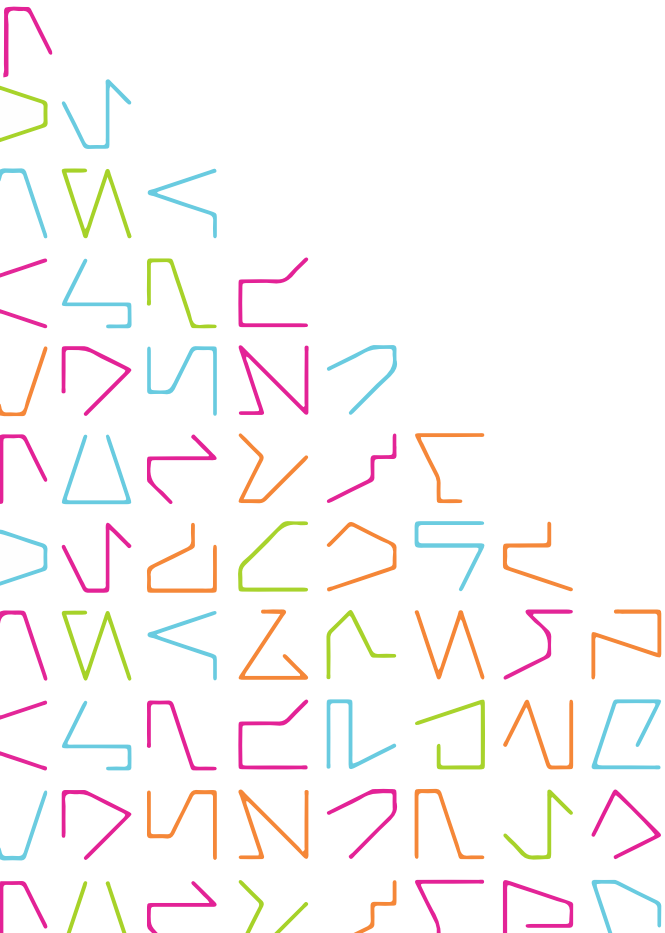
TAMPEREEN
AMMATTIKORKEAKOULU

HENKILÖTIETOJÄRJESTELMÄN KEHITYS LIFERAY-ALUSTALLE

Juuso Korhonen



Opinnäytetyö
Toukokuu 2018
Tietotekniikka
Ohjelmistotekniikka

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka

JUUSO KORHONEN:
Henkilötietojärjestelmän kehitys Liferay-alustalle
Opinnäytetyö 27 sivua, joista liitteitä 0 sivua
Toukokuu 2018

Opinnäytetyön tarkoitus on luoda asiakkaan vanhentuneen ja ulkoisella sivustolla sijaitsevan puhelinluettelojärjestelmän tilalle uusi henkilötietojärjestelmä asiakkaan omalle verkkosivulle.

Uuteen henkilötietojärjestelmään kuuluu yhteystietojen tallennus, tuonti ulkoisesta järjestelmästä asiakkaan omaan Liferay-ympäristöön sekä henkilötietojen näyttäminen kahdella erilaisella toteutuksella. Henkilötietojen näyttöön luotiin kaksi erilaista komponenttia: puhelinluettelo sekä räätälöityjen listojen luontiin ja näyttämiseen tarkoitettu komponentti. Molemmat komponentit luotiin Liferayn portlet-liitännäisinä.

Opinnäytetyön tarkoituksena ei ole kuitenkaan toimia ohjeena sille, miten Liferaylle luodaan portlet-liitännäisiä, vaan se on kuvaus asiakkaalle tehdystä työstä.

Työn toimeksiantaja on Visma Consulting Oy, jonka välityksellä työ tehtiin kolmannen osapuolen asiakkaalle. Asiakas pidetään anonyyminä opinnäytetyössä. Yhteistyötä asiakkaan ja Visma Consultingin välillä on ollut jo ennen opinnäytetyötä ja muun muassa Liferay-ympäristö, jolle portletit asennetaan, on Visma Consultingin toteuttama.

Asiasanat: Java, Spring, Liferay

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
ICT Engineering
Software Engineering

JUUSO KORHONEN:
Developing a contact information system for Liferay platform

Bachelor's thesis 27 pages, appendices 0 pages
May 2018

The purpose of this thesis is to create a new personnel information system into the client's existing website to replace their old telephone directory located on an external site.

The new personnel information system consists of components that handle saving the contact information of the personnel, importing them from the client's external system to their own Liferay-environment and presenting that information using two different implementations. The information is presented as a new telephone directory as well as a component that allows the creation and presentation of custom lists of contacts. Both components were implemented as Liferay portlet-plugins.

The purpose of this thesis is not to act as a guideline on how to create portlet-plugins for Liferay, but to simply showcase and go into detail on what was created for the client.

The components were created for Visma Consulting Oy, who is not the end client, but a consulting company from which the components were ordered from by a third-party client. The client will be kept anonymous in the thesis. Co-operation between the client and Visma Consulting has been happening even before this thesis and the Liferay-environment, among other things, was created for the client by Visma Consulting.

Key words: Java, Spring, Liferay, Portlet

SISÄLLYS

ERITYISSANASTO

CSV	Tiedostomuoto, jossa taulukkomuotoinen data esitetään puolipilkuilla ja pilkuilla eroteltuina
Frontend	Ohjelmiston osa, jonka tarkoitus on luoda käyttäjälle näytettävä näkymä.
Backend	Ohjelmiston osa, palvelinpään logiikka sijaitsee.
JavaBean	JavaBeanillä tarkoitetaan Javaluokkaa, jolla on constructor-metodi, getter- ja setter-metodit sekä sen tulee implementoida Javan Serializable-luokka
Moduuli	Moduuli on Mavenin puitteissa pienikokoinen projekti, joka suoritetaan muiden sille merkattujen sisarmoduulien kanssa yhtäaikaisesti
DTO	Data Transfer Object – käytetään tiedonsiirto-objektina tietokannan entiteetin ja käyttäjälle näytettävän tiedon välillä
CRUD-operaatiot	Create, read, update ja delete: neljä tietokantaentiteeteille tehtävää perusoperaatiota
JSTL	Kirjasto, josta löytyy erilaisia työkaluja työssä käytettävien JSP-sivujen logiikan avuksi
WAR	Web Application Archive: Pakattu web-sovellus, joka sisältää java- ja konfiguraatiotiedostoja
MVC	Malli, jossa toteutus jaetaan kolmeen osaan, model, view ja controller (malli, näkymä ja ohjain). Auttaa luomaan modulaarisempia ja uudelleenkäytettävämpiä sovelluksia

1 JOHDANTO

Projektissa tehtiin ulkoisessa Helpnet-järjestelmässä sijaitseville asiakkaan organisaation henkilötiedoille käsittelytyökalu ja näiden tietojen näyttämiseen kaksi erilaista toteutusta: puhelinluettelo ja räätälöidyt henkilölistat. Nämä ominaisuudet asennettiin asiakkaan jo olemassa olevalle Liferay-portaalialustalle, jolla asiakkaan organisaation kotisivut sijaitsevat. Ominaisuudet toteutettiin Liferayn Portlet-liitännäisinä.

