LAMK
Lahden ammattikorkeakoulu
Lahti University of Applied Sciences

HARJOITTELUPORTAALI

Oppilaitoksen työharjoitteluprosessin hallinta hajautetulla arkkitehtuurilla

LAHDEN AMMATTIKORKEAKOULU
Insinööri (AMK)
Tieto- ja viestintätekniikka
Kevät 2019
Olli Rouvinen

Tiivistelmä

Tekijä(t)	Julkaisun laji	Valmistumisaika
Rouvinen, Olli	Opinnäytetyö, AMK	Kevät 2019
	Sivumäärä	
	38	

Työn nimi

Harjoitteluportaali
Oppilaitoksen työharjoitteluprosessin hallinta hajautetulla arkkitehtuurilla

Tutkinto

Tieto- ja viestintätekniikka, ohjelmistotekniikka, insinööri (AMK)

Tiivistelmä

Työn tavoitteena oli kehittää järjestelmä Lahden Ammattikorkeakoulun opiskelijoiden työharjoitteluprosessin hallintaan. Aiemmin käytössä ollut järjestelmä oli korvattava uudella, sillä se alkoi olla vanhentunut eikä enää toiminut uudemmilla selaimilla kunnolla. Työn aikana määriteltiin järjestelmän vaatimukset käyttötapausten perusteella ja suunniteltiin näiden pohjalta toimiva kokonaisuus.

Järjestelmä toteutettiin hajautetun arkkitehtuurin järjestelmänä, jossa järjestelmän datan tarjoileva palvelinsovellus toteuttaa REST-rajapinnan asiakassovelluksen käytettäväksi. Työssä toteutettiin sekä palvelinsovellus että selaimessa toimiva asiakassovellus. Palvelinsovellus rakennettiin Node.js-JavaScript-suoritusympäristön päälle käyttäen Express-sovelluskehystä. Asiakassovellus ohjelmoitiin React-käyttöliittymäkirjastolla. Molempien sovellusten toteutuskielenä oli JavaScript.

Työn toteutuksessa haasteiksi nousivat autentikaation ja erityisesti autorisoinnin toteutus. Palvelinsovelluksen autentikaatio kehitetiin yleisiä käytäntöjä noudattavana JWT-toteutuksena. Autorisointia varten kehitettiin Express-sovelluskehyksen middleware-ketjua hyödyntävä menetelmä API-polkujen suojaamiseen käyttäjän roolin perusteella sekä menetelmä yksittäisen käyttäjän pyyntöjen autorisointiin. Asiakassovellukseen kehitettiin vastaavanlainen menetelmä näkymien suojaukseen käyttäjän roolin perusteella.

Työn tavoitteet saavutettiin: järjestelmällä voidaan hoitaa harjoitteluprosessi alusta loppuun jokaisen käyttäjäroolin kohdalla. Työn lopputuloksena huomattiin, että projektin koodi olisi ollut helpommin testattavaa ja ylläpidettävää, jos palvelinsovellukseen olisi lisätty erillinen DAL-kerros datan haun tehtäviä varten sekä käytetty kielenä TypeScriptiä tai muuta kieltä, jossa on staattinen tyyppijärjestelmä.

Asiasanat

työharjoittelu, JavaScript, REST, autentikaatio, autorisointi, React, Node.js

Abstract

Author(s)	Type of publication	Published
Rouvinen, Olli	Bachelor's thesis	Spring 2019
	Number of pages	
	38	

Title of publication

Portal for practical training
Managing a practical training process with distributed software architecture

Name of Degree

Bachelor of Software Engineering

Abstract

The objective of the thesis was to develop a software system to manage the internship process of students at the Lahti University of Applied Sciences. The completed system was to replace an older system that was in use but was starting to show signs of age. During the project, requirements for the system were defined based on its use cases and a working solution was designed based on these requirements.

The system was designed on a distributed architecture in which a back-end application implements a REST API to be used by a client application. A back-end and a front-end application were developed. The back-end application was built on the Node.js JavaScript runtime utilizing the Express framework to implement the REST API. The front-end client application was programmed in JavaScript as well, using the React UI library.

Authentication and especially authorization presented challenges in the development. A JWT-based authentication scheme following widely used standards was implemented. For authorization, a way of protecting API endpoints based on user roles was developed using the middleware chain of Express. For the front-end application, a similar method for hiding parts of the UI depending on the user's role was developed. In addition, a mechanism in the back-end for authorizing data for individual users was developed. Separation of concerns between authorization and the rest of the request processing was a major goal in the implementation.

The objectives of the project were met: the finished software system can manage the whole internship process from start to end, for each user role. In the end, it was found that the code for the project would have been easier to maintain and test if a language with a static type system such as TypeScript had been used.

Keywords

practical training, JavaScript, REST, authentication, authorization, React, Node.js

SISÄLLYS

1 JOHDANTO

Työn tavoitteena oli toteuttaa Lahden ammattikorkeakoulun tekniikan laitokselle järjestelmä työharjoitteluprosessin hallintaan. Tekniikan laitoksen suuren opiskelijamäärän vuoksi harjoitteluprosessin ja siihen liittyvien liitetiedostojen hallitsemisen avuksi tarvittiin järjestelmä, joka minimoisi käsin tehtävän työn. Aiemmin käytössä oleva järjestelmä oli osoittanut ikääntymisen merkkejä: se näytti antiikkiselta, ja tekninen toimivuus ei enää ollut kunnossa. Ongelmia oli uudempien selaimien kanssa, ja Internet Explorerin kanssa se ei enää toiminut ollenkaan. (Similä 2019.)

Näistä lähtökohdista oli tarve saada uusi ajanmukaisilla tekniikoilla toteutettu järjestelmä, joka vastaisi paremmin ajan haasteisiin ja palvelisi sen käyttäjiä mahdollisimman hyvin. Järjestelmän käyttäjiin kuuluvat opiskelijat, opettajat (substanssiopettajat ja kielenopettajat) sekä harjoitteluinsinööri.

Keskeiseksi ongelmaksi järjestelmän suunnittelussa ja toteutuksessa muodostui käyttäjien roolien huomioonottaminen ja datan suojaaminen siten, että kukin osapuoli saa nähdä ja käsitellä vain sellaista tietoa, jota hän tarvitsee oman osuutensa hoitamiseen opiskelijan harjoitteluprosessissa. Vaikka työn aikana kehitetty järjestelmän asiakasohjelma onkin luotettu, ei voida olla varmoja, minkälaisia kutsuja rajapintaan tulee ulkoisista lähteistä: kuka tahansa voi haluamallaan asiakasohjelmalla tehdä mitä hyvänsä rajapinnan tukemia pyyntöjä – sellaisiakin, joita järjestelmän toteutuksen asiakasohjelma ei tee. Tästä syystä rajapinnan palveluita toteutettaessa oli kiinnitettävä erityistä huomiota siihen, että dataa ei päädy sellaiselle käyttäjälle, jonka ei siihen pitäisi käsiksi päästä.

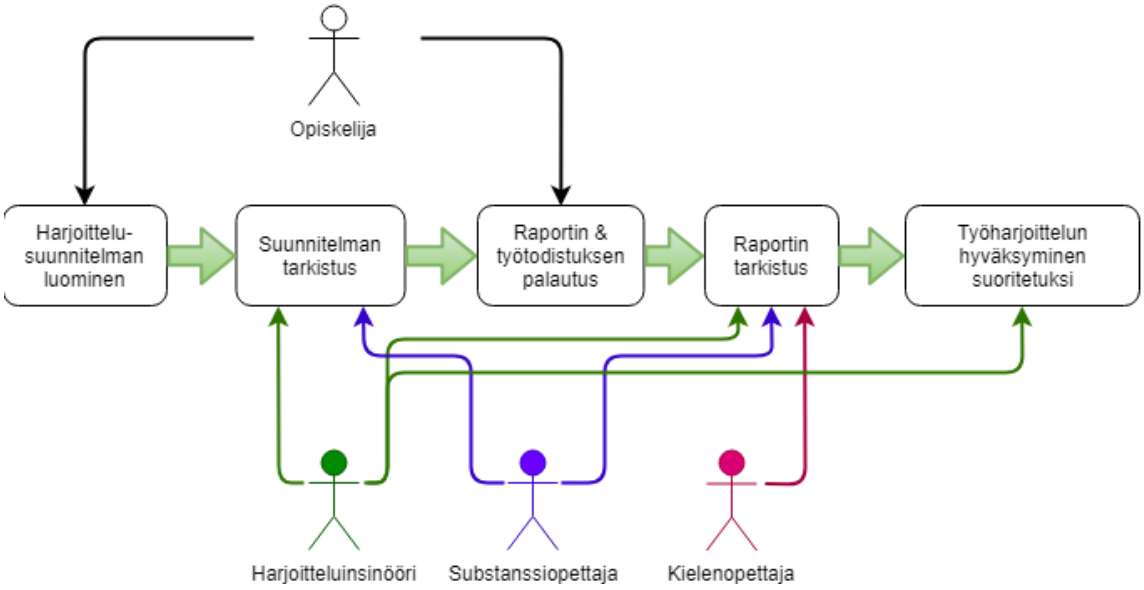
Työn teoriaosuudessa kuvataan harjoitteluprosessin perusteella syntyvät järjestelmän käyttötapaukset. Lisäksi esitellään toteutuksessa käytetyt teknologiat.

Työn toteutuksen kuvauksessa käydään ensin läpi olennaiset käsitteet, joita toteutuksessa käytetään. Sen jälkeen kerrotaan järjestelmän yleisestä arkkitehtuurista. Arkkitehtuurin kuvausta seuraa palvelinsovelluksen toteutus, jossa paneudutaan erityisesti autentikaation ja autorisoinnin toteutukseen. Tämän jälkeen kuvataan front-end-sovelluksen toteutusta teknisesti sekä viimein esitellään loppukäyttäjälle näkyvän käyttöliittymän oleellisimmat osat ja sovelluksen toiminta sitä kautta.

2 JÄRJESTELMÄN KÄYTTÖTAPAUKSET

2.1 Käyttäjien roolit ja yleiset käyttötapaukset

Harjoitteluportaalin käyttäjät jakaantuvat neljään eri rooliin: **opiskelijat, aineenopettajat**, **kielenopettajat** sekä **harjoitteluinsinööri**. Kaikki kirjautuvat järjestelmään oppilaitoksen AD-tunnuksilla. Kullakin on harjoitteluprosessissa omat tehtävänsä (KUVIO 1), joiden hoitamiseen järjestelmän tulisi pystyä. Kaikkien osapuolien tulee saada sähköpostiin ilmoitukset harjoittelusuunnitelmien tilaan liittyvistä muutoksista. Tällaisia ovat esimerkiksi suunnitelman hylkäys, hyväksyminen, dokumenttien (harjoitteluraportti ja työtodistus) palautus ja lopulta opintopisteiden myöntäminen hyväksyttävästi suoritetusta harjoittelusta.



KUVIO 1. Harjoitteluportaalin käyttötapaukset

2.2 Opiskelija

Kun opiskelija solmii työharjoittelusopimuksen, tulee hänen tehdä harjoittelusuunnitelma. Suunnitelman luominen alkaa harjoittelupaikan tietojen tallentamisella. Näihin tietoihin lukeutuvat harjoittelupaikan nimi, harjoittelupaikan yhteyshenkilön tiedot sekä harjoittelun ajankohta. Harjoittelusuunnitelmassa opiskelija asettaa itselleen tavoitteet harjoittelujaksolle. Harjoittelun tavoitteiden tekemisen apuna on valmiiksi muotoiltuja tavoitteita neljästä kategoriasta:

- Oma työpanos ja oppiminen
- Urasuunnittelu
- Työelämä- ja organisaatiotaidot
- Vuorovaikutustaidot.

Valmiiksi muotoiltuja tavoitteita on kussakin kategoriassa 16 kappaletta, joista opiskelija valitsee 5 parhaiten itselleen sopivaa. Näiden lisäksi opiskelija asettaa itselleen vähintään kolme ammatillista tavoitetta, jotka opiskelija itse sanallisesti muotoilee. Valmis harjoittelusuunnitelma lähetetään eteenpäin hyväksyttäväksi.

Kun harjoittelujakso on ohi, opiskelijan tulee palauttaa harjoitteluraportti sekä -todistus. Jos raportissa tai todistuksessa on puutteita, opiskelija saa tiedon näistä sähköpostin kautta. Harjoitteluportaalin raporttinäkymästä hän voi ladata hylkäykseen liittyvät liitetiedostot ja lukea hylkäykseen liittyvän sanallisen palautteen. Kun opiskelija on tehnyt pyydetyt korjaukset, hän palauttaa uuden version halutuista tiedostoista.

2.3 Harjoitteluinsinööri

Kun opiskelijan harjoittelusuunnitelma on valmis, se siirtyy harjoitteluinsinöörin, joka on myös järjestelmän pääkäyttäjä (admin), työjonoon. Harjoitteluinsinööri käy läpi kaikki harjoittelun tavoitteet ja tarkistaa, että suunnitelman tekoon on käytetty riittävästi aikaa. Harjoitteluinsinööri voi hylätä suunnitelman, jolloin se palautuu takaisin opiskelijalle mahdollisen sanallisen palautteen kera.

