa kokeen tiedot tietokantaan. Kullekin oppilaalle, joka löytyy oikean puoleisesta listasta, luodaan oma henkilökohtainen avain, jolla kyseinen oppilas voi kirjautua lyhyen aikaikkunan sisällä koenäkymään, jossa julkaistun kokeen tehtävät ovat samassa järjestyksessä.

3.8 Opettajan tietojen muokkaus

Kehitin tässä välissä uuden vaatimattoman ominaisuuden, jolla käyttäjä (eli opettaja) voisi vaihtaa nimensä ja mahdollisesti muitakin tietoja. (kuten koulun, jossa opettaa) Tietojenmuokkaussivulle pääsee opettajanäkymän etusivulta Omat tiedot-painikkeesta uloskirjautumislinkin vierestä. Samaan navigointipalkkiin on jo etukäteen lisätty Administration-sivun linkki, jolla palvelun ylläpitäjä voi muokata vapaammin mm. tietokantojen sisältöä. (Kuten lisätä uusia kouluja, opettajia jne.)

Tiedonmuokkaussivulle tultaessa palvelin hakee tiedot opettajasta, joka on kirjautunut sisään. Nämä tiedot asetetaan ns. placeholdereina eli tilapäisarvoina input-kenttiin, joihin opettaja itse voi kirjoittaa mahdolliset tiedot, joilla olemassa olevat korvataan. Sivun alaosasta löytyvät Tallenna- ja Takaisin-napit. Jokaista opettajasta kertovaa tietoa var-

ten on olemassa ainakin yksi tilanne, jossa se voi vaihtua, kuten etunimen kohdalla sukupuolen korjaus, sukunimen kohdalla avioliitto jne. Tästä syystä kaikkia tietoja voidaan muokata.



KUVA 14. Tietojen muokkaus-ikkuna

Luotuani erillisen sivunäkymän opettajan tietojen vaihtamiselle, tajusin, miten turha se loppujen lopuksi on. Omia tietoja pitää kyllä voida muokata, mutta jos kyseisiä tietoja on vaivaiset kaksi tai kolme kappaletta, oma sivu niiden mahdolliselle muokkaamiselle on loppupeleissä omasta mielestäni melko ”absurdia”. Tulin siihen tulokseen, että kyseinen ominaisuus tulisi toimimaan paremmin pelkkänä etusivun modal-ikkunana, josta opettaja voisi halutessaan muokata joko etu-/sukunimeään sekä vaihtaa tunnuksensa salasanan.

Kuten näemme kuvasta 11, salasanan vaihtaminen vaatii käyttäjältä kolme tietoa – tunnuksen vanhan salasanan, sekä uuden salasanan kirjattuna kahteen kertaan, samalla tavoin kuin useassa muussa verkkopalvelussa on tapana, mm. niin sanotun ”spämmin” eli järjettömän kirjainten tulvaamisen estämiseksi. Unohdetun salasanan lähetyspyynnön toteuttaminen olisi mahdollista, mutta pohdin sen tärkeyttä vielä uudemman kerran, koska käytännössä unohdetun salasanan voi opettaja pyytää suoraan ylläpitäjältä, jolla

on pääsy näihin tietoihin. Tämä mm. siksi, että Makekon ollessa valmis palvelu olisi suotavaa, jos kutakin koulua kohti olisi oma tunnuksensa, jolla olisi ylläpitäjän käyttöoikeudet.

3.9 Ylläpitäjän työkalun kehittäminen

Seuraavana oli edessä ylläpitäjille oman sivunäkymän kehittäminen. Tällä sivulla ylläpitäjät eli ”administraattorit” voivat päivittää lähes kaikkia palveluun liittyviä tietokantoja, pääasiassa lisäämällä ja muokkaamalla opettajia, oppilaita, kouluja, koetehtäviä ym.