Asiakkaan aikaisempi henkilötietojen näyttötoteutus sisälsi ainoastaan puhelinluettelon ja se sijaitsi kolmannen osapuolen sivustolla. Sen käyttäminen todettiin liian epäkäytännölliseksi sekä ajateltiin, että uusi, omalla sivustolla sijaitseva henkilötietojärjestelmä tekee käyttäjän käyttökokemusta sulavammaksi sekä se saadaan helpommin taivuteltua organisaation tyyliohjeiden mukaiseksi myös jatkossa. Lisäksi asiakas tarvitsi henkilötietojen näyttämiseen myös keinon luoda uusia henkilölistoja ja näyttää näitä oman tyyliohjeensa mukaisesti asianmukaisilla sivuilla.

Työ tehtiin Visma Consulting Oy:lle, joka puolestaan toimii konsulttiyrityksenä kolmannen osapuolen asiakkaalle. Kolmannen osapuolen asiakas pidetään opinnäytetyössä anonyyminä. Tässä dokumentissa puhutaan henkilölistoista, puhelinluettelosta sekä henkilötietojen käsittelytyökalusta yhteisnimellä ”henkilötietojärjestelmä”. Opinnäytetyöraportti on jaettu viiteen pääkokonaisuuteen. Raportin toisessa, kolmannessa ja neljännessä kappaleessa selitetään auki henkilötietojärjestelmän eri osat (käsittelytyökalu, puhelinluettelo sekä henkilöluettelo), viidennessä kappaleessa esitellään, minkälaiseen ympäristöön työ sijoitettiin sekä mitä työkaluja ja teknologioita siinä käytettiin ja lopuksi, kuudennessa kappaleessa, pohditaan työn jatkokehitystä sekä arvioidaan projektin onnistumista.

2 HENKILÖTIETOJEN KÄSITTELYTYÖKALU

Henkilötiedot tuodaan järjestelmään erilliseen Java-pakettiin luodulla käsittelytyökalulla, joka ottaa joka päivä määritellystä paikasta CSV-muotoisen tiedoston, parsii siitä tarvitsemansa datan ja tallentaa datan MySQL-tietokantaan. Lisäksi työkalulla hoidetaan kaikki CRUD-operaatiot henkilötiedoille. Tässä moduulissa ei ole frontendin koodia lainkaan, vaan kaikki toiminta tapahtuu serverin päässä. Henkilötietojen tuonnissa käyttäjän ainut tehtävä on tuoda moduulin tarvitsema CSV-tiedosto levylle. Mahdollisesti myös tämä vaihe tullaan automatisoimaan tulevaisuudessa.

Moduuli on jaettu niin, että siihen kuuluu erikseen API-rajapinta ja erillinen Java-luokka, joka on merkitty Springille ”repository” -luokaksi. API-rajapinta on tarkoitettu tietoja näyttävien moduulien käyttöön ja sitä kautta suoritetaan CRUD-operaatioita henkilötiedoille.

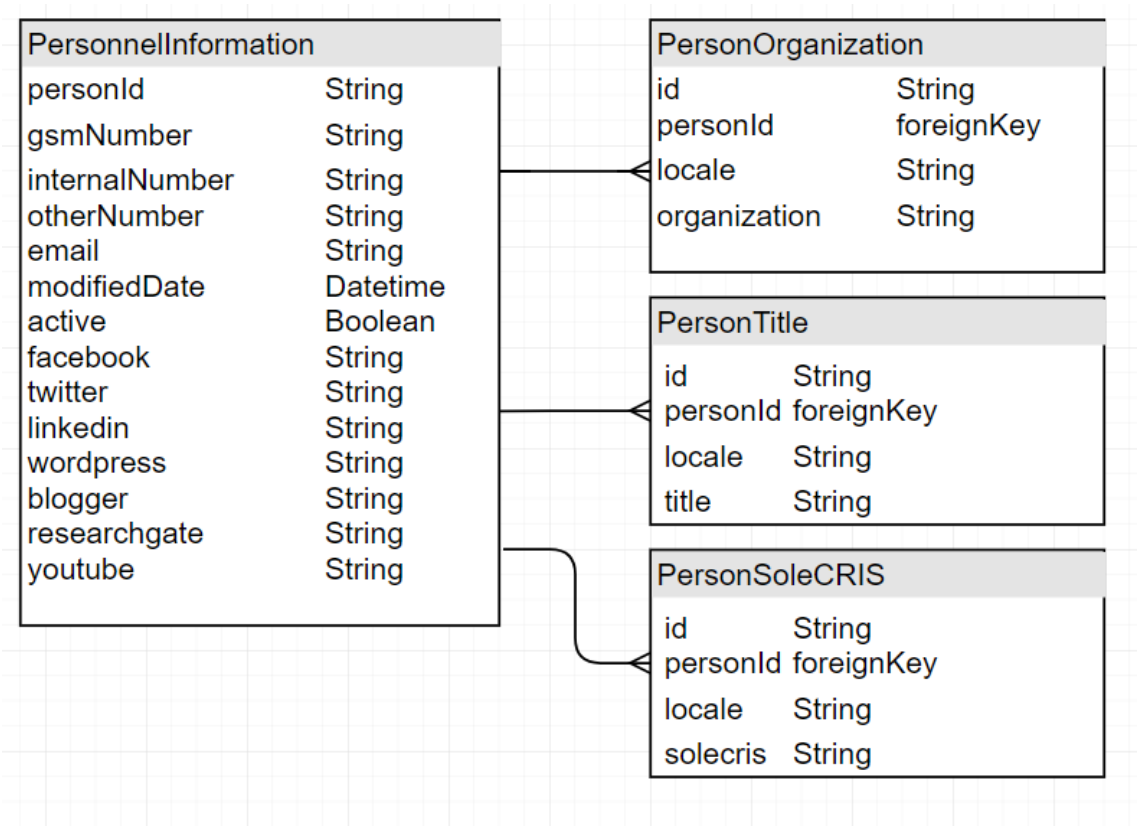
2.1 Henkilötiedon kuvaus

Osalla Helpnetistä tulevista henkilötiedoista on englanninkielinen käännös. Päätimme sijoittaa nämä käännettävät kentät erillisiin tietokantatauluihinsa niin, että niillä on primiary key, joka osoittaa henkilön id-arvoon, käännöksen lyhenne (fi tai en) sekä tallennettava arvo. Näitä käännettäviä kenttiä ovat henkilön ”organisaatiotieto” eli minkälaisen polun päässä ko. organisaatiossa henkilö on (esimerkkimerkkijono voisi olla esimerkiksi ”Suomen Käytettyjen autojen välitys Oy | Tampereen toimipiste | Myyntiosasto”), henkilön huonetieto (esimerkiksi ”A305”), henkilön nimike (esimerkiksi Liiketoiminnanjohtaja) sekä henkilön SoleCRIS-linkki.