Harjoittelun jälkeen opiskelijan palauttaman harjoitteluraportti ja -todistus siirtyvät ensimmäisenä harjoitteluinsinöörin tarkistettavaksi. Jos näissä on puutteita, harjoitteluinsinööri voi hylätä palautetut liitetiedostot sanallisen palautteen kera. Hylkäyksen mukana hän voi sanallisen palautteen lisäksi palauttaa liitetiedostoja, kuten alkuperäisen raportin, johon on merkitty toivotut korjauspyynnöt.

2.4 Substanssiopettaja

Kun harjoitteluinsinööri on hyväksynyt opiskelijan tekemän harjoittelusuunnitelman, se siirtyy substanssiopettajan työjonoon. Opettaja tarkistaa suunnitelmasta opiskelijan

asettamat ammatilliset tavoitteet. Jos tavoitteet ovat puutteellisia, voi opettaja harjoitteluinsinöörin tavoin lähettää suunnitelman takaisin opiskelijalle palautteen kera.

2.5 Kielenopettaja

Harjoitteluinsinöörin sekä substanssiopettajan hyväksyttyä harjoitteluraportin ja -todistuksen siirtyy se kielenopettajan tarkistettavaksi. Kielenopettaja tarkistaa harjoitteluraportin oikeakielisyyden ja pyytää siihen tarvittaessa korjauksia. Kielenopettaja palauttaa hylätyn raportin korjauspyyntöineen järjestelmään, minkä jälkeen opiskelija saa tiedon hylätystä raportista sähköpostilla ja voi ladata korjauspyynnöt sisältävän tiedoston järjestelmän kautta.

3 KÄYTETYT TEKNOLOGIAT

3.1 JavaScript

Koko järjestelmän toteutuskielenä on JavaScript, jota selaimessa suorittaa selaimen JavaScript-moottori, ja palvelimella Node.js (ks. 3.7). Kielestä käytetään selainsovelluksen osalta versiota ES6 (ECMAScript 6) ja palvelimella versiota, joka tukee muun muassa async/await-avainsanoja, joiden avulla asynkronista koodia voidaan kirjoittaa ilman takaisinkutsujen tai Promise-objektin käyttöä.

3.2 REST

REST (Representational State Transfer) on tapa määritellä ja toteuttaa rajapintoja, joiden avulla verkossa toimivat hajautetun arkkitehtuurin sovelluksen osat voivat keskustella keskenään. REST perustuu erilaisten resurssien käyttöön. Resursseja voisivat olla vaikkapa käsitteet, kuten ”käyttäjä” tai ”viesti”. REST-rajapinta tarjoaa URI-osoitteen (Universal Resource Identifier) jokaiselle resurssille. (Representational State Transfer 2019.)

Resurssit näkyvät asiakkaalle HTTP-skeeman URI-osoitteina (ks. 5.1), joihin se tekee pyyntöjä. Palvelin lähettää pyynnön suorittamisen jälkeen asiakkaalle vastauksen (response), joka kertoo, voitiinko pyyntö toteuttaa, ja mahdollisen vastaukseen liittyvän datan, esimerkiksi pyydetyn resurssin tiedot tai luodun uuden resurssin URI-osoitteen.

REST-malli ei ota kantaa siihen, missä muodossa data välitetään rajapinnalta asiakkaalle ja toisinpäin. Data voidaan siirtää verkon yli esimerkiksi XML- tai JSON-muotoon serialisoituna.

3.3 HTTP (Hypertext Transfer Protocol)

REST-rajapinnan toteutuksessa yleisimmin käytetty protokolla on HTTP (Representational State Transfer 2019). Se perustuu pyyntöihin (request) sekä vastauksiin (response). Pyynnöissä käytetään erilaisia metodeja, joiden semantiikka määritellään REST-rajapinnan toteutuksessa. Yleisimmin käytetyt metodit ovat GET, POST, PUT ja DELETE. Näillä voidaan hakea, luoda, päivittää sekä poistaa tietoa, tai oikeammin pyytää näitä asioita tapahtuvaksi. HTTP-kutsuun saatu vastaus kertoo numeerisella HTTP-standardissa määritellyllä paluuarvolla, onnistuiko pyyntö vai ei. Tavallisia paluuarvoja ovat esim. 200 (OK), 404 (Not found) ja 500 (Internal Server Error).

3.4 JSON (JavaScript Object Notation)

JSON on datan serialisointiin tarkoitettu notaatio, jossa data siirretään tekstimuotoisina JavaScript-objektien kuvauksina. JavaScript-objektit ovat helposti muunnettavissa verkon yli siirtoa varten tekstimuotoon ja vastaavasti tekstimuotoinen JSON-data on yksinkertaista muuntaa takaisin JavaScript-objektiksi. (JSON 2019.) JSON-objekti voi olla joko avain-arvo-pareja sisältävä objekti tai taulukko (array). Objektin avainten arvot ja taulukon alkiot voivat olla merkkijonoja, boolean-arvoja, null, numeroita, tai sisäkkäisiä JSON-objekteja, jolloin voidaan ilmaista hierarkkisia tietorakenteita. (KUVIO 2.)

```
{
  "userId": 123,
  "name": "Olli",
  "needsCoffee": true,
  "plans": [
    {
      "name": "Oppari loppuun",
      "description": "Melkein valmis jo!"
    },
    {
      "name": "Kahvi",
      "description": null
    }
  ]
}
```

KUVIO 2. JSON-objekti

3.5 JWT

JWT (JSON Web Token) on standardi pääsyoikeuksien todentamiseen kahden osapuolen välillä. JWT koostuu ”väittämistä” (claim), jotka määrittelevät, mitä JWT:n haltijalla on oikeus tehdä annetussa kontekstissa (Jones, Microsoft, Bradley, Ping Identity, Sakimura, NRI 2015). JWT voi esimerkiksi sisältää tiedon siitä, kuka sen on luonut ja kuinka kauan se on voimassa. Näiden JWT-standardissa määriteltyjen väittämien lisäksi JWT:n mukaan voidaan liittää esimerkiksi käyttäjää kuvaava JSON-objekti, jossa kerrotaan käyttäjän rooli ja muita käyttäjää koskevia tietoja.

JWT voidaan joko salata tai digitaalisesti allekirjoittaa. Digitaalinen allekirjoitus takaa sen, että JWT:n muokkausyritykset voidaan tunnistaa sitä varmennettaessa. JWT siirtyy

osapuolien välillä base64url-koodatussa muodossa, jossa se on helppo liittää esimerkiksi HTTP-pyynnön otsikkotietoihin.

Valmis verkon yli siirrettävä JWT koostuu kolmesta osasta: otsikkotiedoista, varsinaisesta JWT:n mukana kulkevasta datasta sekä allekirjoituksesta. JWT muodostetaan siten, että otsikkotiedot sekä data base64url-koodataan ja erotetaan toisistaan pisteellä, minkä jälkeen näiden perään liitetään salausavain. Näin saatu merkkijono allekirjoitetaan salausavainta käyttäen valitulla algoritmilla ja saatu allekirjoitus lisätään otsikkotietojen jatkeeksi pisteellä eroteltuna. (KUVIO 3.)

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4
gRG9lIiwiaXNTb2NpYWwiOnRydWV9.
4pcPyMD09olPSyXnrXCjTwXyr4BsezdI1AVTmud2fU4
```

KUVIO 3. JWT verkon yli siirrettävässä muodossa (Introduction to JSON Web Tokens 2019)

3.6 MariaDB

MariaDB on relaatiotietokanta. Tiedot tallennetaan tauluihin (table), joista jokainen voi sisältää useita rivejä eli tietueita (row, record). Esimerkiksi tauluun ”users” tallennettaisiin opiskelijat. Kukin tietue sisältää opiskelijan tiedot, kuten nimen ja sähköpostisoitteen sekä opiskelijanumeron. Kullakin tietueella on lisäksi jokin yksilöivä tieto, joka on uniikki ja jonka perusteella haluttuun tietueeseen päästään käsiksi. Opiskelijan tapauksessa tällainen tieto on opiskelijanumero. Yksilöivää tietuetta nimitetään primääriavaimeksi (primary key).

Kun tietue halutaan yhdistää liittyväksi toiseen, se tehdään viiteavaimella (foreign key). Jos tietokannassa on taulu ”packs” (ks. 4), johon tallennetaan työharjoittelua koskevat tiedot, lisätään siihen viiteavain, jonka arvo kussakin tietueessa vastaa halutun oppilastietueen primääriavainta. Näin tiedetään, mikä harjoittelu kuuluu kullekin opiskelijalle.

Tietokantaan tehdään kyselyitä SQL-kielellä. Kyselyiden avulla voidaan lisätä, poistaa ja muokata olemassa olevia taulujen rivejä.

3.7 Node.js

Node.js on alustariippumaton JavaScript-moottori, jonka avulla JavaScript-koodia voidaan suorittaa selaimen ulkopuolella. Se tarjoaa palveluita mm. verkkosovellusten luomiseen. Oleellinen osa Node.js-ekosysteemiä on NPM (Node Package Manager), jonka avulla sovelluksen käyttämiä Node.js:lle kehitettyjä kirjastoja hallitaan (Node.js 2019). Se pitää mm. huolen, että kun jokin paketti asennetaan, asennetaan samalla kaikki sen tarvitsemat paketit.

3.8 Knex.js

Knex.js on relaatiotietokantojen kyselyiden rakentamiseen tarkoitettu kirjasto. SQL-kyselyt voitaisiin kirjoittaa yhtä hyvin myös käsin, mutta Knex.js:n kaltainen kirjasto helpottaa varsinkin monimutkaisten kyselyiden rakentamista annettujen parametrien perusteella, kun kyselyobjektiin voidaan ketjuttaa uusia metodikutsuja tarpeen mukaan, eikä SQL-kyselyä tarvitse rakentaa yhdistelemällä merkkijonoja itse (KUVIO 4). Knex.js:n kyselyn rakennusfunktiot huolehtivat myös kyselyn parametrien turvallisesta käsittelystä, joka estää SQL-injektioiden toteutumisen.

```
let query = knex('users').orderBy('lastName');

if (query.roleName)
  query = query.where({ role: roleName });

if (query.skip)
  query = query.offset(query.skip);

if (query.take)
  query = query.limit(query.take);

// Run the final query
let filteredStudents = await query;
```

KUVIO 4. Esimerkki SQL-kyselyn rakentamisesta parametrien perusteella

3.9 Express.js

Node.js:n päällä toimiva Express.js-sovelluskehys on suunniteltu web-sovellusten ja -rajapintojen toteuttamiseen (Express 2019). Se kuuntelee verkon yli saapuvia HTTP-pyyntöjä ja suorittaa käyttäjän näille määrittelemät toiminnot. Expressin reititysfunktioiden avulla

voidaan esimerkiksi määritellä, mitä tapahtuu, kun palvelimelle saapuu pyyntö URI-osoitteeseen ”/api/users” HTTP-metodilla GET.

Expressin pyyntöjen käsittely pohjautuu ns. middleware-ketjuun. Ketjuun määritellyt funktiot suoritetaan peräkkäin, kun sovellukselle johonkin reittiin tuleva pyyntö käsitellään. Viimeisenä suoritetaan varsinainen kontrollerifunktio, jonka tehtävänä on käsitellä pyyntö ja vastata asiakkaalle. Jos muita middleware-funktiota ei ole, on kontrollerifunktio ketjun ainoa funktio. Ennen ketjun viimeistä funktiota voidaan esimerkiksi autentikoida pyynnön tekevä käyttäjä ja lopettaa pyynnön käsittely, jos autentikaatio ei onnistunut.

Middleware-funktiolle annetaan seuraavat parametrit:

- sisään tuleva pyyntö (objekti, joka kuvaa pyynnön)
- vastausobjekti, jonka kautta pyynnön käsittely voidaan lopettaa tarpeen vaatiessa
- funktio, jota middlewaren tulee kutsua, jotta seuraava ketjussa oleva middleware suoritetaan. Ilman tätä pyynnön käsittely jumiutuu. Viimeisenä ketjussa oleva middleware, eli varsinainen kontrolleri, ei kutsu tätä funktiota, vaan päättää pyynnön käsittelyn vastusobjektia käyttämällä.

Näille parametreille annetaan yleensä koodissa nimet **req**, **res**, ja **next** (Writing Middleware 2019).

3.10 React

React on JavaScript-kirjasto käyttöliittymien ohjelmointiin. Sen käyttö perustuu komponenttien ohjelmointiin, joita yhdistelemällä React-sovellus rakennetaan (React 2019). Sovellus paketoidaan yhdeksi JavaScript-tiedostoksi, joka voidaan ladata normaalisti HTML-sivulle script-elementtiä käyttäen. React-sovelluksen juurikomponentti asetetaan HTML-sivulle ReactDOM-moduulin render-funktiolla, jolle kerrotaan, minkä HTML-elementin sisään sovellus sijoitetaan (ks. 6.2).

3.10.1 Komponentin tila ja parametrit (state, props)

Komponentti voi pitää kirjaa sisäisestä tilastaan. Komponentin alkutila määritellään komponenttiluokan konstruktorissa, ja myöhemmät tilan muutokset tehdään Reactin setState-funktiota käyttäen.

Kukin komponentti voi rakentua pienemmistä komponenteista. Kommunikaatio komponenttien välillä on yksisuuntaista: isäntäkomponentti välittää tarvittavat tiedot lapsikomponenteilla käyttäen komponentin props-ominaisuutta. Käytännössä tämä näyttää samalta kuin HTML-tageille parametrien välittäminen attribuutteina. Annetut parametrit ovat komponentin käytettävissä props-nimisen objektin avaimina. Jos komponenttia ajattelee funktiona, niin props-objektin voi silloin ajatella funktion parametreiksi. Jos komponentti ei käytä sisästä tilaa, niin React salliikin määritellä komponentin funktiona, jonka ainoa parametri on nimeltään props.

Props ja state yhdessä ohjaavat komponentin toimintaa ja määrittelevät miten komponentti piirretään. React pitää huolen siitä, että aina kun jommassakummassa näissä oleva tieto päivittyy, piirretään komponentti uudelleen. React optimoi piirtämisen siten, että vain sivun muuttuneet osat päivitetään (Reconciliation 2019).

3.10.2 Komponentin elinkaari

Komponentti määritellään luokkana, jonka toimintaa ohjaavat komponentin elinkaareen liittyvät metodit (life-cycle methods). React kutsuu näitä metodeja, kun käyttäjä tai muu ulkoinen tekijä on interaktiossa näytettävän sivun kanssa.

Kun React haluaa piirtää komponentin, se kutsuu tämän render-metodia. Render-metodin tulee palauttaa React-elementti, joka kuvaa komponentin näytettävän rakenteen. Toinen usein käytetty elinkaarimetodi on componentDidMount, jota kutsutaan juuri ennen render-funktion ensimmäistä kutsua, ts. silloin kun komponentti on ”asennettu” sivulle, mutta sitä ei ole vielä piirretty. (React.Component 2019.) Tässä funktiossa haetaan palvelimelta tarvittava data ja viimeistellään komponentin tila.

3.10.3 JSX

Kun React haluaa piirtää komponentin, se kutsuu komponentin render-metodia. Render-metodin tulee palauttaa React-elementti, jonka React osaa käsitellä. Render-metodi voi toteuttaa tämän käyttämällä createElement-metodia, jolle annetaan argumenttina objekti, joka kuvaa sivun rakenteen.

Sen sijaan, että käytettäisiin createElement-metodia ja sivun rakenne kuvattaisiin sisäkkäisinä JavaScript-objekteina, voidaan käyttää myös HTML:ltä näyttävää JSX-kuvauskieltä. Se on vaihtoehtoinen tapa kuvata React-komponenttien tuottama JavaScript, jonka

perusteella sivu piirretään. JSX käännetään createElement-kutsut tekeväksi JavaScript-koodiksi sovelluksen käännösvaiheessa. JSX:ää käytettäessä komponentit kuvataan tageilla, joiden väliin ja joiden attribuuttien arvoihin voi upottaa JavaScript-lauseita aaltosulkujen sisään. (KUVIO 5.)