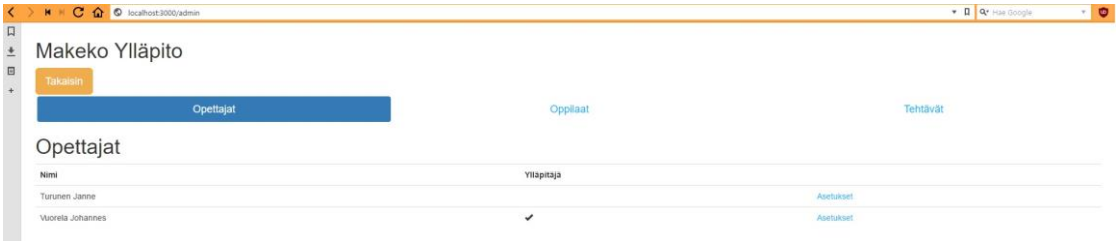
Lähdin suunnittelemaan käyttöliittymää jäsentelemällä ne kaikki tiedot, mitä ylläpitäjän pitää voida palvelulla vaihtaa. Oleellista on se, että näkymässä kaikki eri muokattavat tietueet tulisi selvästi erotella visuaalisesti. Omasta mielestäni loogisin lähestymistapa tälle olisi välilehdet tauluittain.

Yksi oleellinen tietokannan taulu, jonka tietoja pitäisi voida muokata tässä tilassa, on opettajien taulu. Opettajan tunnuksesta tiedetään sähköpostiosoite, jota kyseinen tunnus käyttää sisään kirjautuessaan. Ongelma ylläpitosivua kehittäessä oli se, että sähköpostiosoitteet oli kryptattu, joten niitä oli käytännössä mahdotonta tutkia, saati muokata. Sähköpostiosoitteisiin ei sinällään liity samanlaisia tietoturvahaasteita kuin salasanoihin. Tulin täten siihen tulokseen, että sähköpostiosoitteita ei pitäisi lopputuotteessa syöttää tietokantaan salattuna, jotta niitä voitaisiin tarvittaessa muokata ja tarkastella, jos esimerkiksi joku opettajista sattuisi jostain syystä unohtamaan omansa.

Toinen tieto, joka opettajasta tiedetään ja jota olisi järkevää voida muokata, on hänen nimensä. Tämä tieto kuitenkin on annettu muokattavaksi opettajalle itselleen. Täten opettajalla ei tule koskaan varsinaista syytä pyytää ylläpitäjää muokkaamaan omaa nimeä, mistä johtuen kyseistä ominaisuutta ei tarvita erikseen ylläpitäjän puolella.

Kolmas tieto opettajasta tietokannassa on tieto siitä, onko hänellä ylläpitäjän erityisoikeudet ja täten oikeus päästä ylläpitäjien omaan näkymään. Kyseinen ominaisuus on ilmoitettu lukuna, joka on arvoltaan joko 1 tai 0. Ykkönen merkitsee sitä, että kyseisellä opettajalla on ylläpitäjän oikeudet. Tulin siihen tulokseen, että niiden opettajien, jotka jo ovat ylläpitäjiä, pitäisi voida myöntää oikeuksia muille opettajille.

Lopputuloksena opettajien välilehti tulisi siis tarvitsemaan listan opettajista aakkosjärjestyksessä, josta ylläpitäjä voi poimia yksittäisen opettajan tiedot, vaihtaa sähköpostiosoitetta, jolla joku opettajista kirjautuu, sekä luoda uuden opettajan tarvittavine tietoineen.