Nämä tiedot tallennetaan erillisiin tietokantatauluihinsa ja niillä on many-to-one -suhde henkilötietoihin, eli yhdellä henkilöllä voi olla useamman kielen huonetieto, mutta yhdellä huonetiedolla voi olla vain yksi henkilö. Nämä tiedot päätettiin tallentaa omiin tauluihinsa kahden syystä: tietokanta oli jo valmiiksi eristetty omalle MySQL-käyttäjälleen, joten on varaa levittää tietoja omiin tauluihinsa sekä levittämällä nämä tiedot omiin tauluihinsa sen sijaan, että kaikki tiedot tallennettaisiin samaan tauluun, saadaan tietokanta pidettyä järkevänä siltä varalta, että tallennettavia, käännettäviä tietoja

tullaan tulevaisuudessa lisäämään. Mikäli kaikki tiedot olisivat samassa taulussa, käännettävät kentät tulisi joko tuoda kolumniinsa esimerkiksi JSON-datana, jossa olisi molemmat käännökset, tai sitten kokonaan omiin kolumneihinsa. Molemmat näistä tavoista vaikuttivat suunnitteluvaiheessa huonommilta vaihtoehdoilta toteutettuun verrattuna, joten valitsimme useamman taulun. (Kuva 1)

Henkilön omaan metadataan kuuluu henkilön nimen lisäksi puhelinnumeroiden tiedot (GSM, organisaation sisäinen numero, muu numero), sähköpostiosoite, aikamerkintä viimeisimmälle tallennukselle sekä tieto siitä, onko henkilötieto aktiivinen vai ei (enabled/disabled). Lisäksi käyttäjällä on tallennettuna linkkejä erinäisiin sosiaalisiin verkostoihin (Facebook, Twitter, LinkedIn, WordPress, Blogger, ResearchGate ja Youtube). Kaikki tämä data peilataan suoraan kyseessä olevan organisaation käyttämästä ulkoisesta käyttäjätietojärjestelmästä, joten käyttäjä saa oman datansa poistettua kannasta poistamalla sen ensin tästä ulkoisesta järjestelmästä, jonka jälkeen poistotapahtuma liikkuu seuraavan tuonnin aikana Liferaylle.



Kuva 1. ER-kaavio tallennetuista tietokantatauluista

2.2 Henkilötietojen tuonti

Datan serverille tuonti on toistaiseksi koko prosessin ainut manuaalinen osuus. Käyttäjä (ylläpitäjä) tuo CSV-muotoisen tiedoston serverille ennalta määriteltyyn paikkaan, josta työkalu tarkistaa sille konfiguroidun CRON-säännön (Unix-pohjaisten käyttöjärjestelmien ajastuspalvelu) mukaan. Tämän jälkeen tiedostosta haetaan henkilötiedot rivi kerrallaan, ne sidotaan niille luotuihin DTO:ihin ja tarkistetaan, onko käyttämäämme kantaan jäänyt henkilötietoja, jotka on jo ulkoisesta järjestelmästä poistettu ja päivitetään tietokanta.

CSV-tiedoston viikottaisen tarkistuksen suorittaa ImportPersonnelInformation-niminen luokka, jonka run-metodi on merkattu Springille ajastetuksi scheduled-annotaatiolla (kuva 7). Tätä annotaatiota ohjataan Unixista tutuilla cron-lauseilla, muttei sillä ole mitään muuta yhteistä Unixin cron-palvelun kanssa. Metodin voi myös ajaa tarvittaessa käsin. Kun metodi ajetaan, CSV-tiedostolta löytyneet rivit annetaan eteenpäin PersonnelImporterFacade luokalle, joka toimii API:na tälle henkilötietojen käsittelytyökalulle. Kaikki pyynnöt, joilla pyritään muokkaamaan/lukemaan henkilötietojen dataa, tehdään kyseessäolevalle luokalle. Lisäksi kuvassa 2 nähdään value-annotaatio, jolla voidaan hakea erillisestä Springiin konfiguroidusta property-tiedostosta arvo annotoidulle muuttujalle. Tällä annotaatiolla saadaan kyseessäolevalle luokalle csv-tiedoston sijainti polkuna sekä tiedoston nimi.

```
@Service("competenceEvaluationProcessor")
public class ImportPersonnelInformation {

    Logger log = Logger.getLogger(ImportPersonnelInformation.class.getName());

    @Autowired
    PersonnelImporterFacade personnelImporterFacade;

    @Value("${dumpfile.dir}")
    String dumpPath;

    @Value("${dumpfile.name}")
    String dumpName;

    @Scheduled(cron="${dumpfile.schedule}", zone="Europe/Istanbul")
    public void run() {
        String csvFile = "";
        if(dumpPath.startsWith("/")){
            csvFile = dumpPath+"/"+dumpName;
        }else{
            csvFile = PropsUtil.get("liferay.home") +"/"+dumpPath+"/"+dumpName;
        }

        try {
            personnelImporterFacade.importFile(csvFile);
        }
        catch(Exception e){
            e.printStackTrace();
        }


    }



}
```

Kuva 2. ImportPersonnelInformation-luokka

2.3 API-rajapinta

Henkilötietojen käsittelytyökalua käytetään pääosin sille rakennetun API-rajapinnan kautta. Ulkoiset Java-luokat hakevat työkalun JavaBeanin Springin avulla käyttöönsä ja kutsuvat rajapinnan Interface-luokan metodeja. Tällä luokalla ei ole lainkaan REST-rajapintaa, vaan se ottaa kutsuja vastaan ainoastaan Javalla. REST-rajapintaa emme tälle työkalulle nähneet tarpeelliseksi, vaan käytämme sitä mielummin riippuvuutena sekä puhelinluettelo- että henkilölistausportleteissa. Ongelma tässä ratkaisussa on se, että tämä

käsittelytyökalu on luotuna JavaBeanina siis kahteen kertaan. Ongelmallista tilanteessa on se, että aina kun käsittelytyökaluun tehdään päivityksiä, tulee molemmat portletit päivittää, jotta molemmat käyttävät uusinta versiota työkalusta. Tämä ei kuitenkaan ole niin merkittävä ongelma, että optimoinnille olisi ollut tarvetta.

Rajapinnalla on CRUD-operaatioille omat metodinsa ja näiden lisäksi sillä on private-merkattuja metodeja, joilla luokka muun muassa muuttaa tietokannasta tulevan datan DTO-objekteiksi, tuo CSV-tiedoston kiintolevyltä, tarkastaa tämän tiedoston oikeellisuuden (pituus yli 2000 merkkiä, puolipilkulla erotellut rivit) sekä erinäisiä helper-metodeja esimerkiksi tietokantaentiteetin alustukseen.

3 PUHELINLUETTELO

Puhelinluettelo-portletti on ensimmäinen kahdesta tavasta näyttää henkilötietoja. Portletti koostuu kahdesta sivusta: näyttösivusta ja konfiguraatiosivusta. Konfiguraatiosivun tarkoituksena on ainoastaan konfiguroida portletille asetettavia preferenssejä, joilla voidaan muuttaa näyttösivulla näytettävää henkilöjoukkoa ja sitä, mitä tietoja henkilöistä näytetään. Näyttösivulla taas näytetään sille tuodut henkilötiedot, sillä voidaan etsiä vapaasanahaulla konfiguraatiosivulla mahdollisesti asetettujen suodattimien sisällä ja Liferay-sivustolle merkityt ylläpitäjät voivat muuttaa henkilötietoja sekä luoda uusia henkilöitä.

3.1 Backend-toteutus

Puhelinluettelon oma backend on hyvin suppea – se toteuttaa portletin näyttämiseen ja konfigurointiin tarvittavat metodit ja toimii käsittelijänä puhelinluettelon ja edellämainitun henkilötietorajapinnan välillä. Backend on toteutettu MVC-mallia toteuttaen niin, että se on jaettu malli- (model), näkymä- (view) ja käsittelijäosiin (controller). Näistä malli- ja käsittelijäosat sijaitsevat backendissä. Käsittelijäosaan kuuluu kaksi käsittelijäluokkaa PhonebookController ja PhonebookEditController, joista ensimmäisen tehtävänä on hoitaa kaikki portletin normaalinäkymästä tapahtuvat kutsut, ja jälkimmäisen tehtävä on hoitaa kaikki portletin konfiguraationäkymästä tapahtuvat kutsut. Näiden lisäksi portletin kuuluu malliosan PhonebookService-luokka, jossa suurin osa backend-logiikasta sijaitsee.