```
render() {
  if (!this.props.category)
    return null;

  let { cards } = this.props.category;

  return(
  <div>
    <h2>{this.props.category.title}</h2>
    {this.props.showInstructions &&
      <p>Valitse tästä kategoriasta viisi korttia, jotka kuvaavat harjoittelujakson tavoitteitasi parhaiten.</p>}

    {cards.map(c =>
      <Card
        card={c}
        onClick={this.props.cardClicked}
        style={this.getCardStyle(c, this.props.category)}
        key={c.cardId}
      />
    )}
  </div>);
}
```

KUVIO 5. JSX-koodia komponentin render-metodissa

3.11 Axios

Axios on HTTP-asiakaskirjasto. Kun selain haluaa tehdä asynkronisia HTTP-kutsuja, se käyttää XmlHttpRequest-objektia (joka nimestään huolimatta ei ole sidottu XML-muotoiseen dataan). Axios huolehtii XmlHttpRequest-objektin käsittelystä ja tarjoaa ohjelmoijalle mukavamman rajapinnan HTTP-kutsujen tekemiseen ja vastausten käsittelyyn.

Axios-kutsu palauttaa Promise-objektin, joka kuvaa asynkronista operaatiota. Sen onnistunut tulos saadaan käsiteltyä objektin then-metodissa, jonka parametrina on vastauksen tuottama data. Virhetilanteet saadaan käsiteltyä catch-metodissa, jonka perusteella voidaan näyttää käyttöliittymässä virheilmoitus tai muuten reagoida virheeseen.

4 TOTEUTUS YLEISESTI

4.1 Toteutuksen käsitteet

Harjoitteluportaalin pääkäyttäjä (**admin**) on harjoitteluinsinööri, jonka vastuulla on hyväksyä harjoittelusuunnitelma ja suoritetun harjoittelun jälkeen myöntää harjoittelusta opintopisteet. Tietokannassa ja lähdekoodissa käsite **user** tarkoittaa ainoastaan opiskelijaa. Opettajat sekä pääkäyttäjät kuvataan omalla **staff**-käsitteellä.

Opiskelijan harjoittelujaksoa kuvataan ”harjoittelupaketin” käsitteellä. Tietokannassa ja koodissa tämä esiintyy nimellä **pack**. Se sisältää yleiset tiedot harjoittelusta, kuten harjoittelupaikan nimen, harjoittelun ajanjakson sekä yhteyshenkilön tiedot. Pack on aina jossain tilassa (**status**, ks. 5.2). Mahdollisia tiloja ovat esimerkiksi ”odottaa harjoittelusuunnitelman luomista”, ”harjoittelusuunnitelma odottaa harjoitteluinsinöörin hyväksyntää” tai ”harjoittelu valmis”.

Harjoittelusuunnitelma – eli **plan** – sisältää opiskelijan asettamat tavoitteet harjoittelujaksolle ja liittyy aina johonkin harjoitteluun eli ”packiin”.

4.2 Arkkitehtuuri

Järjestelmä perustuu hajautettuun REST-tyyppiseen arkkitehtuuriin, jossa sovelluksen käyttöliittymäkerros – eli front-end – erotetaan sen ns. back-end -logiikasta, joka on itsenäinen palvelinsovellus. Mallissa kumpikaan kerros ei ole riippuvainen toistensa toteutuksesta eikä niiden tarvitse tietää toistensa sisäisestä toiminnasta mitään.

Hajautetun arkkitehtuurin etuna on sen modulaarisuus: kumpikin osa voidaan korvata toisella, kunhan näiden välissä oleva rajapinta säilyy samana. Käytännössä palvelinsovellus toteuttaa rajapinnan, jota kautta asiakassovellus (client) pyytää tarvitsemaansa dataa ja lähettää käyttäjän tekemien toimintojen aiheuttamien muutoksien perusteella pyyntöjä päivittää dataa. Koko sovelluksen taustalla on relaatiotietokanta, jonka kanssa back-end -kerros keskustelee suoraan.

Kaikki sovelluksen kerrokset voivat sijaita fyysisesti eri palvelimilla. Toteutuksessa tietokantapalvelin sijaitsee palvelinsovelluksen kanssa samalla palvelimella. Käyttöliittymäsovellusta suorittaa järjestelmän käyttäjän asiakassovellus. Toteutuksessa tämä on selaimessa suoritettava JavaScript-ohjelma, mutta se voisi yhtä hyvin olla esim. puhelimessa tai muussa mobiililaitteessa toimiva sovellus.

Toteutuksessa kaikki mahdollinen ns. business-logiikka hoidetaan palvelinsovelluksessa. Asiakasohjelma on mahdollisimman ”tyhmä”, ja palvelinsovelluksen vastuulla on tarjota harjoitteluprosessin hoitamiseen tarvittavat palvelut sen rajapinnan kautta.

5 BACK-ENDIN TOTEUTUS

5.1 API:n rakenne

Harjoitteluportaalin palvelinsovelluksen tarjoaman rajapinnan rakenne noudattaa hierarkista muotoa, jossa käsiteltävä resurssi ilmoitetaan URI-osoitteessa ensin. Jos käsitellään tiettyä resurssia, sen yksilöivä tunniste seuraa hierarkiassa seuraavana. Mahdollinen resurssiin liittyvä aliresurssi seuraa taas tätä saman kaavan mukaan. API-osoitteiden etuliite on ”/api”, jolloin koko rajapinta on tavoitettavissa osoitteesta https://harjoittelu.lamk.fi/api.

Oheisessa taulukossa (TAULUKKO 1) on esitetty joitakin esimerkkejä osoitteista, joista API vastaa. PUT- ja POST-metodeilla määritellyissä metodeissa pyyntöön liittyy parametreja, jotka välitetään pyynnön sisältöosassa (request body) JSON-muodossa. Esimerkiksi /api/authenticate-pyynnön mukana tulee lähettää käyttäjätunnus ja salasana. Mikäli vaadittuja parametreja ei lähetetä, tai jos API-osoitteen resurssin yksilöivä tunniste on väärässä muodossa, palauttaa rajapinta HTTP-vastauksen 400 (Bad Request). Vastauksen mukana palautetaan JSON-objektin sisällä virhettä tarkemmin kuvaava virheviesti (esim. ”Invalid user id”).

GET-kutsujen parametrit välitetään URL-parametreina. Näin esimerkiksi /api/users-osoitteen palauttamaa käyttäjälistaa voi rajata opiskelijan nimen, opiskelijanumeron, sähköpostiosoitteen ja muiden opiskelijaan liittyvien tietojen perusteella. Esimerkkinä /api/users?lname=rouvinen&group=07tvt15, joka hakee kaikki Rouviset, jotka kuuluvat opiskelijaryhmään 07TVT15.

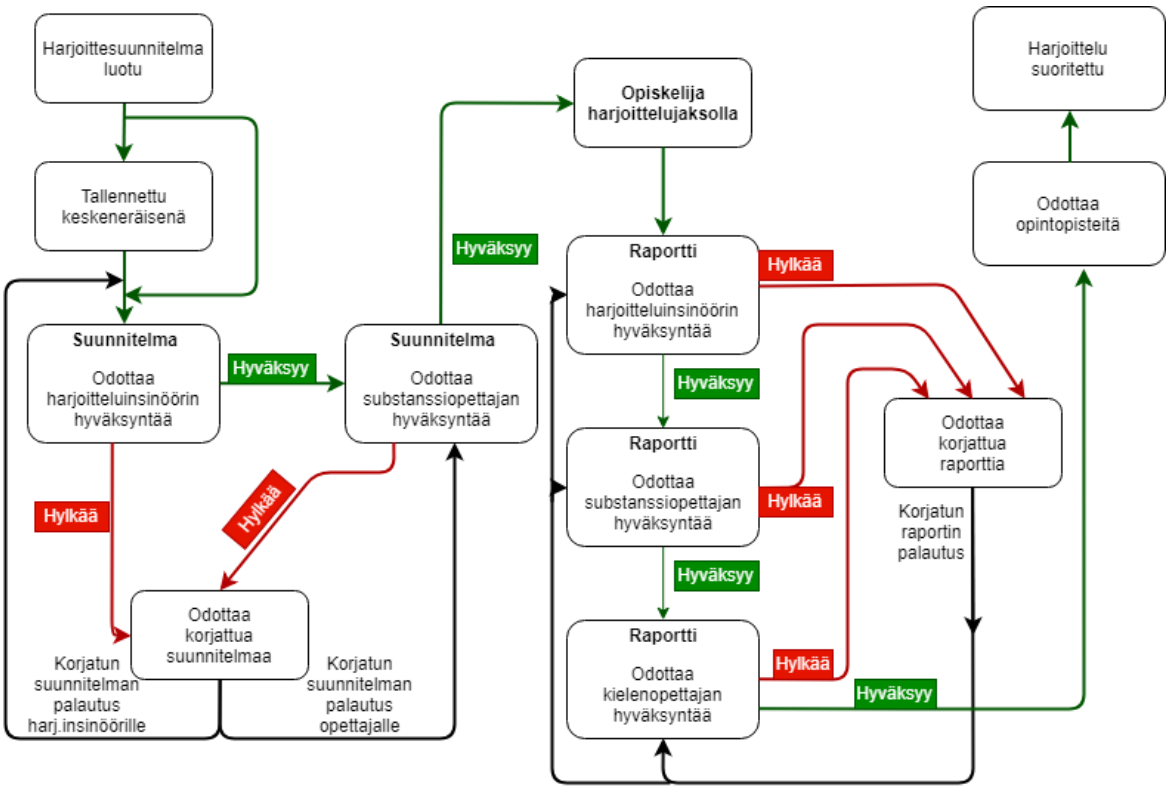
TAULUKKO 1. Ei-täydellinen lista rajapinnan osoitteista

HTTP-metodi	URI	Kuvaus
POST	/api/authenticate	Asiakkaan autentikointi
GET	/api/users	Käyttäjälista (haku)
GET	/api/users/{id}	Käyttäjän tiedot
GET	/api/users/{id}/packs	Käyttäjän harjoittelupaketit
PUT	/api/users/{id}	Päivittää käyttäjän tiedot
GET	/api/packs	Harjoittelupakettien haku (pääkäyttäjälle)
POST	/api/packs	Harjoittelupaketin luonti

GET	/api/packs/{id}/plan	Suunnitelman haku
GET	/api/plans/{id}/cards	Suunnitelman tavoitekortit
DELETE	/api/notes/{id}	Oppilasta koskevan muistiinpanon poistaminen
PUT	/api/packs/{id}/nextstatus	Asettaa harjoittelun seuraavaan tilaan

5.2 Harjoittelun tilan hallinta

Kun harjoittelu on luotu harjoitteluportaaliin, on se koko työharjoitteluprosessin ajan aina jossain tilassa (KUVIO 6). Tila tallennetaan numeerisena arvona, joka kertoo, mitä harjoitteluprosessissa tulee seuraavaksi tapahtua, että se etenee.



KUVIO 6. Harjoitteluprosessin mahdolliset tilat

Tilat ja niiden numeeriset arvot ovat seuraavat:

0. Suunnitelma on luotu, mutta sen tekoa ei ole vielä aloitettu.
1. Suunnitelma on tallennettu keskeneräisenä.
2. Suunnitelma odottaa harjoitteluinsinöörin hyväksyntää.
3. Suunnitelma odottaa substanssiopettajan hyväksyntää.
4. Suunnitelma on hyväksytty, harjoittelujakso on meneillään.
5. Järjestelmä odottaa harjoitteluraportin palautusta.
6. Raportti odottaa harjoitteluinsinöörin hyväksyntää.
7. Raportti odottaa substanssiopettajan hyväksyntää.
8. Raportti odottaa kielenopettajan hyväksyntää (kieliasun tarkistus).
9. Järjestelmä odottaa, että harjoitteluinsinööri merkitsee opintopisteet kirjatuksi.
10. Opintopisteet on kirjattu ja harjoittelu on suoritettu.

Näiden lisäksi käytetään arvoja -2 ja -1 kun halutaan ilmaista, että odotetaan korjauksia joko harjoittelusuunnitelmaan tai -raporttiin. Kun prosessi etenee normaalisti, tilan arvoa kasvatetaan yhdellä, jolloin seuraava tila aktivoituu. Kun opettaja tai harjoitteluinsinööri pyytää korjauksia suunnitelmaan tai raporttiin, tallennetaan tieto edellisestä tilasta tietokannan pack-tietueeseen (ks. 4). Kun opiskelija tekee pyydetyt korjaukset, tila palautetaan takaisin korjauspyyntöä edeltävään tilaan.

Korjauspyyntöjen tilat käyttävät negatiivisia arvoja, jotta prosessin edetessä uuden tilan asettaminen on yksinkertaista tehdä vain tilan arvoa kasvattamalla. Tietyn tilan tarkistamiseen ei tarvitse käyttää koodissa sen numeerista arvoa, vaan arvot yhdistetään symboliseen nimeen JavaScript-objektin avulla. Siten koodissa tilan arvoa ei tarvitse käsitellä numeerisilla arvoilla, vaan tilaan voidaan viitata sille annetulla nimellä (KUVIO 7). Koodi on näin helppolukuisempaa ja helpommin ylläpidettävää, kun tilan mahdolliset arvot määritellään vain yhdessä paikassa.

