Websovellus tietokannan käyttöön

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulun opinnäytetyö

Tieto- ja viestintätekniikka, Riihimäki

Kevät 2021

Iiro Niemelä

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tieto- ja viestintätekniikka Tiivistelmä

Riihimäki

Tekijä Iiro Niemelä Vuosi 2021

Työn nimi Websovellus tietokannan käyttöön

Ohjaajat Jari Mustajärvi

TIIVISTELMÄ

Opinnäytetyön tavoitteena oli saada toimeksiantajalle ohjelma johon voi tallentaa simulaattoreiden kokoonpanotiedot ja tarkastella niitä. Ongelma työn aloitusvaiheessa oli tietojen tallennus taulukkoon. Tätä taulukkoa tuli simulaattoreiden käyttäjien ylläpitää. Kun simulaattorin kokoonpanoon tulee muutoksia, ne kirjataan taulukkoon ja aiemmasta kokoonpanosta ei sen jälkeen ollut mitään tietoja.

Manuaalisesti käsin kirjattavat tiedot lisäävät mahdollisuutta virheisiin. Automatisoimalla tätä työtä säästetään simulaattoreiden käyttäjien työaikaa muihin toimiin. Tätä dataa käytetään testausraportteihin ja laajalla tietokannalla niihin saadaan tarkempaa tietoa.

Lopputulokseksi saatiin tehtyä demoversio tietokantaohjelmasta, jota alettiin kehittämään perustelluin syin. Ohjelman käyttöliittymä on liitettävissä olemassa olevaan websovellukseen. Varsinainen tietokanta vaatii käyttöönottoa varten jatkokehitystä. Automatisointia varten tehtiin ohjelma joka hakee tietokannassa olevat objektit ja tekee niiden pohjalta uuden objektin johon syötetään simulaattorilta haettu data.

Avainsanat Full stack, Django, React

Sivut 22 sivua

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Information and Communication technology **Abstract**

Riihimäki

Author Iiro Niemelä Year 2021

Subject Web application for database

Supervisors Jari Mustajärvi

ABSTRACT

The goal of this project was to make program for the commissioning party to save simulator configurations and explore them. The problem in the beginning the was saving format. Maintaining the data was simulator users' responsibility. Every time there were changes made to the configuration, users had to write it down to the chart and after that there was no information about previous the configuration.

Maintaining data manually increases the possibility of error. Automating this system will release more work time for the users of simulators. This data will be used for software testing reports and it will be more accurate by having large database.

As a result, a demo version of web the application was made. User interface can be migrated to existing web application. Database program demands more development. To automate data harvesting, a script was made to get existing objects and recreate them with updated information.

Keywords Full stack, Django, React

Pages 22 pages

Sisällys

Kuvat, taulukot ja kaavat

Käytetyt termit ja käsitteet

API

Lyhenne englannin kielisestä nimikkeestä Application Programming Interface. Suomeksi tämä tarkoittaa ohjelmointirajapintaa. Tätä rajapintaa käytetään eri alustojen väliseen kommunikointiin. Tässä projektissa APIa käytetään käyttöliittymäohjelman ja tietokannanohjelman väliseen keskusteluun. (”Ohjelmistorajapinta”, 2021)

Full Stack

Web-järjestelmien yhteydessä Full Stack sanaparilla tarkoitetaan kokonaisuutta, joka kattaa tietokantajärjestelmän, käyttöliittymäsovelluksen ja palvelimen. (”Full stack”, 2021)

HTTP

Lyhenne englannin kielisestä nimikkeestä HyperText Transfer Protocol. Suomeksi tämä tarkoittaa hypertekstin siirtoprotokollaa. Tällä protokollalla siiretään näytölle piirrettävä dokumentti. (”Hypertext transfer protocol”, 2021)

JSON

Lyhenne englannin kielisestä nimikkeestä JavaScript Object Notation. JSON on kevyt tiedonsiirto- ja tallenusmuoto. Helpoksi sen tekee ihmisluettavuus ja useat tunnetut ohjelmointikielet osaavat tulkita JSON formaattia. (ECMA, n.d.)

Kokoonpano

Käännös englannin kielisestä sanasta configuration. Tässä kontekstissa kokoopanolla tarkoitetaan tietoja simulaattorin komponenttien nimikkeistä, niiden ohjelmistoversioista ja viivakoodeista. Lisäksi tietoihin kuuluu simulaattorin vastuuhenkilön nimi, hissikuilun kerrosluku, sähköistysmalli sekä hissin nopeus.