PhonebookController-luokka sisältää Liferay-portletin tarvitseman render-metodin, jossa asetetaan näytettävät henkilöt malliin, joka viedään eteenpäin frontendille. Malliin asetetaan portletin konfiguraatioasetukset portletin preferensseistä (konfiguraationäkymästä asetettavia avain/arvo -pareja), konfiguraatiot erillisestä .property-tiedostosta ja alkunäkymässä näytettävät henkilöt. Lisäksi luokka sisältää metodit henkilöiden hakemista (ottaa vastaan merkkijonon hakua varten) sekä henkilön tallentamista, muokkaamista sekä poistamista varten. Portletilla itsellään voi muokata käyttäjien tietoja, mutta ne yliajetaan aina csv-tiedostosta löytyvillä tiedoilla, sillä ajatuksena on, että tämä tieto on aina uusin ja ajan tasalla. Portletilla voi myös lisätä

manuaalisesti käyttäjiä, jolloin näille asetetaan ”MAN”-alkuinen ID ja niitä ylläpidetään ainoastaan portletin kautta. Niitä ei myöskään poisteta CSV-tiedoston ajossa. PhonebookEditController-luokka sisältää portletin konfiguraationäkymälle tarpeelliset metodit, kuten createForm, jolla lomake luodaan sivua alustaessa, save, jolla näkymältä saatu lomake tallennetaan portletin preferenssihin, ja render, jolla tämä näkymä näytetään. Lisäksi luokalla on ModelAttribute-merkattu metodi, jolla luodaan näkymän käyttämä lomake ja populoidaan sille data portletin preferensseistä ennen sen näyttämistä.

Portletti on tehty niin, että mikäli sille ei ole konfiguroitu yhtään organisaatiota suodattimeksi konfiguraationäkymästä, ei alkunäkymässä näytetä yhtään henkilöä näiden suuren määrän (yli 1000) vuoksi. Mikäli taas on valittu joku suodatin, alkunäkymässä näytetään kaikki tämän organisaation jäsene.

3.2 Frontend-toteutus

Portletin frontend on rakennettu kolmella jsp-sivulla. Näihin kuuluvat edit, view ja listTemplate. Editillä ja viewillä näytetään portletin perusnäkymä ja konfiguraationäkymä ja listTemplate on yksittäisen henkilötiedon näyttämiselle tarkoitettu pohja.

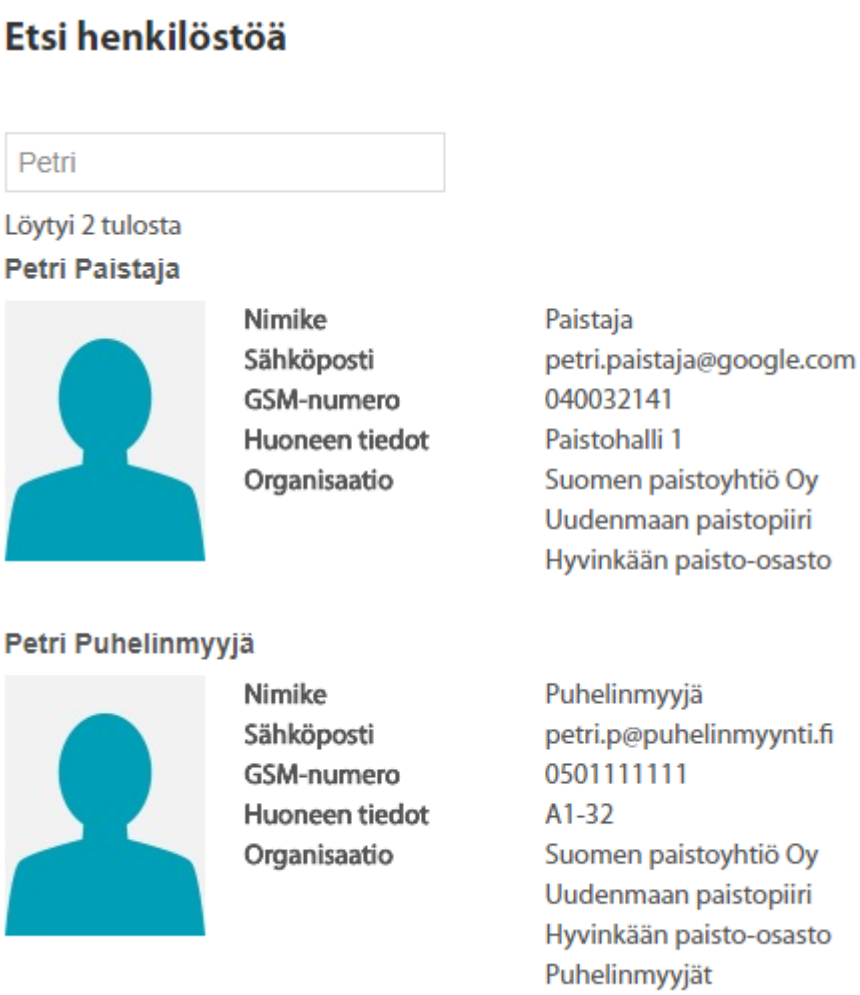
Edit-sivulla otetaan vastaan PhonebookEditControllerin lähettämä lomake ja näytetään se HTML:llä ja JSTL-logiikalla (kuva 3). Sillä ei ole muuta käyttötarkoitusta, kuin toimia näkymänä tälle lomakkeelle.

```
<div>
    Selected:
    <c:forEach var="field" items="${editForm.organizationsFilter}">
        <div>${field}</div>
    </c:forEach>
</div>
```

Kuva 3 JSTL-kirjaston logiikkaa

View-sivulla on hieman enemmän logiikkaa – sen tarkoituksena on antaa kosketuspinta vapaasanahaulle ja henkilötietojen muokkaukselle ja lisäämiselle sekä näyttää käyttäjän haluamat henkilötiedot.

Vapaasanahaulla voidaan hakea tietoja mistä tahansa henkilötiedon kentästä (jotka on valittu näytettäviksi) halutuilla merkkijonoilla välilyönnillä eroteltuina. Esimerkiksi merkkijono ”Pasi Kuljettaja” hakee merkkijonoa ”Pasi” ja merkkijonoa ”Kuljettaja” kaikista henkilötiedoista ja palauttaa tiedot, joista löytyvät molemmat. Nämä kaksi merkkijonoa voivat olla joko samassa tai eri kentissä. Vapaasanahaun kentän sisältö lähetetään käyttäjän painallusten jälkeen (odotetaan puolen sekunnin verran viimeisimmästä painalluksesta ennen hakua) backendille PhonebookControllerille, josta palautetaan uudet henkilötiedot näytettäviksi. Tämän jälkeen tiedot korvataan ajonaikaisesti näytölle.