```
if (pack.status === 5) {
  // käsittely
}

if (pack.status === ProcessStatus.waitingForReport) {
  // käsittely
}
```

KUVIO 7. Tilan käsittely numeerisella arvolla ja symbolisella nimellä

Käytännössä tilan muutoksissa joudutaan ottamaan huomioon muutoksen aiheuttavan käyttäjän rooli sekä se, kuuluuko kyseessä olevan harjoittelun raportti kielenopettajan tarkistuksen piiriin (tarkistus määritellään ryhmäkohtaisesti). Kun käyttäjä tekee PUT-pyynnön osoitteeseen /api/pack/{id}/nextstatus, päivitetään tila seuraavaan tilaan. Palvelinsovellus huolehtii siitä, että tila päivitetään oikein edellä mainitut asiat huomioon ottaen.

5.3 Autentikointi ja autorisointi

Autentikoinnilla (authentication) tarkoitetaan käyttäjän identiteetin todentamista. Ts. autentikaatiolla varmistetaan, että käyttäjä on todella se kuka hän väittää olevansa. Harjoitteluportaalin autentikaatio on ulkoistettu LAMK:n ActiveDirectory-hakemistopalvelulle (AD). Käyttäjätunnuksena toimii käyttäjän AD-tunnus (ks. 5.3.1) ja salasanana vastaavasti AD-tunnuksen salasana. Nämä varmistetaan AD:n LDAP-rajapinnasta kirjautumisen yhteydessä.

Kun asiakas tekee pyyntöjä palvelimelle, pitää tarkistaa, onko hänellä oikeus päästä käsiksi pyydettyyn resurssiin. Tätä prosessia kutsutaan **autorisoinniksi** (authorization)**.** Hyvin usein autentikaatio on edellytys autorisoinnille, mutta aina näin ei ole – esimerkiksi kun kyseessä on kaikkien saatavilla oleva data. Julkista dataa voisivat olla esimerkiksi tiedotteet tai uutiset, joita halutaan näyttää, kun käyttäjä saapuu järjestelmän etusivulle, mutta ei ole vielä kirjautunut järjestelmään.

5.3.1 Autentikaation toteutus

REST-arkkitehtuurilla toteutetun palvelinsovelluksen eräs piirre on, että sovellus ei pidä yllä olemassa olevaa tilaa (state), vaan sen tuottamat vastaukset ovat täysin riippuvaisia sille sisään tulevista pyynnöistä. Palvelinsovellus ei siis pidä yllä sessiosta, jonka tila

muuttuisi asiakkaan tekemien pyyntöjen seurauksena. Tämä tarkoittaa sitä, että asiakkaan pitää välittää jonkinlainen osoitus voimassa olevasta autentikaatiosta jokaisen palvelimelle lähtevän pyynnön mukana.

Autentikaatio perustuu JWT:n käyttöön (ks. 3.5). Autentikaatio alkaa, kun asiakas lähettää HTTP:n POST-metodilla pyynnön palvelimen osoitteeseen /api/authenticate. Pyynnön mukana lähetetään käyttäjän AD-käyttäjätunnus sekä salasana. Palvelimelta käyttäjätunnus ja salasana välitetään tunnistautumista varten oppilaitoksen LDAP-rajapintaan, josta palvelinsovellus saa vastauksena tietueen, joka sisältää käyttäjän tiedot, mukaan lukien tämän rooli (”student” tai ”staff”).

Harjoitteluportaali valitsee vielä oman tietokantansa käyttäjätietojen perusteella käyttäjälle tarkemman roolin (pääkäyttäjä, substanssiopettaja, kielenopettaja tai opiskelija) ja luo valmiin käyttäjäobjektin, johon sisältyy käyttäjän sähköpostiosoite, nimi, rooli, sekä käyttäjän id, joka on sama kuin tietokannan käyttäjätaulun primääriavain. Käyttäjäobjektin perusteella luodaan allekirjoitettu tiketti, joka lähetetään takaisin asiakkaalle. Asiakkaan vastuulla on tästä eteenpäin pitää tiketti tallessa seuraavia pyyntöjä varten.

Kun palvelimelle saapuu pyyntö, joka vaatii autentikaatiota, palvelin odottaa löytävänsä koskemattoman ja voimassa olevan JWT-tiketin pyynnön otsikkotiedoista (request header) avaimen Authorization alta muodossa ”Bearer xxxxx.xxxx.xxx”, jossa Bearer-tunnisteen jälkeen seuraa palvelimelta saatu tiketti. Bearer-tunniste kertoo autentikaation tavan (Fielding, Adobe, Reschke, greenbytes 2014). Tikettiä ei ole salattu, mutta se on allekirjoitettu palvelimelle asetetulla salaisella avaimella, joten jos tiketin tietoja on yritetty muuttaa, se huomataan, kun tiketin oikeellisuutta varmistetaan. Varmistus tapahtuu ennen haluttua kontrolleria middleware-funktiossa, joka palauttaa asiakkaalle 401 (Unauthorized)-virhekoodin, mikäli tiketti ei ole kunnossa.

5.3.2 Autorisoinnin vaatimukset

Autorisoinnin toteutuksessa pitää ottaa huomioon pyynnön tekevän käyttäjän rooli seuraavasti:

- Jotkin resurssit eivät ole sallittuja kuin pääkäyttäjälle.
- Oppilaat eivät saa päästä käsiksi muihin kuin omiin tietoihinsa.
- Opettajat (niin substanssi-, kuin kielenopettajat) saavat nähdä vain ne resurssit, jotka liittyvät heidän työjonossaan oleviin harjoittelusuunnitelmiin ja raportteihin.

5.3.3 Käyttäjän rooliin perustuvan autorisoinnin toteutus

Käyttäjän rooliin perustuvaan pyyntöjen suodatukseen on toteutettu middleware-funktio authorizedFor. Sille kerrotaan argumentteina ne roolit, joihin kuuluvat käyttäjät saavat kyseiseen API-reittiin pyyntöjä tehdä. Mikäli pyynnön tekevän käyttäjän rooli ei ole sallittujen joukossa, funktio lopettaa pyynnön käsittelyn HTTP-koodilla 403. Muussa tapauksessa suoritus siirretään middleware-ketjun seuraavalle funktiolle. Pyynnön tekevän käyttäjän tiedot on asetettu req-objektiin aiemmin middleware-ketjussa autentikaation varmistavan funktion toimesta, joten se on saatavilla ketjun funktioiden req-parametrin kautta. (KUVIO 8.)

AuthorizedFor-funktion toteutuksessa huomionarvoinen seikka on, että se luo sulkeuman (closure), jonka avulla sen arguments-parametri on middleware-funktion käytettävissä, vaikka ulomman funktion (authorizedFor) suoritus on jo päättynyt siinä vaiheessa, kun sen palauttaman sisemmän funktion suoritus aloitetaan. (Closures 2019.)

Arguments-parametria ei erikseen esitellä funktion argumenttilistassa, sillä se on näkymätön parametri, joka on funktioiden käytössä. Se sisältää listan funktiolle annetuista argumenteista. (The Arguments Object 2019.)

```
// NOTE: needs the enclosing function to create closure over
// `arguments`
function authorizedFor() {
  return (req, res, next) => {
    let { user } = req;
    if (!user)
      return res.status(500).json({ error:  'missing authentication in middleware chain' });

    let allowedRoles = Array.from(arguments);

    if (!allowedRoles.find(r => r === user.role))
      return res.status(403).end();

    next();
  };
}
```

KUVIO 8. Middleware-funktio käyttäjäroolin perusteella autorisointiin

Kun authorizedFor-funktio on määritelty, voidaan sitä käyttää halutuissa reiteissä kertomalla reitin määrittelyssä mitkä roolit kyseiseen reittiin saavat pyyntöjä tehdä. Käyttäjän

roolin perusteella tapahtuva autorisointi on näin siirretty pois varsinaisten kontrollerien vastuulta. (KUVIO 9.)

```
app.get('/api/notes/:id', requireAuth, authorizedFor(roleNames.admin), notes.getById);
app.get('/api/users/:id/notes', requireAuth, authorizedFor(roleNames.admin), notes.getByUserId);
app.post('/api/notes', requireAuth, authorizedFor(roleNames.admin), notes.create);
app.put('/api/notes/:id', requireAuth, authorizedFor(roleNames.admin), notes.update);
app.delete('/api/notes/:id', requireAuth, authorizedFor(roleNames.admin), notes.remove);
```

KUVIO 9. API-reittien suojaus authorizedFor-funktiolla

5.3.4 Yksittäisen käyttäjän pyyntöjen autorisointi

Vaikka rooliin perustuva autorisointi on mahdollista siirtää pois kontrollerien vastuulta middleware-funktiota käyttämällä (ks. 5.3.3), ei samanlainen lähestymistapa onnistu yhtä helposti, kun pitää päättää, kuuluuko pyydetty resurssi todella sitä pyytäneelle käyttäjälle.

Tavoitteena oli minimoida kontrollereissa suoritettava autorisoinnista aiheutuva toistuva työ. Ongelmaa lähestyttiin sillä ajatuksella, että ohjelmoidessa olisi hyödyllistä, jos kontrollerissa voisi yhdellä funktiokutsulla pyytää haluttua dataa, ja sitä voisi sen jälkeen käsitellä sillä oletuksella, että pyynnön tekevä käyttäjä on siihen oikeutettu.

Kaikkien resurssimoduulien kontrollereissa toistuvat usein kaksi eri tehtävää:

1. **resurssijoukon** haku käsittelyä ja mahdollisesti jatkosuodatusta varten
2. **yksittäisen resurssin** haku käsittelyä varten.

Näitä varten tarvittiin kaksi eri funktiota. Yksi, jonka avulla voi luoda helposti kyselyn, joka suoritettaessa palauttaa kaikki pyyntöä tekevälle käyttäjälle kuuluvat resurssit, ja toinen, jolla haetaan yksittäinen käyttäjälle kuuluvan resurssi sen yksilöivän tunnisteen perusteella.

Funktio 1, joka luo kyselyn resurssijoukon hakua varten toimii pohjana funktiolle 2, joka hakee yksittäisen resurssin, sillä yksittäisen resurssin suodattava where-ehto on helppo lisätä resurssijoukon hakevaan kyselyyn. Ongelmaksi jää oikeastaan sen varmistaminen, että haetun resurssijoukon jokainen tietue kuuluu halutulle käyttäjälle.

Autorisoinnin lähtökohtana on, että kaikki autorisointia vaativa data yhdistyy relaatioiden kautta aina lopulta packs-tauluun, josta kunkin tietueen ”omistava” taho saadaan selvitettyä user_id- tai staff_id-kentän arvosta. Oppilaan tapauksessa se yhdistetään opiskelijaan user_id-kentän arvolla, mutta koska harjoittelun suunnitelman ja raportin tarkistus kuuluvat

aina jonkin opettajan (staff) vastuulle, tulee pääsyoikeus opettajien tapauksessa tarkistaa tietueen staff_id-kentän perusteella. Näin ollen jokaiseen autorisointia vaativaan kyselyyn pitää liittää osaksi packs-taulu SQL:n join-lauseella, jota Knex-kirjastossa vastaa funktio nimeltä ”join”. Kun packs on yhdistetty kyselyyn, voidaan kyselyä tekevän käyttäjän id:n perusteella valita kyselyn tuloksiin vain sallitut tietueet.

Ratkaisua auttamaan on määritelty funktio authFilter, joka luo sellaisen SQL-kyselyn where-lausetta kuvaavan objektin, joka suodattaa ei-sallitut tietueet kyselyn tuloksista pois (KUVIO 10). Where-lause määritellään Knex-kyselyobjektin metodilla where, jolle yhtäsuuruusvertailun voi määrittää suoraan avain-arvo-parina, esim. **packs.where({ user_id: 123 })**.