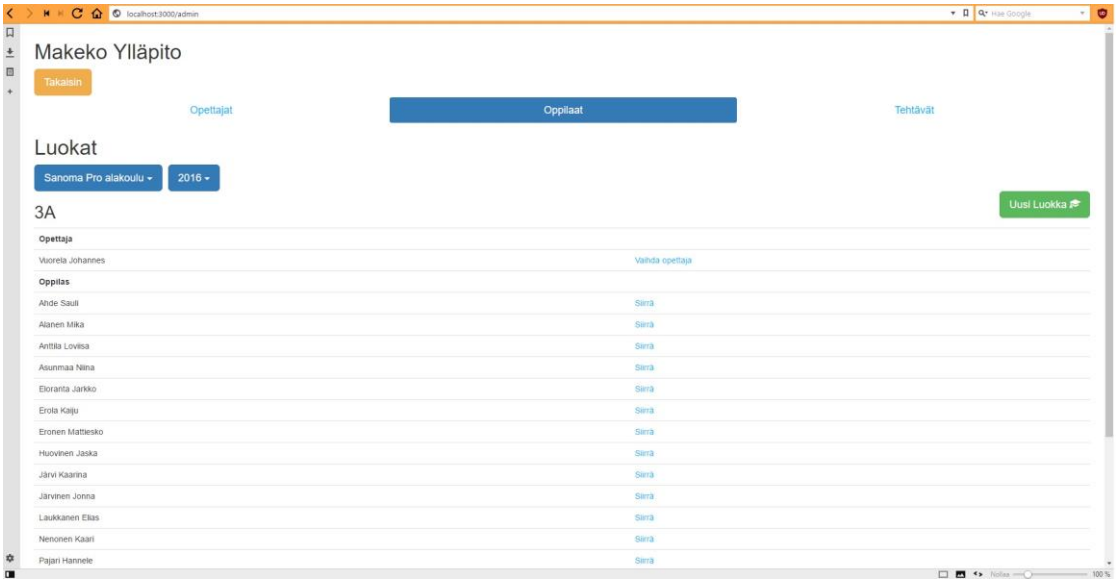
KUVA 15. Ylläpito-työkalun Opettajat-välilehti

Toinen muokattava kokonaisuus, joka ansaitsee ylläpitäjän näkymässä oman välilehtensä, on luokat. Mainitsin sanan ”kokonaisuus”, koska luokkien muokkaamiseen kuuluu useampikin eri tietotaulu, toisin kuin esim. opettajien välilehdessä. Yksittäinen koulu koostuu useista luokista, jotka jakautuvat mm. ikäluokkien mukaan yhdeksään eri vuosiluokkaan. Kukin vuosiluokka koostuu useasta opiskelijasta sekä yhdestä opettajasta. Tästä johtuen välilehden eri ominaisuuksien määritteleminen lähtee liikkeelle erilaisten käyttäjätapausten kautta, toisin kuin opettajien välilehdessä.

Kenties ainoa tapaus, jossa uusia luokkia tullaan luoneeksi – tapahtuu kunkin vuoden elo- ja syyskuun aikana, jolloin uusi lukuvuosi alkaa, ja jolloin uudet oppilaat aloittavat ensimmäisen luokan sekä vanhat oppilaat nousevat vuosiluokassa yhden asteen ylöspäin. Tämän toteuttaminen käytännössä onnistuisi helpoiten lisäämällä kunkin listasta löytyvän oppilaan perään linkin, josta painamalla kyseinen oppilas voidaan siirtää seuraavalle luokalla. Tämä toiminto olisi käytettävissä ainoastaan kunkin vuoden kesäkauden aikana. Lisäksi välilehden yläreunaan lisättäisiin oppilaanluomisnappula, jolla voitaisiin luoda uusi ensimmäisen luokan aloittava opiskelija. Oppilaan pitää voida tarvittaessa myös vaihtaa luokkaa kesken lukuvuoden. Tämä ominaisuus voidaan lisätä kätevästi kunkin listatun oppilaan nimen perään linkkinä.