Kuva 4 Puhelinluettelon näyttösivu

4 HENKILÖLUETTELO

Henkilöluettelo-portletti on perustuksiltaan hyvin samanlainen kuin puhelinluettelo: sekin noudattaa portlettien käyttämää rakennetta, jossa sille määritellään erikseen ”view”- ja ”edit”-näkymät ja näille molemmille näkymille on määritelty omat controller-java-luokkansa. Henkilölista itsessään näyttää päällepäin hyvin samalta, kuin puhelinluettelo sillä erolla, että siitä puuttuu vapaasanahakukenttä. Tämä ei kuitenkaan tarkoita sitä, että henkilölista olisi vain katkottu versio puhelinluettelosta, vaan suurimmat erot löytyvät backendin puolelta.

4.1 Backend-toteutus

Portletin pääasiallinen tarkoitus on mahdollistaa räätälöityjen henkilölistojen luonti, muokkaus ja näyttö. Tätä mahdollistavat metodit saveList ja deleteList löytyvät HenkilostoEditController-luokasta. Listan luonnille ei omaa metodiaan tarvita, sillä saveList tarkistaa onko kyseessäolevaa listaa olemassa ja tallentaa uutena rivinä, jos ei ole. Listat tallennetaan JSON-muotoisena datana Liferayn räätälöityihin kenttiin.

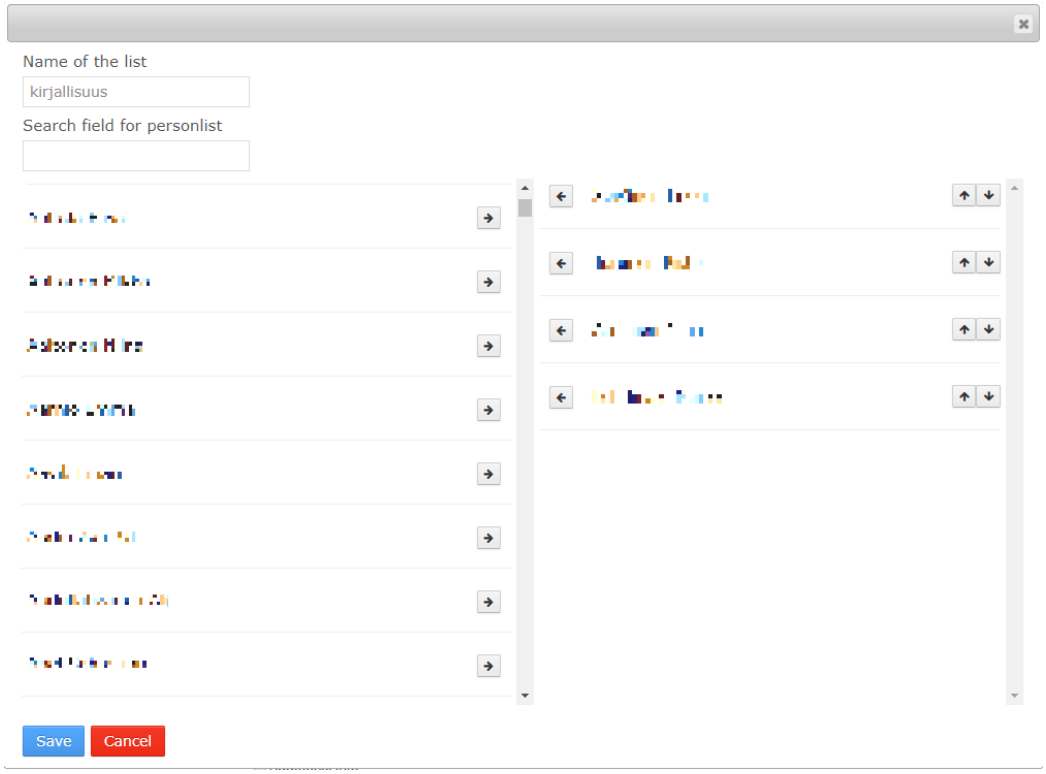
Listojen manipulointiin liittyvien metodien lisäksi HenkilöstöEditController toteuttaa puhelinluettelosta tutut createForm, save ja render -metodit, joilla luodaan, ylläpidetään, tallennetaan ja näytetään konfiguraatioon tarkoitettua lomaketta. Portletille voi konfiguroida puhelinluettelon tapaan sen, mitä kenttiä henkilöistä näytetään ja halutaanko näyttää kentille otsikoita.

HenkilölistaController-luokka sisältää portletin näyttösivulle tarvittavan backend-koodin. Tämä luokka toteuttaa ainoastaan render-metodin, jossa haetaan portletin preferences-arvoista näytettävän listan id, haetaan Liferayltä listaan kuuluvien henkilöiden id:t ja haetaan henkilötietojen käsittelytyökalulla id:itä vastaavat henkilöt. Lopuksi henkilöt annetaan eteenpäin jsp-sivulle.

4.2 Frontend-toteutus

Frontend-puolella henkilölista-portletin näyttöpuoli on hyvin samanlainen kuin puhelinluettelossa. HTML-elementtipuu itse listassa on sama. View-jsp-sivulle saadaan backendiltä joukko henkilöitä, jotka näytetään käyttäjälle.

Henkilölista-portletin konfiguraationäkymässä on portletin muokkaamiseen tarvittavat työkalut. Näihin kuuluvat listassa näytettävät kentät, valinta siitä, näytetäänkö kenttien otsikot, ja näytettävä lista. Näytettävän listan valinnan lisäksi käyttäjä voi myös muokata valittua listaa, poistaa sen tai lisätä täysin uuden listan. Listan muokkaus ja lisäys tapahtuu kuvan 5 näkymässä. Ensin sille tuodaan kaikki olemassaolevat henkilöt vasemmalle puolelle, lisätään jo valitut henkilöt oikealle puolelle ja poistetaan ne vasemmalta puolelta. Kaikki tämän näkymän logiikka toteutetaan Javascriptillä ja jQueryllä.



Kuva 5. Henkilölistojen muokkausnäkymä

4.3 Tietokanta

Portlettiin lisätään riippuvuutena henkilötietojen käsittelytyökalu Mavenin avulla samoin kuin puhelinluetteloonkin. Työkalun avulla portletille haetaan henkilötietoja näiden id-arvoja vastaan. Henkilölistaportletista ei voi tehdä itse henkilötietoihin mitään muutoksia, vaan kaikki muutokset tulee tehdä puhelinluettelon kautta.

Henkilölistat tallennetaan konfiguraationäkymän kautta Liferayn tietokantaan. Liferayllä on varattuna tietokantataulukko erinäisen räätälöidyn datan tallennukselle, ja tätä hyödynnetään henkilölistojen säilyttämiseen. Liferayn räätälöityihin kenttiin (expandofield) voidaan antaa yksi Java-luokka, käytössä olevan sivuston id-arvo sekä yksi tallennettava String-tyyppinen kenttä. Henkilölistadata oli Liferayn räätälöityjen kenttien toimintaperiaatteen vuoksi tallennettava siis yhtenä merkkijonona, joten data päätettiin tallentaa JSON-muodossa

5 YMPÄRISTÖ JA KÄYTETYT TEKNOLOGIAT

Henkilötietojärjestelmä luotiin asiakkaan Liferay-portaalille, jota ajetaan Apache Tomcat serverillä. Henkilötietojärjestelmä asennettiin serverillä kahtena WAR-tiedostoksi pakattuna portlettina, jotka käännettiin ja rakennettiin Apachen Maven-työkalulla. Järjestelmän frontend-osat luotiin JSP-sivuilla ja kaikki tallennettava data tallennettiin MySQL-palvelimelle. Kehityksessä käytettiin apuna myös versionhallintaohjelma Gitiä sekä Intellij Idea -kehitysympäristöä.