```
// User table prefix can be used to resolve ambigous userId references
// in case it is joined in from multiple tables (i.e. "packs" and "plans")
export function authFilter(user, userIdTablePrefix = null) {
  if (user.role === roleNames.admin)
    return true;

  let userIdKey =
    user.role === roleNames.student
      ? 'user_id'
      : 'staff_id';

  if (userIdTablePrefix)
    userIdKey = userIdTablePrefix + "." + userIdKey;

  let userIdValue =
    user.role === roleNames.student
      ? user.userId
      : user.staffId;

  let filter = { [userIdKey]: userIdValue };

  return filter;
}
```

KUVIO 10. AuthFilter-funktio

AuthFilter-funktion avulla on helppo toteuttaa funktio, joka luo kyselyn resurssijoukon hakemista varten; kyselyyn tulee vain ketjuttaa Knexin where-funktio, jolle välitetään authFilter-funktion palauttama objekti. Funktio on plan-resurssin tapauksessa toteutettu nimellä currentUserPlans. Yksittäisen resurssin hakeva funktio getPlan on toteutettu sen avulla (KUVIO 11).

```
function currentUserPlans() {
  let user = getUser();
  let plans = knex('plans AS pl')
    .join('packs AS pk', 'pk.pack_id', 'pl.pack_id')
    .where(authFilter(user, 'pk'))
    .select([
      'pl.plan_id', 'pl.mod_date', 'pl.start_date', 'pl.end_date',
      'pk.user_id', 'pl.pack_id'
    ]);
  return plans;
}

async function getPlan(planId) {
  let plan = (await currentUserPlans()
    .where({ plan_id: planId }))[0];

  return plan;
}
```

KUVIO 11. Käyttäjälle kuuluvien resurssien hakeminen

Kun kontrollerissa pyydetään ja/tai halutaan käsitellä jotakin harjoittelusuunnitelmaa, haetaan se käyttämällä näitä apufunktioita. Jos resurssia ei löydy, voidaan lopettaa pyyntö HTTP-koodilla 404 (Not Found). Muuten voidaan olla varmoja siitä, että ollaan käsittelemässä sellaista dataa, johon käyttäjällä on todella oikeus päästä käsiksi.

Muiden resurssityyppien kohdalla on määritelty vastaavanlaiset apufunktiot. Ilman näitä funktioita jokaisen resurssin kontrollerifunktiossa pitäisi jotenkin käsin suodattaa kyselyn tuloksista pois ei-halutut tietueet. Tämä lisäisi koodin toistoa ja ohjelmointivirheiden mahdollisuuksia.

5.4 Pyyntöjen käsittely

Pyynnön käsittelevä kontrollerifunktio on Expressin kannalta kuten mikä tahansa muu middleware-funktio. Se asetetaan middleware-ketjun viimeiseksi ja erona muihin ketjun funktioihin on se, että tällainen funktio ei kutsu middleware-ketjua jatkavaa next-funktiota, vaan käyttää sille välitettyä vastausobjektia pyynnön päättämiseen. Vastausobjektin avulla voidaan asettaa vastauksen otsikkokentät, HTTP-vastauskoodi sekä mahdollinen pyydetty data. Data palautetaan JSON-muodossa ja siihen käytetään Expressin sisäänrakennettua json-funktiota, joka huolehtii datan serialisoinnista verkon yli siirtoa varten ja vastauksen Content-Type -otsikkokentän arvosta, jonka tulee olla application/json. (Bray, Textuality, 2006). (KUVIO 12.)

src/api/groups.js:

```
export const byId = async (req, res) => {
  let groupId = parseInt(req.params.groupId, 10);
  if (!groupId)
    return res.status(400).json({ error: "invalid group id" });

  try {
    let group = await getGroupById(groupId);
    if (!group)
      res.status(404).end();
    else
      res.json(group);
  } catch(err) {
    res.status(500).json({ error: err.message });
  }
};
```

src/main.js:

```
import * as groups from './api/groups';

app.get('/api/groups/:groupId', requireAuth, unauthorizedFor(roleNames.student), groups.byId);
```

KUVIO 12. Kontrollerifunktion toteutus ja asetus API-polkuun

5.5 Parannukset

Back-end sovelluksen modulaarisuus ei ole aivan loppuun asti viety. Kontrollerin tehtävänä tulisi olla sisään tulevan pyynnön käsittely ja siihen vastaaminen. Nykyisessä toteutuksessa yksittäisen kontrollerifunktion vastuulla on kuitenkin myös tarvittavan datan hakeminen tietokannasta ja sen mahdollinen muokkaus oikeaan muotoon vastausta varten. Datan hakemisen tehtävä voitaisiin siirtää oman ns. datakerrokseen (Data Access Layer), joka tarjoaisi funktiot tarvittavien tietojen hakemiseen.

Kontrollereiden käyttämiin useasti toistuviin datan hakutehtäviin on toteutettu apufunktioita, mutta ne sijaitsevat niitä käyttävien kontrollerifunktioiden kanssa samassa moduulissa. Jos jokin muu kontrollerimoduuli sattuu tarvitsemaan tällaista apufunktiota, joutuu se tuomaan funktion osaksi omaa moduuliaan import-lauseella. Tämä lisää projektin arkkitehtuurin sekavuutta eikä ole isommassa projektissa kestävä ratkaisu.

Nykyisessä muodossa esimerkiksi yksikkötestien (unit test) toteuttaminen olisi erittäin vaikeaa, ellei mahdotonta. Toteutettavat testit olisivat väistämättä integraatiotestejä, ja niiden suorittaminen veisi kauemmin kuin tarpeellista, kun yksittäisen kontrollerin testaaminen aiheuttaisi yhden tai useampia tietokantakyselyjä. Lisäksi testejä varten pitäisi olla jokin vakaa tietokannan alkutila ja mahdollisesti myös oma tietokanta testausta varten.

Testausautomaation puuttuminen itsessään on puute, varsinkin kun sovellus on toteutettu JavaScriptillä, jonka dynaamisen tyypityksen vuoksi kääntäjällä ei ole mahdollisuutta ilmoittaa ilmiselvistä tyyppivirheistä. Dynaamista tyypitystä käytettäessä tällaiset virheet ilmenevät ajon aikaisina virheinä, sen sijaan että niihin voitaisiin puuttua jo käännösvaiheessa.

6 FRONT-ENDIN TOTEUTUS

6.1 Arkkitehtuuri

Harjoitteluportaalin käyttöliittymä on itsenäinen selaimessa suoritettava JavaScript-sovellus. Se keskustelee palvelinsovelluksen kanssa käyttämällä asynkronisia HTTP-kutsuja. Käyttäjän selaimeen sovellus lähetetään palvelinsovelluksen toimesta normaalin HTML-sivunlatauksen yhteydessä, kun käyttäjä lataa selaimella harjoitteluportaalin etusivun.

Front-end on ohjelmoitu React-käyttöliittymäkirjastolla. React-sovellukset koostuvat komponenteista. Kulloinkin näytettävä www-sivu on yksi iso komponentti, joka rakentuu pienemmistä komponenteista. Se hakee näkymän tarvitseman datan palvelimelta ja välittää sen omille lapsikomponenteilleen tarpeen mukaan.

6.1.1 Datan hakeminen ja lähettäminen

React ei tarjoa palveluita HTTP-pyyntöjen tekemiseen, vaan ohjelmoija itse päättää, millä menetelmällä pyynnöt tehdään. Harjoitteluportaali tekee HTTP-pyynnöt Axios-kirjastoa käyttämällä.

Datan hakemiseen ja lähettämiseen tarvittavat funktiot on määritelty sovelluksen api-moduulissa funktioina, jotka tekevät varsinaiset API-kutsut Axios-kirjastolla. Tällä tavalla usein käytetyt rajapinnan osoitteet voidaan määritellä yhdessä paikassa, eikä niitä tarvitse parametreineen aina luoda merkkijonona, kun rajapintakutsuja tehdään. Moduuli ei siis ole mitenkään React-sidonnainen, vaan se on määritelty normaalina JavaScript-moduulina helpottamaan ohjelmointia. Api-moduulin funktioita kutsutaan tyypillisesti komponenttien componentDidMount-metodissa (ks. 3.10.2) sekä tapahtumakäsittelijöissä (esim. tallenna-painikkeen painaminen). (KUVIO 13.)

```
componentDidMount() {
  api.getUserPacks(getUser().userId)
    .then(res => {
      this.setState({
        packs: res.data.map(pack => ({ ...pack, expanded: false })),
        error: null,
      });
    })
    .catch(error => this.setState({ error }))
    .then(() => this.setState({ loading: false }));
}
```

KUVIO 13. Opiskelijan harjoittelupakettien haku componenDidMount-metodissa

6.1.2 Autentikaatio

Sovelluksen alussa Axios initialisoidaan kutsumalla api-moduulin funktiota initApi. Kutsun jälkeen autentikaatiotiketti – jos sellainen on olemassa – liitetään jokaisen pyynnön otsikkotietoihin automaattisesti, joten käyttöliittymän ohjelmoijan ei tarvitse huolehtia sen käsittelystä. Tämä tehdään käyttämällä Axiosin request interceptor-ominaisuutta, jonka avulla tehtyjä pyyntöjä voidaan muokata vielä ennen kuin Axios lähettää ne haluttuun osoitteeseen. Jos rajapintakutsu palauttaa HTTP-koodin 401, joka seuraa autentikaation vaativista kutsuista ilman voimassa olevaa autentikaatiota, kirjataan käyttäjä ulos sovelluksesta poistamalla vanhentunut tai virheellinen autentikaatiotiketti selaimen localStorage-muistista. Tämän jälkeen selain ohjataan automaattisesti sovelluksen etusivulle, joka näyttää käyttäjälle kirjautumislomakkeen. Tämä puolestaan tapahtuu response interceptor -osassa, joka suoritetaan aina kun HTTP-kutsuun saadaan vastaus. (KUVIO 14.)

```
export function initApi() {
  // Send auth token with each request
  axios.interceptors.request.use(config => {
    const token = localStorage.getItem('token');
    if (token)
      config.headers.Authorization = `Bearer ${token}`;
    return config;
  });

  // If no valid authentication is present, remove old auth token (if any) and
  // redirect to the start page.
  axios.interceptors.response.use(res => res,
    error => {
      if (error.response.status === 401) {
        auth.logout();
        browserHistory.push('/');
      }
      return Promise.reject(error);
  });
}
```

KUVIO 14. Autentikaation hallinta Axios-kirjaston interceptor-metodeilla

6.2 Reititys

Kulloinkin näkyvissä oleva komponentti valitaan Reactin router-komponentin toimesta. Sille kerrotaan, missä URL-osoitteessa näytetään mikäkin komponentti. Router on sovelluksen juurikomponentti, jonka vastuulla on näyttää oikeat komponentit selaimen osoiterivillä näkyvän osoitteen perusteella. Tunnistamattomat osoitteet saadaan käsiteltyä ”*”-säännön avulla. (KUVIO 15.)