Simulaattori

Tässä kontekstissa simulaattorilla tarkoitetaan hissisimulaattoria, jota käytetään tuotekehityksessä. Simulaattori käsittää kaikki hissin sähköiset komponentit, joita tarvitaan yhden hissin käyttämiseen. Simulaattorilla testataan hissin ohjelmistoa.

1 Johdanto

Selaimella käytettävät sovellukset ovat nopeita ottaa käyttöön koska ne eivät vaadi käyttäjältä mitään asennusta, riittää kun sovellus on kertaalleen asennettu palvelimelle. Jokaisella toimijalla, joka tätä informaatiota käyttää, on käytössään päätelaite, jonka kautta sovellusta on mahdollista käyttää.

Työn tilaajana on KONE Oyj:n, tarkemmin Reliability Laboratory, joka on tuotekehitysosasto Hyvinkäällä. Tämän osaston tarkoitus on todeta tuotetun ohjelmiston ja järjestelmien toimivuuden tila. Hisseissä käytettävän ohjelmiston testauksesta kirjoitetaan raportti julkaisukelpoisen ohjelmiston julkaisun yhteydessä. Työn tavoitteena on saada tuotekehityksen käyttöön tietokantaohjelma simulaattoreiden konfiguraatioiden tallentamiseen. Näitä tietoja käytetään testausraporttien täydentämiseen ja näin ollen tavoitteena on lisätä testausraporttien validiteettia.

Tämän opinnäytetyön tarkoituksena on löytää sopiva tietokantasovellus. Ratkaisun löytämiseksi pohditaan eri vaihtoehtoja: kolmannen osapuolen palvelu, itse toteutettu sovellus tai olemassa olevien menetelmien tehostaminen. Työn tekijän motivaatio on oppia websovellusten suunnittelua ja toteutusta.

2 Yritysesittely

KONE Oyj on suomalainen yritys, joka valmistaa hissejä, liukuportaita sekä automaattiovia. Yritys on perustettu vuonna 1910. KONEen toiminta-alue on maailmanlaajuinen ja sen kotipaikka on Helsinki. Liikevaihto vuonna 2020 oli vajaa 10 miljardia euroa ja työllistää noin 60 000 henkilöä. Asiakkaita KONEella on noin 550 00 yli 60 maassa ja laitteita huoltokannassa on 1 400 000. Tuotekehitysyksiköitä on 8 kappaletta ympäri maailmaa. Tutkimus- ja tuotekehitysmenot ovat noin 174 miljoonaa euroa eli 1.8 % liikevaihdosta. KONEen missio on tehdä kaupungeista parempia paikkoja elää. Visio on luoda paras käyttäjäkokemus. (KONE, 2020, vuosikatsaus)

2.2 Nykyinen menetelmä

Opinnäytetyön aloitusvaiheessa konfiguraatiotiedot päivitetään manuaalisesti käsin taulukkoon. Kuvassa 1 on esimerkki tällaisesta taulukosta. Taulukoita on jokaiselle simulaattorille omansa ja niitä säilytetään pilvipalvelussa. Kun simulaattoriin tehdään muutoksia, esimerkiksi päivitetään komponentteja tai ohjelmistoja, muutokset tulee kirjata taulukkoon. Näitä taulukoita käytetään testausraporttien liitteinä. Jotta raportin laatija saa tiedot raporttiin, pitää hänen käydä etsimässä kyseisen simulaattorin kokoonpanotaulukko, ottaa siitä kuvakaappaus ja liittää se raporttiin.

Kuva 1. Kuvakaappaus taulukosta, johon kokoonpano tallenetaan.

SIMULATOR CONFIGURATION FILE					
UNIT:				RESPONSIBLE:	
SIMULATOR:				UPDATED:	
SIMULATOR NAME:					
SPEED:		SIDES:			
FLOORS:					
HW		Variant	Revision	PCS	Notes
Control system					
Accessories					
Drive					
Door					
Shaft					
Motor					
Car signalization					Details in Simulator signalization.xls
					Details in Simulator signalization.xls
Landing sign.					Details in Simulator signalization.xls

Taulukoiden ylläpito voi jäädä myös tekemättä. Tästä syystä testausraporttien validiteettia voidaan kyseenalaistaa, kun ei voi olla täysin varma kokoonpanotietojen paikkaansa pitävyydestä. Tämä saatta johtua simulaattorin käyttäjän puutteellisesta perehdytyksestä, joka on helposti korjattavissa. Muutoksia fyysisiin kokoonpanoihin tehdään harvemmin kuin loogisia muutoksia.