5.1 Backend-teknologiat

Henkilötietojärjestelmä rakennettiin jo olemassa olevalle Liferay-portaalille, jonka ympärille web-sivut on rakennettu. Projekti ja sen riippuvuudet käännetään ja rakennetaan Apachen Maven-työkalulla ja sitä ajetaan Apachen Tomcat palvelimella.

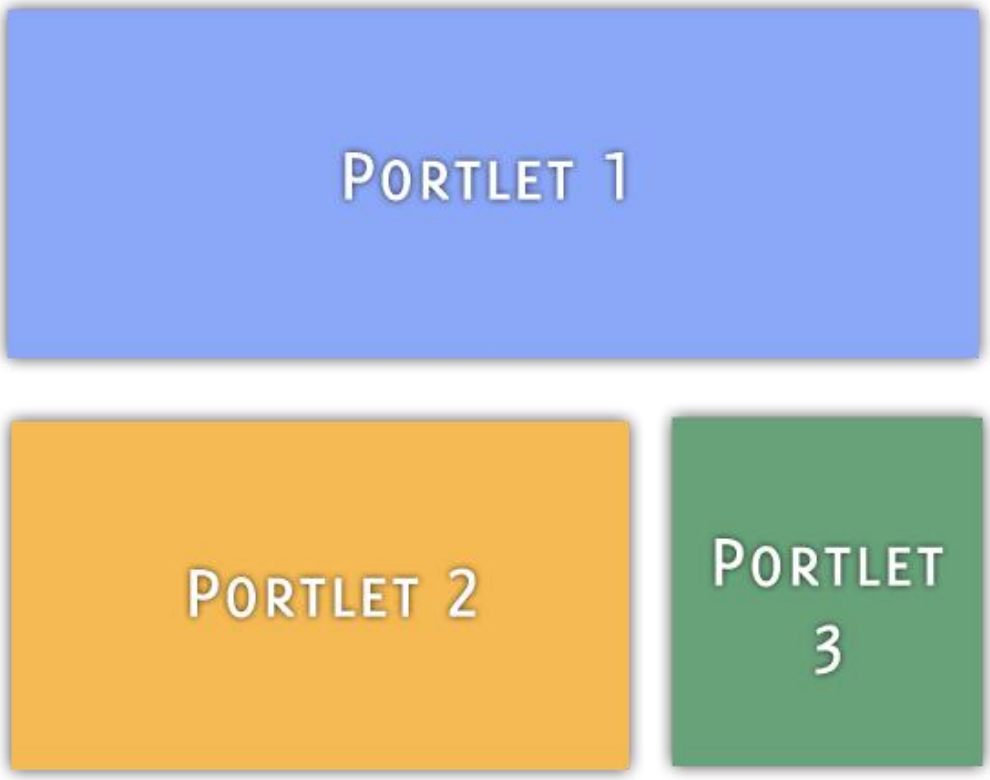
5.1.1 Liferay-portaali

Liferay tarjoaa avoimen lähdekoodin portaalituotteen, jonka tarkoituksena on antaa käyttäjälleen valmis verkkoportaalipohja. Tämän portaalipohjan päälle käyttäjä voi rakentaa yhden tai useamman sivuston, jotka voivat koostua yhdestä tai useammasta verkkosivusta. Portaali toteuttaa myös sisällönhallinnan, jolla käyttäjät voivat portaalin hallintapaneelista säädettyjen rooliensa ja oikeuksiensa puitteissa julkaista web-sisältöjä, joihin voivat kuulua muun muassa blogikirjoitukset, uutiset tai muut infotekstit. Lisäksi suurin osa portaalin lähdekoodista on ”hookattavissa” eli kehittäjät voivat korvata omistamansa portaalin lähdekoodeja omilla toteutuksillaan¸ luoden paljon uusia käyttökokemuksia ja -tarkoituksia verkkosivustoilleen.

Liferay-yritys perustettin vuonna 2004, jonka jälkeen siitä on kehitetty lukuisia uusia versioita. Viimeisin versio on Liferay 7.0, joka julkaistiin elokuussa 2016, mutta tämän työn alustana käytettiin kuitenkin Liferay 6.2.3:a, koska portaali luotiin ennen 7.0:n julkaisemista.

5.1.2 Portlet-komponentit

Portletit, kuten servletit, ovat web-sovelluksia, jotka toimivat komponentteina Liferay-palvelimella. Portletit ottavat Tomcatiltä vastaan erilaisia kutsuja, käsittelevät niitä sekä palauttavat tarvittavia vastauksia. Portletit ovat siitä erityisiä, että ne käsittelevät ainoastaan oman datansa, jonka ansiosta niitä voi asentaa portaalin sivulle useampia esimerkiksi kuvan 1 mukaisesti. Tämän ansiosta esimerkiksi työn henkilölistoja voi asetella haluamilleen portaalin sivuille häiritsemättä muiden komponenttejen toimintaa. (Liferay Developer -dokumentaatio, Portlets)



Kuva 6. Portlettien esimerkkiasettelu Liferay-portaalin sivulla

5.1.3 Spring-alusta

Liferay pohjautuu Javalle rakennettuun Spring-alustaan (englanniksi framework). Springin ominaisuuksiin kuuluvat mm. riippuvuuksien injektointi (dependency injection), käyttäjähallinta, tietokantatransaktioiden hallinta, lokalisaatio, metodien ajon ajastus ja paljon muuta.

Springiä ohjataan sille luoduilla annotaatioilla, joilla voidaan esimerkiksi hakea erikseen konfiguroituja JavaBeanejä eri luokkien käyttöön. Tämä on erityisen hyödyllistä esimerkiksi tapauksessa, jossa luodaan tietokannan kanssa kommunikointiin erillinen ”repository”-luokka, joka sisältää CRUD-operaatiot ko. entiteeteille. Springin avulla tämä luokka voidaan tuoda minkä tahansa Springin kontekstissa luodun luokan käyttöön ja suorittaa tietokantaoperaatioita loogisesti oman luokkansa kautta. Esimerkki Springin annotaatioiden käytöstä löytyy kuvasta 7. Kuvassa ohjataan Liferayn portletin näyttöpuolen (view) kutsut RequestMapping-annotaatiolla Controller-annotoidulle PhonebookController-luokalle. Lisäksi luokalle haetaan Autowired-annotaatiolla PersonnelApi-luokka henkilötietojen CRUD-operaatioiden suorittamista varten sekä sille haetaan erilaisia konfiguraatioita Properties-luokan mukana. (Spring Framework Documentation, 2018)

```
@RequestMapping(value = "VIEW")
@Controller
public class PhonebookController {

    @Autowired
    PersonelApi personnelImporterFacade;

    @Autowired
    private Properties puhluProperties;
```