```
ReactDOM.render(
  <Router history={browserHistory}>
    <Route path="/"                          component={App}>
      <IndexRoute                            component={FrontPage} />
      <Route path="/plans/:id"               component={PlanView} />
      <Route path="/packs/:id/report"        component={ReportView} />
      <Route path="/settings/front-page"     component={ComponentRoleGuard(FrontPageContentEdit, [roles.admin])} />
      <Route path="/settings/groups"         component={ComponentRoleGuard(GroupsEdit, [roles.admin])} />
      <Route path="/settings/lang-teachers" component={ComponentRoleGuard(LangTeachersEdit, [roles.admin])} />
      <Route path="/settings"                component={ComponentRoleGuard(Settings, [roles.admin])} />
      <Route path="/user-notes/:userId"      component={ComponentRoleGuard(UserNotes, [roles.admin])} />
      <Route path="/users/:userId"           component={ComponentRoleGuard(UserEdit, [roles.admin])} />
      <Route path="/email"                   component={ComponentRoleGuard(SendEmail, [roles.admin, roles.teacher, roles.langTeacher])} />
      <Route path="/student-search"          component={onentRoleGuard(StudentSearch, [roles.admin])} />
      <Route path="/teacher-search"          component={ComponentRoleGuard(TeacherSearch, [roles.admin])} />
      <Route path="*" component={NotFound} />
    </Route>
  </Router>,
  document.getElementById('app')
);
```

KUVIO 15. Sovelluksen reititys router-komponentilla

6.3 Komponenttien hierarkia

Sovelluksen komponentit on toteutettu käyttämällä jaottelua, jossa käyttöliittymän sovelluslogiikka erotetaan niistä osista, joiden tehtävänä on puhtaasti näkymän esittäminen ruudulla. Sovelluksen logiikan määrittelevät komponentit, joista käytetään koodissa nimeä **container,** hakevat palvelimelta tarvittavan datan ja välittävät sen näkymäkomponentille. Näkymäkomponentteja kutsutaan koodissa nimellä **component**. Kutsuttakoon näitä nimillä **älykomponentti** ja **näkymäkomponentti**. Tarvittavan datan lisäksi älykomponentti välittää näkymäkomponentille viittaukset tapahtumakäsittelijöihin, jolloin se voi vastata käyttäjän suorittamiin toimintoihin, kuten vaikkapa jonkin painikkeen painamiseen. Jokaisen näkymän juurikomponenttina on käytännössä älykomponentti, jonka render()-metodissa piirretään näkymäkomponentteina näkymän käyttöliittymän osat.

Erottelun ideana on erotella käyttöliittymän logiikan toteutus sen ruudulla esitettävistä osista. Näin yksittäisen näkymäkomponentti ei ole sidottu yksittäiseen datan lähteeseen, vaan sitä voidaan käyttää useiden eri lähteistä peräisin olevan datan kanssa.

6.4 Sovelluksen tilan ja datan hallinta

Harjoitteluportaalin front-end ei (juurikaan) käytä sovellukselle yhteistä globaalia tilaa, eikä se toteuta esim. Flux-arkkitehtuuria. Sen sijaan kukin älykomponentti pitää yllä omaa tilaansa ja pyytää palvelinsovelluksen rajapinnalta tarvitsemansa tiedot, jotka se sitten välittää näkymäkomponentille. Yksittäisen käyttäjäsession yli säilyvät tiedot, joiden pitää olla sovelluksen saatavilla globaalisti, säilytetään selaimen localStorage-muistissa. Käytännössä tällaisia tietoja ovat kirjautuneen käyttäjän tiedot sekä autentikaatiotiketti.

6.5 Käyttäjäroolien huomioon ottaminen

Sovelluksen käyttäjälle tarjoamat näkymät riippuvat paljon käyttäjän roolista. Lähtökohtana on, että käyttäjä ei pääse sellaiseen näkymään, jota hänen ei tarvitse ja jota hän ei saa nähdä. Sellaiset komponentit, jotka ovat sallittuja vain tietylle roolille tai rooleille, suojataan sovelluksen alussa luomalla niistä versio, joka rajoittaa niiden näyttämisen vain sallituille rooleille.

Jos käyttäjä arvaisi, mistä URL-osoitteesta jokin suojattu näkymä löytyy, voisi hän käsin muuttaa localStorageen tallennetun käyttäjän roolin sellaiseksi, että hän saa halutun näkymän esiin. Vaikka pahantahtoinen käyttäjä saisi näkymän esiin, ei palvelinsovellus

kuitenkaan anna käyttäjälle sellaista dataa, johon hänellä ei ole oikeutta päästä käsiksi, koska autorisointi suoritetaan autentikaatiotiketin perusteella. Vahinkoa ei näin pääse tapahtumaan ja siksi roolin tarkistaminen validoimalla autentikaatiotiketti käyttöliittymäsovelluksessa ei ole tarkoituksenmukaista lisätyötä.

Näkymien suojaus on sovelluksessa toteutettu luomalla funktio, joka muuttaa sille annetun komponentin sellaiseksi, että vain halutut käyttäjäroolit saavat sen näkyviin. Funktion parametrit ovat suojattava komponentti sekä roolit, jotka saavat nähdä kyseisen komponentin ja se palauttaa komponentin, joka näyttää yleisen NotFound-näkymän, jos käyttäjällä ei ole pääsyä kyseiseen näkymään. Muussa tapauksessa se palauttaa halutun komponentin. (KUVIO 16.)

```
import React from 'react';
import NotFound from '../components/not-found';
import { getUserRole } from '../auth';

export default (Component, allowedRoles) => {
  const GuardedComponent = props => {
    if (!allowedRoles.some(role => role === getUserRole()))
      return <NotFound />;

    return <Component {...props} />;
  };
  return GuardedComponent;
};
```

KUVIO 16. Komponentin suojaava funktio

Kun jokin komponentti halutaan suojata, se tuodaan haluttuun moduuliin normaalisti import-lauseella. Sen jälkeen siitä luodaan edellä kuvatulla funktiolla suojattu komponentti, joka asetetaan URL-reittien määrittelyssä haluttuun osoitteeseen. (KUVIO 17.)

```
import ComponentRoleGuard from './components/component-role-guard';