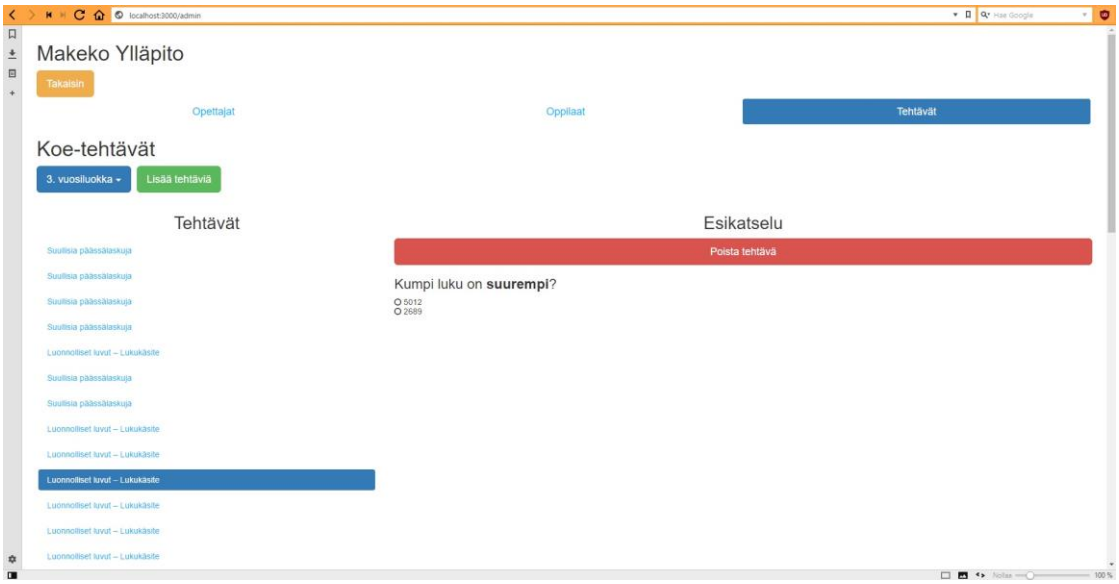
Lopullisessa näkymässä tulee siis olla koulu kerrallaan jokainen luokka omana listanaan, jossa luetellaan kaikki kyseisen luokan oppilaat. Kunkin oppilaan nimen perässä

on linkki, josta ko. oppilas voidaan siirtää rinnakkaiselle tai astetta korkeammalle luokalle.



KUVA 16. Ylläpito-työkalun Oppilaat-välilehti

Koetehtävät tarvitsevat oman välilehtensä, jossa niitä voidaan pääasiassa ainoastaan lisätä tai poistaa. Koetehtävät ovat toimeksiantajan puolelta ennalta määrättyjä xml-tiedostoja, joita yksinkertaisesti pitäisi voida lisätä tietokantaan aina tarvittaessa. Tarvitaan siis toiminto, jossa nappia painamalla voitaisiin tuoda kovalevyltä uusi xml-tiedosto ja lisätä sen sisältämä tieto kantaan muine tärkeine tietoineen, kuten se vuosiluokka, jonka oppilaille ko. tehtävä on suunnattu.



KUVA 17. Ylläpito-työkalun Tehtävät-välilehti

Koetehtäviä olisi hyvä lisäksi esikatsella tässä näkymässä, joka helpoiten toteutuu sillä, että otamme koe-editorista löytyvän esikatseluikkunan ja liitämme sen tänne. Tehtävän kokeeseen lisäämis- ja poistamisominaisuutta ei kuitenkaan tarvita tässä näkymässä.

4 PÄÄTÄNTÖ

Työn lopputuloksena on palvelu, jolla ylläpitäjä voi kirjautua sisään, luoda tietokantoihin tietoja sekä muokata niitä. Ylläpitäjä voi lisätä opettajia, oppilaita sekä koetehtäviä. Palvelulla opettaja voi kirjautua sisään erikseen laaditulla tilillä, joka kantaa hänen nimeään. Opettaja voi vaihtaa tilinsä nimen sekä salasanan pyytämättä lupaa tai oikeuksia ylläpitäjiltä. Lisäksi hän voi luoda ja poistaa kokeita, lisätä niihin tehtäviä, poistaa niistä tehtäviä, sekä julkaista niitä erikseen määritellyn joukon oppilaita tehtäväksi haluamanansa ajankohtana. Tämänhetkinen ohjelmisto sisältää viisi näkymää: sisäänkirjautumisnäkymän kaksi eri versiota, opettajan päänäkymä, ylläpitäjänäkymä sekä koe-editori.

Lopullinen Makeko-palvelu tulee vielä tarvitsemaan ainakin seuraavat ominaisuudet: Oppilaiden mahdollisuus suorittaa kokeita, suoritusten tallentaminen ja tilastointi, sekä suoritusten visuaalinen prosessointi.