Kuva 7. Esimerkki Java-luokalle annettavista Spring-annotaatioista

5.1.4 Maven-työkalu

Koko Liferay-projekti käännetään ja paketoidaan Apachen Maven-työkalulla ja täten sitä käytetään myös henkilötietojärjestelmän kääntämiseen ja rakentamiseen. Maven on työkalu Java-projekteihin, jolla voidaan ensisijaisesti määritellä projektin rakenne XML-tiedoston avulla (pom.xml – Project Object Model, kuva 8) ja esimerkiksi paketoida projekti .jar- ja .war -tiedostoihin. Maven lukee tämän tiedoston, jonka jälkeen se kääntää ja rakentaa projektin eri osat tiedoston määrittelemällä tavalla. Samalla Maven noutaa omasta repositoriostaan projektiin tai ”moduuliin” merkatut riippuvuudet, jotka ovat yleisimmin ulkopuolisia Java-kirjastoja. Kaikilla maven-artifakteilla tulee olla groupId, artifactId ja versionumero. (Apache Maven Documentation)

GroupId sijoittaa artifaktin sille asetettuun nimiavaruuteen, jolla eri kokonaisuuksiin liittyvien artifaktien löytäminen helpottuu, kun taas ArtifactId on artifaktin oma id. GroupId:t annetaan mavenille samassa muodossa kuin Javassa pakettien polku, eli esimerkiksi ”com.mycompany.app” ja artifactId on itse artifaktille annettava identifioiva nimi. Lopuksi määritellään kyseessä olevalle projektille tai moduulille riippuvuudet, jotka maven noutaa scope-asetuksella asetetussa vaiheessa. Esimerkiksi kuvassa 8 haetaan testausta varten junit-paketti silloin, kun projektille ajetaan test-vaihe.

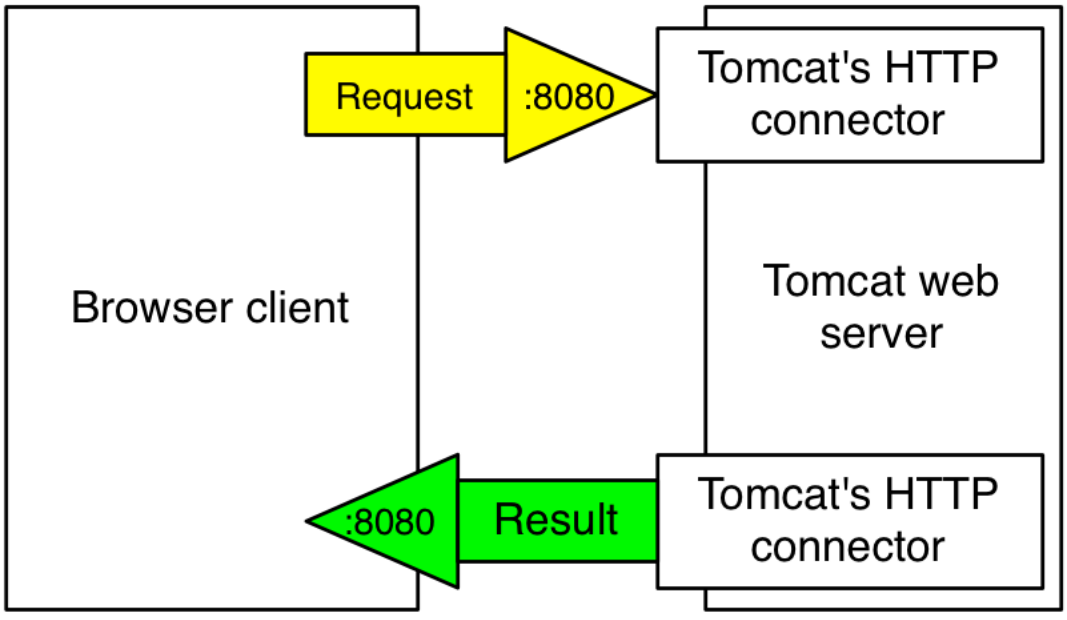
```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>Maven Quick Start Archetype</name>
  <url>http://maven.apache.org</url>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.8.2</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

Kuva 8. Yksinkertainen pom-tiedosto

Tässä projektissa Mavenin käyttö ilmenee niin, että käyttäjätietojen käsittelytyökalu on erillinen Maven-moduuli, joka tuodaan riippuvuutena sekä puhelinluettelo-, että henkilöstölistamoduuleille.

5.1.5 Apache Tomcat -palvelin

Apachen Tomcat on avoimen lähdekoodin palvelin, joka sisältää Java servlettisäiliö Catalinan. Java servletti on ohjelman osa, jonka tarkoituksena on saada pyyntöjä (web-kehityksessä yleensä HTTP) käyttäjiltä ja uudelleenohjata ne konfiguraatioidensa mukaisesti kehittäjän haluamille Java-luokille. Lopuksi, pyynnöstä riippuen, Tomcat ohjaa myös Java-luokan mahdollisesti palauttaman dokumentin (yleensä HTML-sivun) takaisin käyttäjälle. Catalinassa pyörii useita Liferayn portletteja samanaikaisesti ja Tomcatin tehtävä tässä yhteistyössä on ottaa käyttäjältä tulevat pyynnöt ja ohjata ne oikeille portleteille Liferayn käsiteltäväksi.



Kuva 9. Tomcatin HTTP request-response -vaihdon yksinkertainen kuvaus, käytössä portti 8080

5.2 Frontend-teknologiat

Moduulin frontend-puoli ei ole monimutkainen, joten uudempien templaattikielien sijaan päätettiin käyttää Liferayn muutenkin runsaasti käyttämiä JavaServer-sivuja (JSP). JSP-sivuihin ajauduin suureksi osaksi sen vuoksi, etten ollut aikaisemmin käyttänyt mitään templaattikieliä eikä projektin resursoinnissa ollut tilaa itselleni uuden teknologian opetteluun. Sen sijaan JSP-sivut hyödyntävät Javaa, johon syvennyin muutenkin projektin aikana.

5.2.1 JSP-sivut

JSP, eli JavaServer Pages, on Sun Microsystemsin kehittämä tiedostomuoto, jonka tarkoituksena on tarjota nopea ja helppo tapa tarjota dynaamista web-sisältöä. JSP-sivu koostuu tavallisesta HTML-koodista, johon on upotettu kahdenlaisia ulkoisia elementtejä – JSTL-merkintöjä (JSP Standard Tag Library) ja Expression Language -merkintöjä.

JSTL-merkinnöillä voidaan kääntää Java-koodia ja täten tuoda serverin puolelta JavaBeanien dataa HTML-sivulle Java ”skriplettien” avulla. Skriplettejä käytetään merkintätavoilla ”<% .. %>” tai ”<%= .. %>”. Näistä ensimmäisellä merkataan kokonaisen Javakoodiblokin alkua ja loppua, kun taas jälkimmäisellä merkataan ainoastaan yhden joko arvon tai evaluaation korvausta. Esimerkiksi kuvassa 10 oleva for-looppi ja muuttuja ’i’ evaluoidaan serverin puolella, jonka jälkeen käyttäjälle tuodaan sivu, jolla kaikki Java skripletit on muutettu selaimelle sopivaksi HTML-muotoiseksi tekstiksi ja kaikki arvot evaluoitu valmiiksi. (Oracle, JavaServer Pages Technology)