                                    ⇩

<Route path="/settings/front-page"    component={ComponentRoleGuard(FrontPageContentEdit, [roles.admin])} />
<Route path="/settings/groups"        component={ComponentRoleGuard(GroupsEdit, [roles.admin])} />
<Route path="/settings/lang-teachers" component={ComponentRoleGuard(LangTeachersEdit, [roles.admin])} />
<Route path="/settings"               component={ComponentRoleGuard(Settings, [roles.admin])} />
```

KUVIO 17. Komponenttien suojaus ComponenRoleGuard-funktiolla

7 KÄYTTÖLIITTYMÄN TOIMINNOT

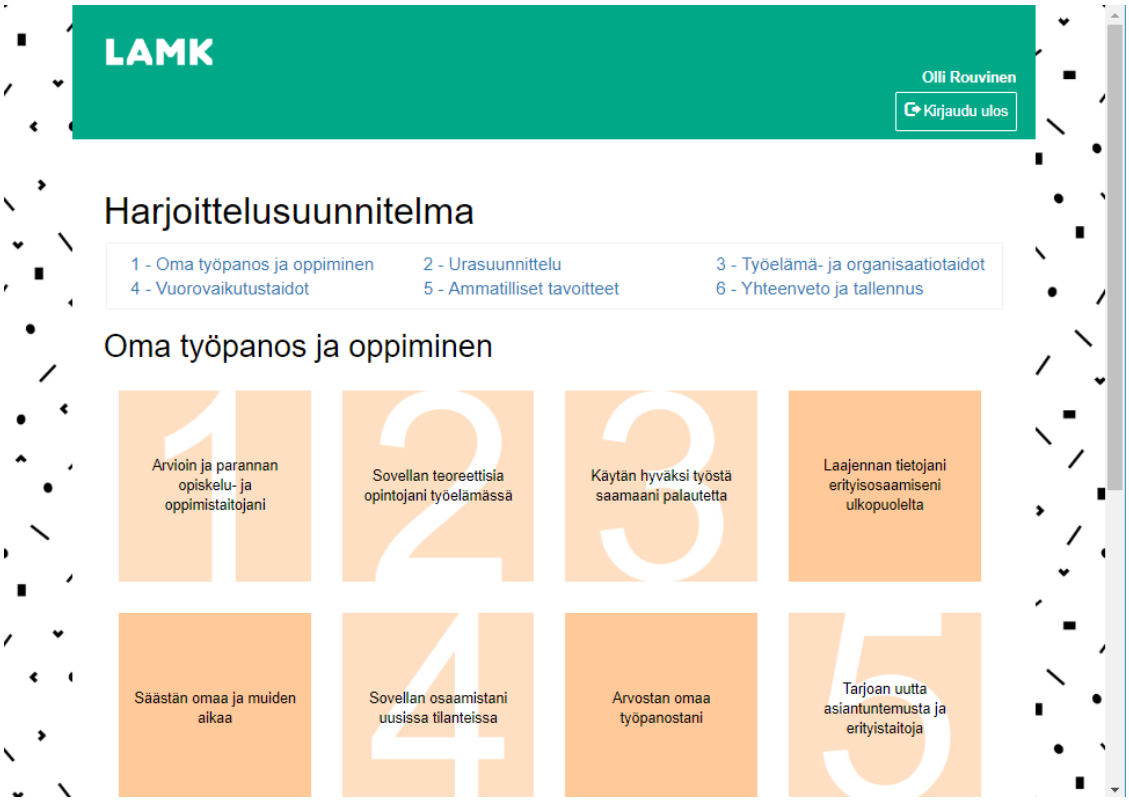
7.1 Opiskelija: harjoittelusuunnitelman luominen

Suunnitelman luominen alkaa harjoittelun perustietojen täyttämisellä. Kun perustiedot on täytetty, näytetään käyttäjälle harjoittelusuunnitelman muokkauksen näkymä. Näkymässä harjoittelusuunnitelma voidaan hylätä, tallentaa keskeneräisenä tai lähettää valmis suunnitelma hyväksyttäväksi.

Suunnitelma luodaan asettamalla harjoittelujaksolle oppimistavoitteet. Ammatilliset tavoitteet ovat opiskelijan vapaasti muotoilemia tavoitteita, joita opiskelijan tulee antaa vähintään kolme kappaletta. Muut tavoitteet sijoittuvat neljään eri kategoriaan:

1. Oma työpanos ja oppiminen
2. Urasuunnittelu
3. Työelämä- ja organisaatiotaidot
4. Vuorovaikutustaidot.

Näistä kategorioista valitaan kustakin viisi itselleen parhaiten sopivaa tavoitetta. Valittavissa olevat tavoitteet visualisoidaan kortteina, joita voidaan valita hiiren painikkeella. (KUVIO 18.)



KUVIO 18. Harjoittelusuunnitelman tavoitteet korttipelinä

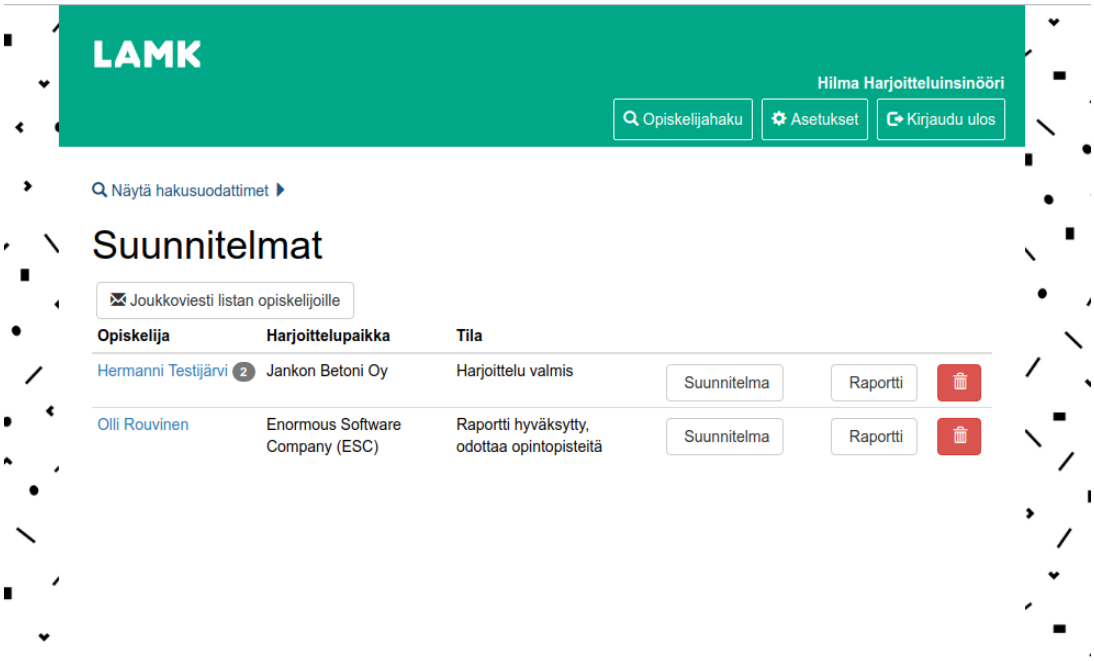
7.2 Opiskelija: liitetiedostojen palautus

Harjoittelun suorituksen edellytyksenä on harjoitteluraportin sekä -todistuksen palauttaminen. Kun harjoittelu on ohi, opiskelija voi avata raporttinäkymän, josta liitteet palautetaan. Jos raportissa on korjattavaa, opiskelija saa ilmoituksen korjauspyynnöstä sähköpostiin. Korjauksia pyytäneen opettajan tai harjoitteluinsinöörin palaute näkyy opiskelijalle raporttinäkymässä ja hän voi palauttaa korjatun raportin.

7.3 Harjoitteluinsinööri: etusivu

Harjoitteluinsinöörin päänäkymässä näytetään listaus niistä harjoitteluista, joiden suunnitelma tai liitetiedostot vaativat harjoitteluinsinöörin hyväksymistä. Harjoittelut listataan oletuksena aikajärjestyksessä vanhin ensin, jolloin työjonoa on helppo purkaa siinä järjestyksessä, jossa uusia suunnitelmia ja liitetiedostoja järjestelmään tallennetaan. Hakusuodattimilla listan harjoittelut voidaan rajata haluttujen hakuehtojen mukaan (esim. suunnitelman tila, opiskelijan nimi, jne.) tai järjestää haluttuun järjestykseen. Harjoitteluinsinööri voi myös lähettää joukkosähköpostin kaikille listalla näkyville opiskelijoille. Jos opiskelijaa

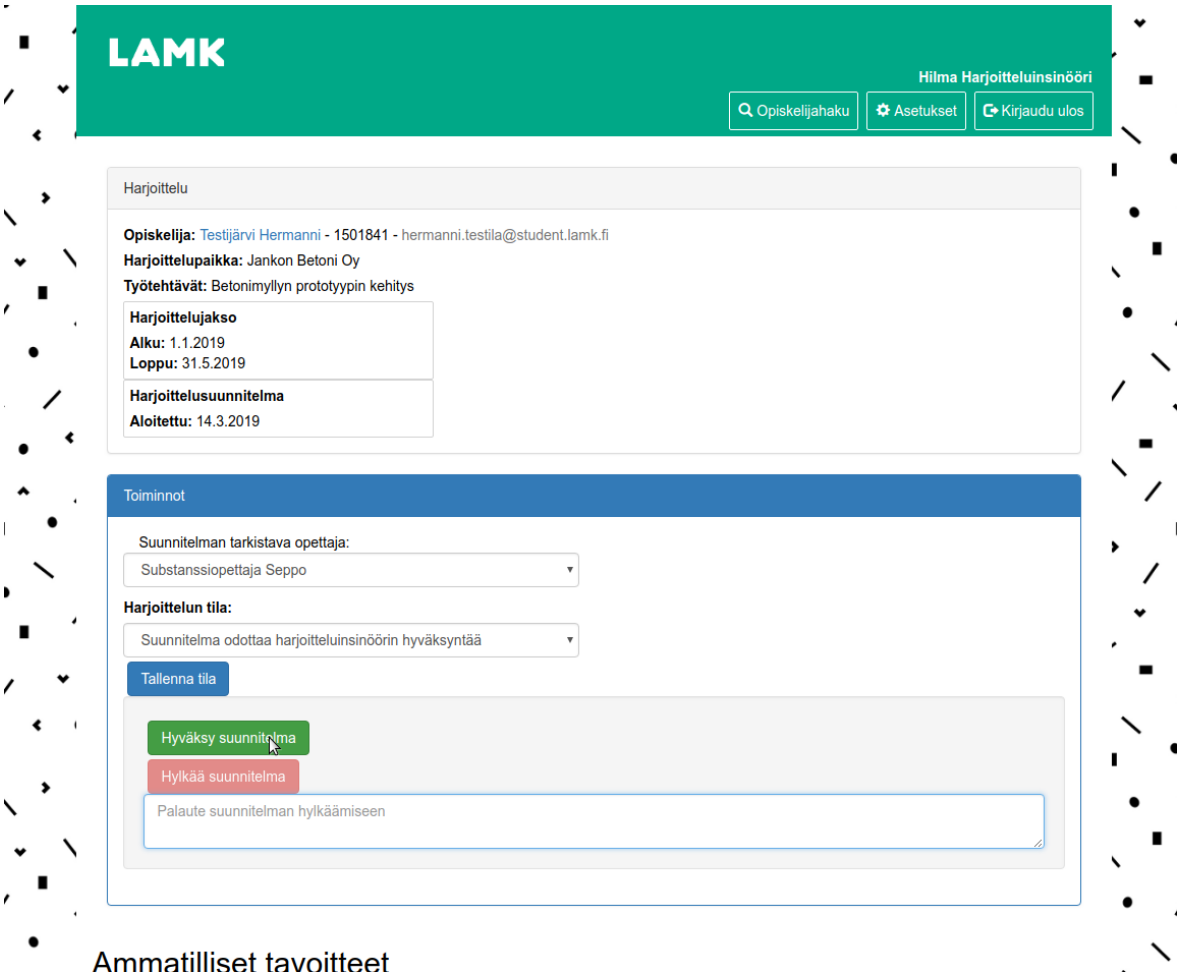
koskien on tehty järjestelmään muistiinpanoja (ks. 7.8), näkyy muistiinpanojen määrä opiskelijan nimen vieressä pienenä numerona, jota painamalla muistiinpanot saa näkyviin. (KUVIO 19.)



KUVIO 19. Harjoitteluinsinöörin päänäkymä

7.4 Harjoitteluinsinööri: harjoittelusuunnitelman tarkistus

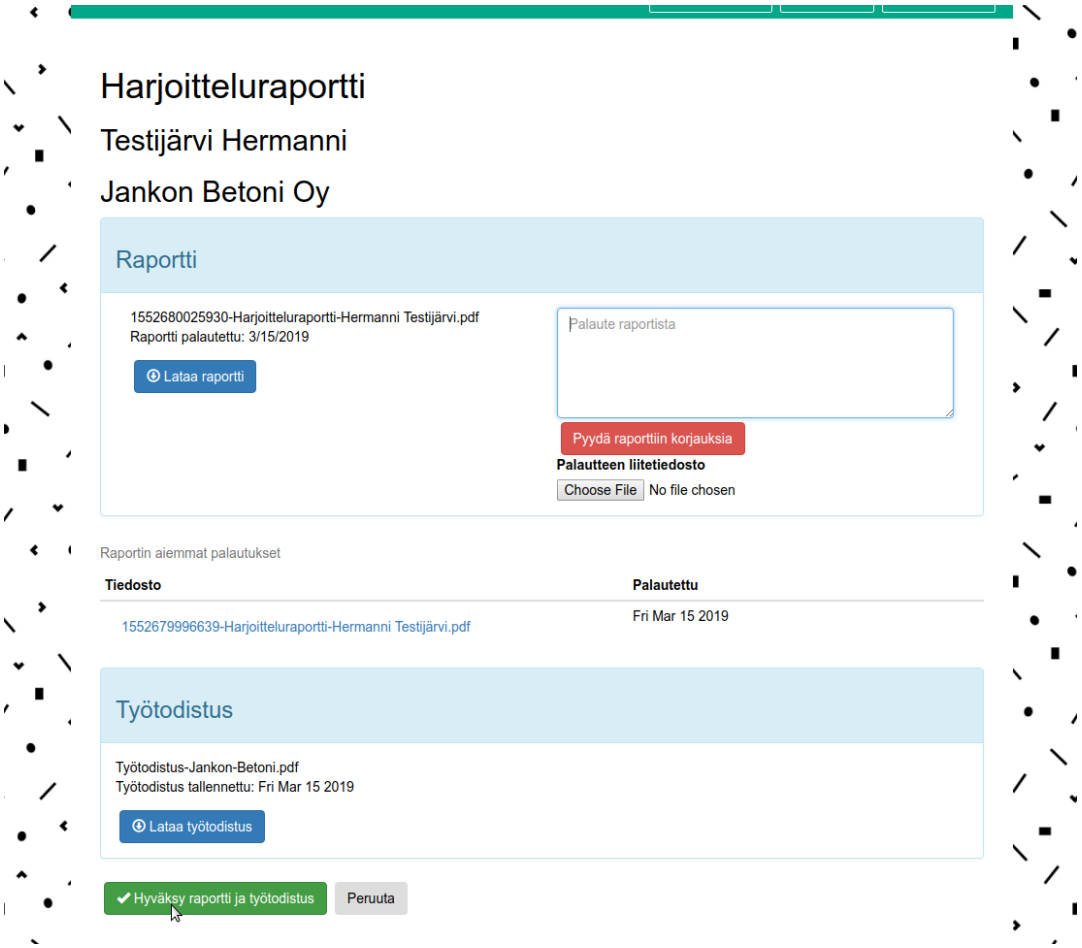
Kun harjoitteluinsinööri avaa suunnitelman, hän näkee yhteenvedon harjoittelun tavoitteista ja ajankohdan, koska suunnitelman teko on aloitettu sekä koska se on lähetetty hyväksyttäväksi. Hän voi hyväksyä suunnitelman tai pyytää siihen korjauksia. Korjauspyynnöstä lähtee sähköposti-ilmoitus opiskelijalle ja harjoittelun tilaksi asetetaan ”odottaa korjauksia suunnitelmaan”. Suunnitelmanäkymässä harjoitteluinsinööri voi myös suoraan muokata harjoittelun tilaa haluamakseen ja vaihtaa substanssiopettajan, jonka työjonoon suunnitelma siirtyy hyväksyttäväksi. (KUVIO 20.)



KUVIO 20. Harjoittelusuunnitelman tarkistus

7.5 Harjoitteluinsinööri: harjoitteluraportin ja työtodistuksen tarkistus

Liitetiedostojen hallintanäkymässä voidaan ladata liitetiedostot ja antaa raportista palautetta, mikäli se hylätään. Raporttien palautushistoriaa voidaan myös tarkastella siinä tapauksessa, että raportti on jouduttu palauttamaan useammin kuin kerran. (KUVIO 21.)

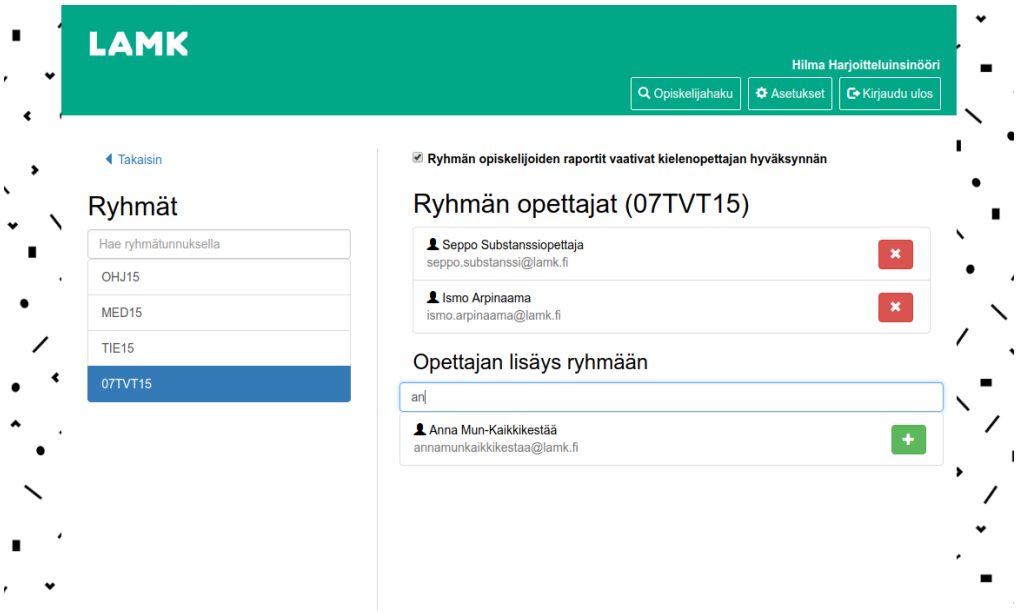


KUVIO 21. Harjoitteluraportin ja työtodistuksen hallinta

7.6 Harjoitteluinsinööri: asetukset

Opettajien lisäys järjestelmään mahdollistaa opettajien lisäämisen ennen kuin nämä ovat kirjautuneet harjoitteluportaaliin ensimmäisen kerran. Lisäyksen jälkeen opettaja on lisättävissä ryhmän substanssiopettajaksi ja/tai kielenopettajaksi. Normaalisti opettaja lisätään harjoitteluportaalin tietokantaan automaattisesti ensimmäisen kirjautumisen yhteydessä AD:sta saatujen tietojen perusteella.

Ryhmien asetuksista harjoitteluinsinööri voi valita ryhmän substanssiopettajat, sekä vaaditaanko ryhmän opiskelijoiden harjoitteluraporteille kielenopettajan hyväksyntä (KUVIO 22). Lisäksi harjoitteluinsinööri voi asettaa kielenopettajat sekä muokata harjoitteluportaalin etusivulla näkyvän sisällön (otsikko + teksti).



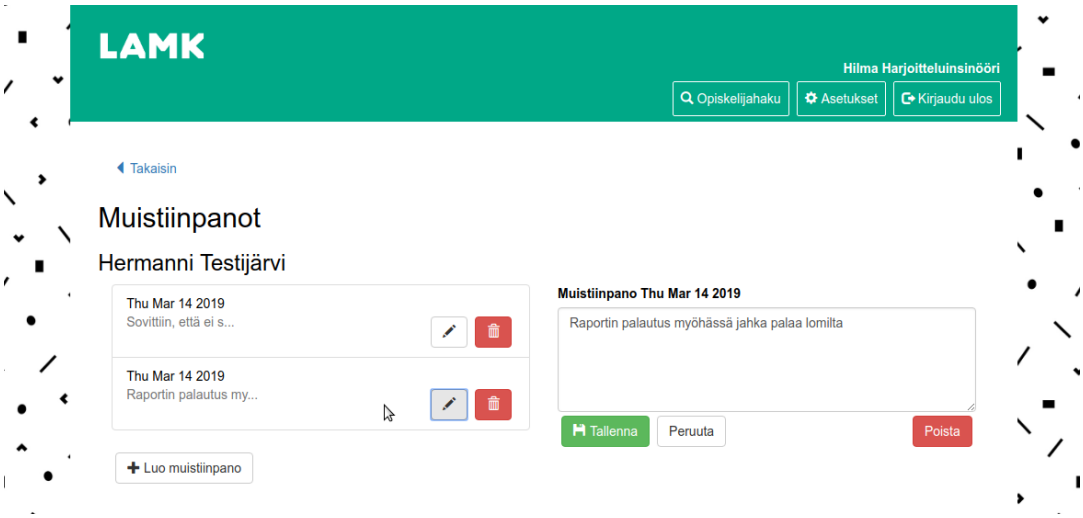
KUVIO 22. Ryhmän asetukset

7.7 Harjoitteluinsinööri: opiskelijahaku

Hakutoiminnon avulla harjoitteluinsinööri voi hakea opiskelijoita nimen, sähköpostiosoitteen, opiskelijanumeron tai opiskelijan tilan perusteella. Löydetyille opiskelijoille voi lähettää joko yksittäin tai joukkona sähköpostia. Opiskelijoiden tietoja pääsee tarkastelemaan erillisessä näkymässä, josta voi myös luoda muistiinpanoja opiskelijaa koskien (esim. muistutukseksi jostakin sovitusta asiasta harjoitteluprosessiin liittyen, ks. 7.8).

7.8 Harjoitteluinsinööri: muistiinpanot

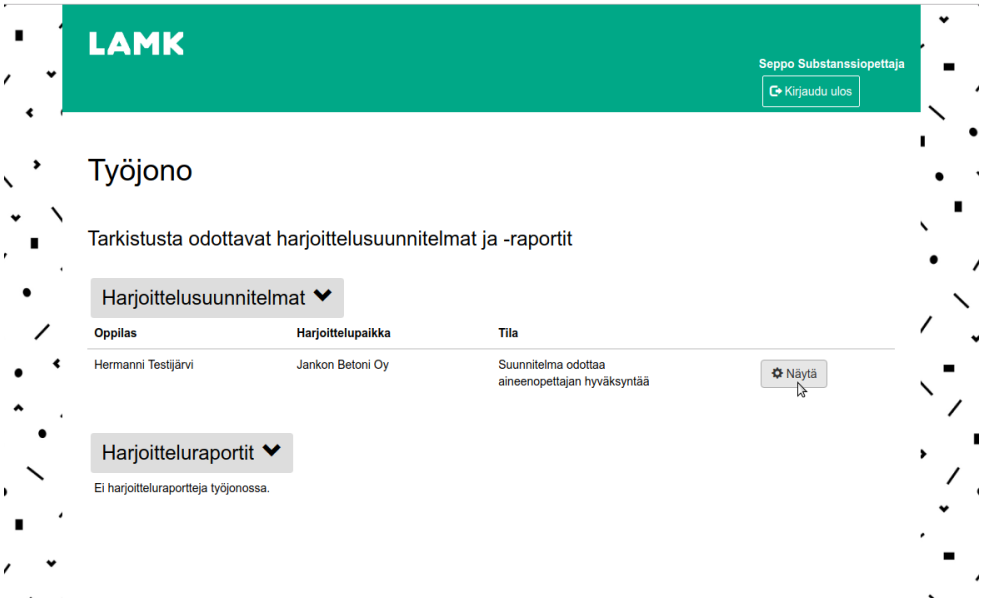
Harjoitteluinsinööri voi luoda muistiinpanoja opiskelijaa koskien. Kustakin muistiinpanosta näytetään sen luontipäivämäärä sekä varsinainen tekstisisältö. Muistiinpanojen tekomahdollisuus haluttiin, koska opiskelijan harjoitteluprosessista saatetaan välillä sopia opiskelijan kanssa jotain sellaista, jota harjoitteluportaalin tiedoista ei suoraan saa selville. (KUVIO 23.)