Laitteistoa vaihdetaan tarpeen mukaan, esimerkiksi jos tehdään uusi versio piirikortista tai vaihdetaan piirikortin tyyppiä ominaisuuksien takia.

2.1 Sisäisten työkalujen kehitys

Sisäisillä työkaluilla tarkoitetaan työkaluja, jotka eivät näy ulospäin asiakkaalle. Sisäisten työkalujen tarkoitus on tukea, nopeuttaa ja näin ollen parantaa työntekijän tulosta. Kehittäjän tulee tuntea käyttötarkoitus ja ymmärtää kenelle työkalua ollaan tekemässä. Korjattavan ongelman vaikutus tulee tunnistaa myös isossa kuvassa. Välttämättä vaatimusmäärittelyitä ei ole tehty, vaan aletaan heti tekemään. Tätä voi paikata kehittämällä työkalua läpinäkyvästi työkalun käyttäjien kanssa, jotta vaatimukset tulee täytettyä (Whorton, 2017) .

Sisäisten työkalujen kehittämisessä piilee myös ongelmia. Kuka ylläpitää, kuinka paljon tämä saa maksaa? Käyttöliittymästä ja käyttökokemuksesta on helppo oikoa kulujen kanssa, mutta silloin on vaarana käyttäjien kaikkoaminen. Ohjelman kehitys, jota ei käytetä tulee kalliiksi. Ulkopuolelta hankitun palvelun mukana saa ylläpidon, monitorointia ja vikojen korjausta. Työkaluja pitkään kehittäessä ohjelmisto monimutkistuu. Dokumentoinnin tulee olla kunnossa, jotta jatkossa mahdollisia ongelmia voidaan korjata. Ohjelmiston vaatimusten muuttuessa, myös työkalujen tulee olla mukautuvia (Harley, 2018).

3 Toiminnalliset vaatimukset

Käyttäjän näkökulmasta vaatimuksena on tietyn simulaattorin kokoonpanon löytyminen tietyltä päivämäärältä. Teknisesti tämä tarkoittaa, että ohjelmiston tulee tarjota mahdollisuus automatisointiin ja datan versiointiin. Automatisointia perustellaan sillä, että kerättävää dataa on paljon ja sen kerääminen manuaalisesti joka päivältä vie liikaa työaikaa. Jotta datan kerääminen saadaan automatisointua, tarvitaan rajapinta, jonka kautta automaatio keskustelee tietokannan kanssa. Kokoonpanojen versioinnissa pohdittiin mahdollisuutta pelkkien muutosten kirjaamiseen, jotta saadaan tietokannan kokoa piennettyä. Tämän toteuttaminen on vaikeaa ja mahdollista toteutus on alttiimpi vioille monimutkaisuutensa takia. Tietokannan koko on kuitenkin tietokoneelle pieni. Kaikkien simulaattorien kokoonpanot yhteensä taulukkomuodossa ovat 2

megatavua. Esimerkiksi SQLite tietokanta venyy 281 teratavun kokoiseksi (SQLite, n.d.), joten tämä ei aiheuta ongelmaa. Ylläpito, hinta ja jatkuvuus ovat myös perusteina valittavalle alustalle. Tulevaisuuden vaatimukset tulee ottaa huomioon. Sovelluksella pitää olla tilaa laajentua.

4 Teknologiavaihtoehdot

Tässä kappaleessa esitellään kolme erilaista vaihtoehtoa toteuksesta, joita voidaan käyttää tietokoneella selaimen kautta. Tällaisia ohjelmia kutsutaan web-sovelluksiksi. Web-sovellukset ovat käteviä nopean levityksen ansioista. Sovellusta voi käyttää kaikissa tietokoneissa, joissa on selainohjelma ja verkkoyhteys käytettävään palveluun, voi sovellusta käyttää. Jotain yhteensopivuuteen liittyviä ongelmia voi tulla, jos web-sovellus on kehitetty tietyn selaimen ominaisuuksia hyödyntäen.