Ollakseni rehellinen itselleni voin sanoa olevani hieman pettynyt omaan suoritukseeni. Alkuperäinen tavoite oli toteuttaa koko palvelu kokonaisuudessaan kaikkine alkuperäisine vaatimuksineen sekä pitkälle tyylitellyine ulkoasuineen. Uskoin vakaasti pitkään, että ennalta määritelty seitsemän kuukauden aikaraja riittäisi tähän. Lopulta stressi sekä muut tekijät vaikuttivat negatiivisesti lopullisen tuotoksen laatuun. Kaiken lisäksi työ piti toteuttaa sellaisilla tekniikoilla, joista itselläni ei ollut lainkaan kokemusta, mikä ei helpottanut asiaa. Kolmea kuukautta myöhemmin työ olisi ollut paljon pidemmälle viety sekä kaikin puolin tyylitellympi sekä se olisi ajanut alkuperäisen tarkoituksensa kokonaisuudessaan. Kaikesta huolimatta en voi siltikään kieltää, etteikö lopputulos olisi jollain tapaa kokonainen ja huoliteltu.

Loppujen lopuksi voin todeta, että nämä seitsemän kuukautta tämän työn parissa ovat olleet antoisaa aikaa. Uuden, paljon tätä nykyä peräänkuulutetun ympäristön opetteleminen on ollut henkisesti hyvinkin kehittävä prosessi, puhumattakaan tämän kokoisen

palvelun toteuttamisesta tyhjästä tiukassa aikataulussa. Alkuperäinen tavoite oli kieltämättä ehkä liiankin mahtipontinen sekä kunnianhimoinen, minkä seurauksena joitain oleellisia ominaisuuksia jäi toteuttamatta tässä työssä. Toisaalta osa ominaisuuksista näin jälkikäteen mietittynä vaikuttivat turhilta, kuten mm. responsiivisuus, ja ne olisivat jääneet toteuttamatta joka tapauksessa.

Toimeksiannon toteuttaminen tähän pisteeseen oli sopivassa määrin haastavaa, opettavaista ja hauskaa. Palvelu tullaan kehittämään jatkossa Sanoma Pron päättämällä tavalla, joko minun tai jonkun toisen toimesta.

LÄHTEET

Bootstrap. 2016. Mark Otto. WWW-dokumentti. http://getbootstrap.com. Luettu 9.11.2016

Express application generator. 2016. Strongloop. WWW-dokumentti. https://expressjs.com/en/starter/generator.html. Luettu 9.11.2016

Express routing. 2016. Strongloop. WWW-dokumentti. https://expressjs.com/en/guide/routing.html. Päivitetty 2016. Luettu 8.11.2016

GitHub – mysqljs/mysql. 2016. Diogo Resende. WWW-dokumentti. https://github.com/mysqljs/mysql. Luettu 9.11.2016

Herron, David 2013. Node Web Development Second Edition. Iso-Britannia: Packt Publishing

jQuery API documentation. 2016. The jQuery Foundation. WWW-dokumentti. http://api.jquery.com/jQuery.get/. Luettu 12.11.2016

Learn Jade. 2016. Glenn Yonemitsu. WWW-dokumentti. http://learnjade.com/tour/nesting/. Luettu 9.11.2016

MEAN.IO. 2016. Linnovate. WWW-dokumentti. http://mean.io. Luettu 9.11.2016

Migrating to Express 4. 2016. Strongloop. WWW-dokumentti. https://expressjs.com/en/guide/migrating-4.html. Luettu 9.11.2016

Node.js. 2016. NodeJs Foundation. WWW-dokumentti. https://nodejs.org/en/. Päivitetty 2016. Luettu 8.11.2016

Regular Expression tutorial. 2016. Jan Goyvaerts. WWW-dokumentti. http://www.regular-expressions.info/tutorial.html. Luettu 9.11.2016

Using a 'package.json'. 2016. npm Inc. WWW-dokumentti. https://docs.npmjs.com/getting-started/using-a-package.json. Luettu 8.11.2016

Using template engines with Express. 2016. Strongloop. WWW-dokumentti. https://expressjs.com/en/guide/using-template-engines.html. Luettu 9.11.2016

What is npm? 2016. npm Inc. WWW-dokumentti. https://docs.npmjs.com/getting-started/what-is-npm. Päivitetty 2016. Luettu 7.11.2016