KUVIO 23. Opiskelijaa koskevat muistiinpanot

7.9 Substanssiopettajan toiminnot

Opettajalla on kaksi työjonoa: hyväksymistä odottavat harjoittelusuunnitelmat ja hyväksymistä odottavat raportit (KUVIO 24). Nämä näkyvät opettajan aloitussivulla omina listoinaan. Suunnitelmien ja raporttien hyväksymisnäkymät ovat samat kuin harjoitteluinsinöörillä. Ainoana erona on, että opettaja ei pääse suoraan muokkaamaan suunnitelman tilaa suunnitelman näkymästä. Kun opettaja hyväksyy suunnitelman, on harjoittelusuunnitelma hyväksytty ja opiskelija voi aloittaa harjoittelun.

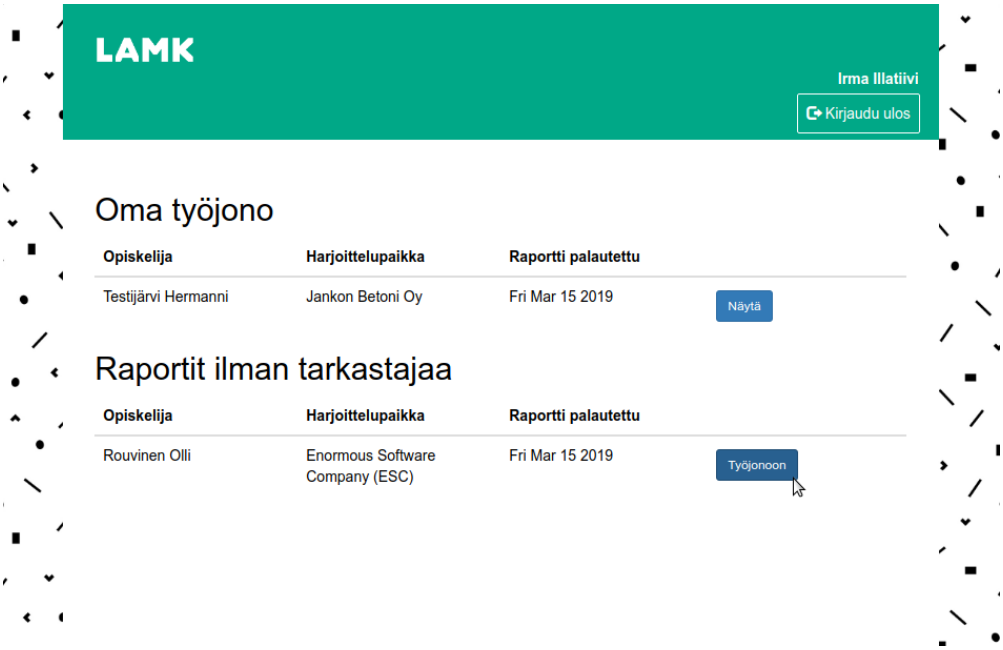


KUVIO 24. Substanssiopettajan työjonot

7.10 Kielenopettajan toiminnot

Kielenopettajan tehtäviin kuuluu ainoastaan harjoitteluraporttien kieliasun tarkistus. Kielenopettajia ei yhdistetä suoraan minkään opiskelijaryhmään, vaan päänäkymässä näkyvät kaikki järjestelmään tallennetut raportit, jotka odottavat kielenopettajan tarkistusta. Näkymässä kielenopettaja voi siirtää omaan työjonoon sellaisia raportteja, joita muut kielenopettajat eivät ole vielä siirtäneet omaan työjonoonsa (KUVIO 25).

Omasta työjonosta kielenopettaja voi siirtyä haluamansa raportin näkymään, josta hän voi ladata raporttitiedoston ja hyväksyä raportin, tai pyytää raporttiin korjauksia.



KUVIO 25. Kielenopettajan päänäkymä

Näkymä, josta kielenopettaja lataa opiskelijan tallentaman raportin ja hyväksyy tai hylkää sen, on sama kuin harjoitteluinsinöörillä ja substanssiopettajalla (ks. 7.5). Ainoana erona on se, että kielenopettajan näkymässä ei ole työtodistuksen hallintaan liittyviä toimintoja, sillä kielenopettajan vastuulla on ainoastaan raportin oikeakielisyyden tarkistus.

8 YHTEENVETO

Työn tavoitteena oli toteuttaa järjestelmä, jonka avulla LAMK:n tekniikan laitoksen oppilaiden työharjoitteluprosessia hallitaan. Järjestelmä toteutettiin REST-tyyppisellä hajautetulla arkkitehtuurilla. REST on arkkitehtuurina modulaarinen: datan käsittelyyn määritellään rajapinta, jonka eri puolilla olevien järjestelmän osapuolien ei tarvitse tietää toistensa toteutuksesta mitään – molemmat noudattavat ainoastaan yhteistä rajapinnan määrittelemää sopimusta.

Työn aikana paneuduttiin erityisesti autentikaation toteutukseen hajautetussa arkkitehtuurissa sekä pyyntöjen autorisointiin niin käyttäjäroolien kuin yksittäisen käyttäjän perusteella. Roolien perusteella autorisointi onnistuttiin siirtämään osaksi Express-sovelluskehyksen deklaratiivista API-reittien määrittelyä. Käyttäjän perusteella autorisointi vaatii nykyisessä toteutuksessa apufunktioiden toteutusta resurssien kontrollerimoduuleihin, mutta ratkaisu on vähintäänkin tyydyttävä. Alun perin ad-hoc-tyyppisestä jokaisessa kontrollerissa tehtävästä tarkistelusta päästiin nykyiseen ratkaisuun, joka on huomattavasti mukavampi ohjelmoijan kannalta ja vähemmän altis virheille, kun autorisointiin liittyvä koodi on vain kertaalleen kirjoitettu.

Työtä tehdessä huomattiin, että jatkokehityksenä palvelinsovelluksen arkkitehtuuriin kannattaisi lisätä erillinen datakerros kontrollerilogiikan ja tietokannan väliin. Näin saavutettaisiin parempi testattavuus sekä selkeämpi projektin moduulien rakenne. Ohjelmointikielen vaihtaminen esim. TypeScriptiin olisi hyödyllistä, sillä kehityksen aikana olisi vältytty useilta ohjelmointivirheiltä, kun tyyppivirheet olisi huomattu jo käännösaikana. Toinen peruste TypeScriptiin siirtymiselle on, että taukojen jälkeen projektin pariin palaaminen olisi ollut helpompaa, kun käytössä olevien tietotyyppien määrittelyistä olisi voinut tarkistaa minkä tyyppistä dataa ollaan milloinkin käsittelemässä, eikä arvailuun ja datan käsin selvittelyyn olisi kulunut aikaa. Staattista tyypitystä käytettäessä myös kehitysympäristö tai editori osaa tavallisesti auttaa ohjelmoijaa tarjoamalla informaatiota tietotyyppien rakenteesta.

Valmiilla järjestelmällä voidaan hoitaa harjoitteluprosessi alusta loppuun, joskin sen todellinen testaus on jäänyt toistaiseksi vähäiseksi. Mahdolliset muutokset olisivat käytettävyyteen liittyviä ja niiden toteuttaminen ei luultavasti vaatisi suuria mullistuksia projektin koodin perusrakenteisiin.

LÄHTEET

Bray, Ed. T., Textuality 2006. The JavaScript Object Notation (JSON) Data Interchange Format [viitattu 8.3.2019]. Saatavissa: https://www.ietf.org/rfc/rfc4627.txt

Closures 2019. Mozilla [viitattu 8.3.2019]. Saatavissa: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Closures

Express 2019 [viitattu 8.3.2019]. Saatavissa: https://expressjs.com

Fielding, R., Adobe, Reschke, J., greenbytes 2014. Hypertext Transfer Protocol (HTTP/1.1): Authentication [viitattu: 8.3.2019]. Saatavissa: https://tools.ietf.org/html/rfc7235

Introduction to JSON Web Tokens 2019 [viitattu 20.3.2019]. Saatavissa: https://jwt.io/introduction/

Jones, M., Microsoft, Bradley, J., Ping Identity, Sakimura, N., NRI 2015. JSON Web Token (JWT) [viitattu 8.3.2019]. Saatavissa: https://tools.ietf.org/html/rfc7519

JSON 2019 [viitattu 13.3.2019]. Saatavissa: https://json.org

Node.js 2019 [viitattu 13.3.2019]. Saatavissa: https://en.wikipedia.org/wiki/Node.js

React 2019. Facebook Inc. [viitattu 8.3.2019]. Saatavissa: https://reactjs.org

React.Component 2019. Facebook Inc. [19.4.2019]. Saatavissa: https://reactjs.org/docs/react-component.html

Reconciliation 2019. Facebook Inc. [viitattu 13.3.2019]. Saatavissa: https://reactjs.org/docs/reconciliation.html

Representational State Transfer 2019 [viitattu 8.3.2019]. Saatavissa: https://en.wikipedia.org/wiki/Representational_state_transfer

Similä, N. 2019. VS: Harjoitteluportaali. Sähköpostiviesti. Vastaanottaja Rouvinen, O. Lähetetty 13.3.2019.

The Arguments Object 2019. Mozilla [viitattu 20.3.2019]. Saatavissa: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/arguments

Writing Middleware 2019 [viitattu 8.3.2019]. Saatavissa: https://expressjs.com/en/guide/writing-middleware.html