```
<p>Counting to three:</p>
<% for (int i=1; i<4; i++) { %>
    <p>This number is <%= i %>.</p>
<% } %>
<p>OK.</p>
```

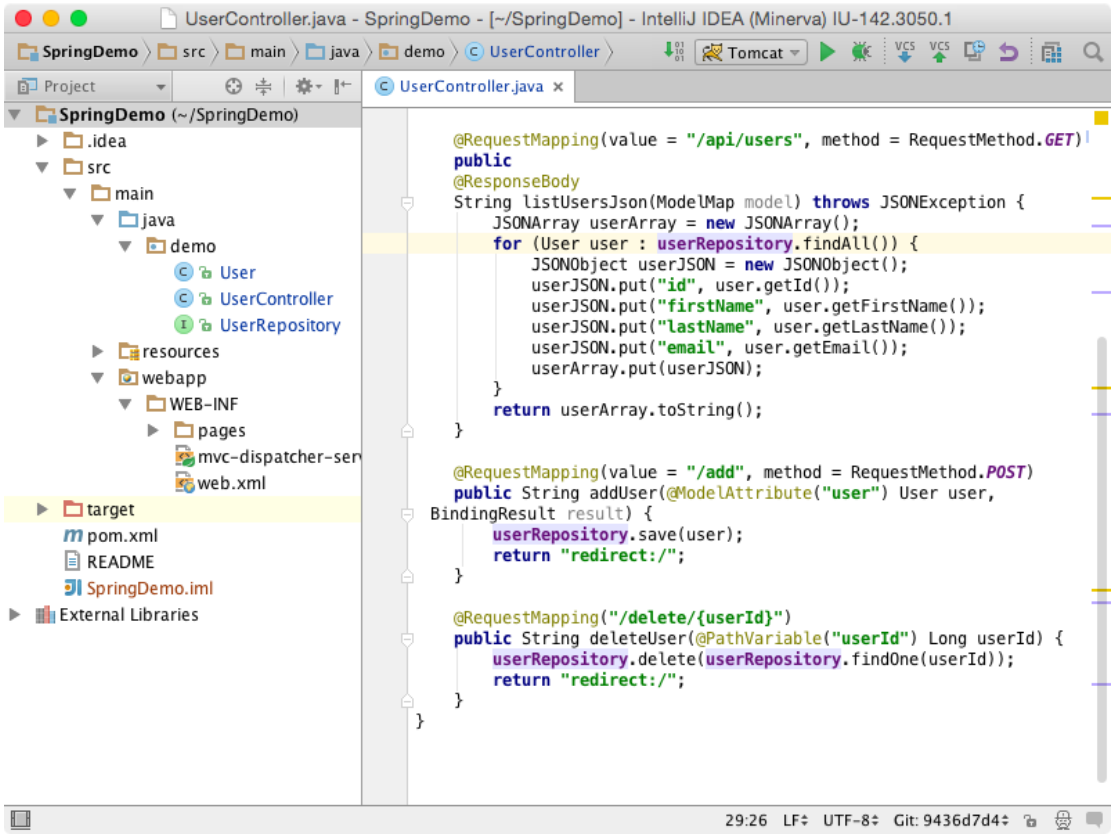
Kuva 10. JavaServer Pagen logiikkaa

5.3 Tietokanta

Liferay käyttää MySQL-kantaa oman datansa tallentamiseen. Päätimme käyttää henkilötietojen tallentamiseen samaa teknologiaa, kuin portaalissa on jo aiemminkin käytetty, joten päädyimme luomaan henkilötiedoille oman tietokantansa erilliselle MySQL-käyttäjälle.

5.4 Muut työkalut

Työssä käytettiin myös muita työkaluja, jotka eivät sisälly projektin rakenteeseen. Näihin kuuluvat Intellij IDEA -kehitysympäristöä sekä Git-versionhallintaohjelmaa. Intellij Idea on NetBrainsin vuonna 2001 kehittämä kehitysympäristö, joka tarjoaa erilaisia sovelluskehitystä helpottavia toimintoja, kuten koodin automaattista täydentämistä, integraatioita muihin eri palveluihin, kuten Gitiin ja MySQL-tietokantoihin suoraan kehitysympäristöstä sekä satoja erilaisia liitännäisiä, joilla se helpottaa kehittäjän työtä automatisoimalla eri toimintoja tai generoimalla valmiita luokkia. Ideasta käytettiin vuoden 2017 versiota. Kuvankaappaus Idean käyttöliittymästä kuvassa 11.



Kuva 11. Intellij Idea graafinen käyttöliittymä

Git on avoimen lähdekoodin versionhallintajärjestelmä, jolla voidaan helpottaa kehitystyötä yksin tai projektiryhmissä säilyttämällä projektin lähdekoodit sekä sen eri versiot verkossa. Gitiä voidaan käyttää monella eri tavalla, mutta tässä projektissa käytettiin Intellij Idean omaa Git-liitännäistä Gitin eri ominaisuuksien käytössä.

6 JATKOKEHITYS JA POHDINTA

Molemmat portletit ovat jo asiakkaan käytössä tuotantopalvelimella ja ne ovat ylläpidettävänä. Portleteille ei olla vielä suunniteltu toteutettavaksi laajoja ominaisuuksia, vaan asiakas ostaa pienempiä kokonaisuuksia muutostöinä. Jatkokehityksen työt tulevat olemaan todennäköisesti uusien kenttien lisäyksien, pienten tyylimuutosten tai näkymän muokkaukseen liittyvien konfiguraatioasetusten tyylisiä kokonaisuuksia.

Projekti sujui yllättävän sujuvasti ottaen huomioon, ettei minulla ollut projektia aloittaessa juurikaan käsitystä web-kehittämisestä, tai edes Java-kielestä. Kuitenkin projektin edetessä käytänteet tulivat tutuiksi ja työ saatiin päätökseen enemmän tai vähemmän asetettujen aikarajoitteiden puitteissa. Liferayn ja sen portlettien toiminnan sekä Springin toiminnan ymmärtämiset olivat suurimmat esteet projektissa, sillä ne ovat massiivisia kokonaisuuksia, joihin ensikertalaisen on hankala päästä alkuun. Lisäksi Liferayn dokumentaatio on jokseenkin hajanaista ja ripoteltuna blogikirjoitusten ja foorumikysymysten joukkoon. Spring taas on todella tehokas työkalu, jonka opettelu on varmasti projektin paras anti. Kaikessa monimutkaisuudessaan Springiä on loppujen lopuksi yllättävänkin helppo käyttää, kunhan vain saa kiinni muutamasta konseptista sen ympärillä. Sen osaaminen helpottaa varmasti myös tulevaisuudessa vastaan tulevien kehysten opettelussa.

Asiakkuus on pysynyt ja pysyy toistaiseksi ennaltaan: Liferay-ympäristöön liittyvät huoltotyöt, uudet ominaisuudet ja Liferayn logiikan muokkaus tilataan vielä asiakkaan puolesta.

LÄHTEET

Apache Tomcat kotisivut, luettu 11.3.
http://tomcat.apache.org/

Apache Maven Documentation, POM Reference, luettu 11.3.
https://maven.apache.org/pom.html/

Spring Framework Documentation, luettu 11.3.
https://projects.spring.io/spring-framework/

Oracle, JavaServer Pages Technology, luettu 13.3.
http://www.oracle.com/technetwork/java/javaee/jsp/index.html

Liferay Developer Documentation, luettu 17.5.
https://dev.liferay.com/fi/develop/tutorials/-/knowledge_base/6-2/tutorials

Liferay Portlets, luettu 17.5.
https://dev.liferay.com/fi/develop/tutorials/-/knowledge_base/6-2/developing-jsp-portlets-using-liferay-mvc