Käytännössä web-sovellukset ovat selaimella näkyviä dokumentteja, jotka haetaan palvelimelta. Tätä dokumenttia voidaan muuttaa dynaamisesti hyödyntämällä esimerkiksi JavaScriptiä, Flashiä tai Javaa. Näihin dokumentteihin voidaan jättää nimikkeitä muuttujiksi, joihin haetaan tietokannasta arvo. Tämä arvo voi olla esimerkiksi kuva, tekstiä tai dokumentin osa. Web-sovellus voidaan näin jakaa kolmeen osaan: esitys, sovellus ja varastointi. (”Web-application”, 2021).

4.1 Valmiit palvelut

Valmiita selaimen kautta käytettäviä sovelluksia tuli vastaan yllättävän paljon. Tällaisista sovelluksista käytetään lyhennettä SaaS, joka tulee nimikkeestä Software as a Service. Ulkopuolisena palveluna hankittu sovellus on nopea ottaa käyttöön, koska sitä ei tarvitse alkaa ohjelmoimaan.

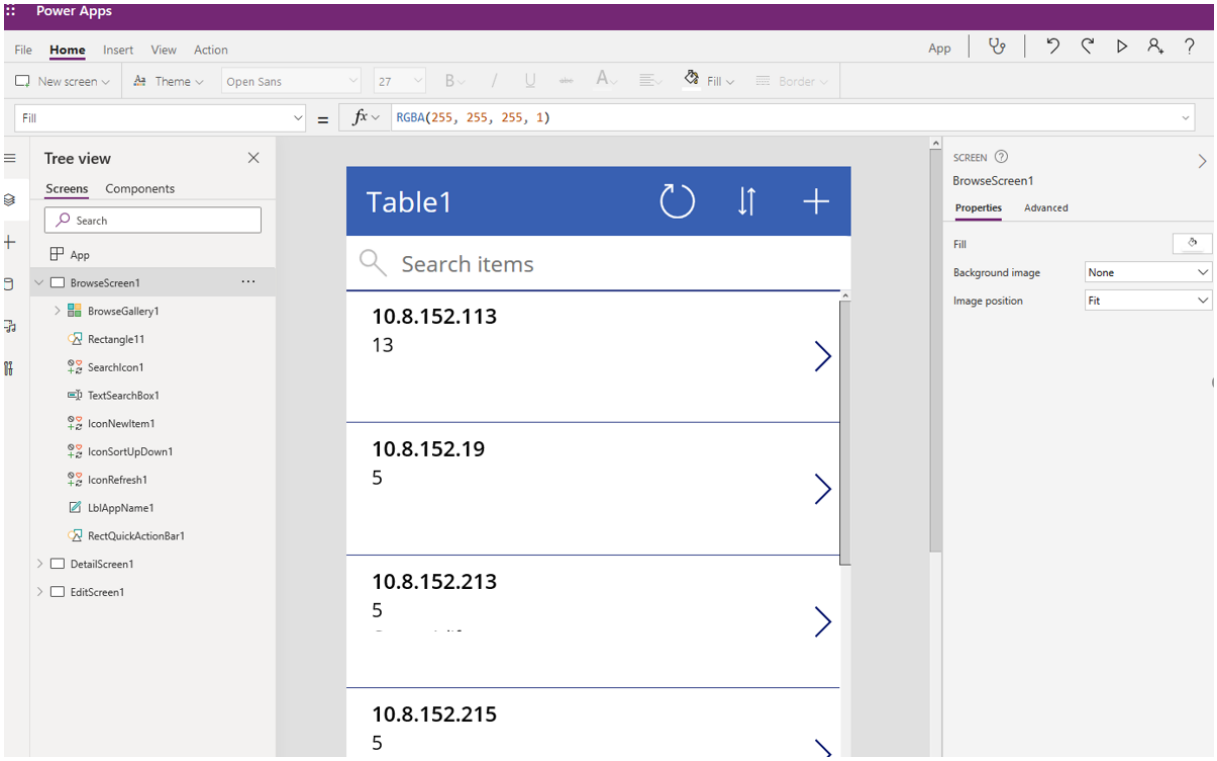
Kolmannen osapuolen palvelussa on tosin ongelma, tietoturva. Palveluiden turvallisuutta on vaikea lähteä arvioimaan nopealla aikataululla. Tietojen herkkyyden vuoksi yhtiön palvelimien ulkopuolelle tehtävä tietojen tallennus on riski. Tällöin tietokantaan pääsee myös yhtiön sisäisen verkon ulkopuolelta. Sisäisessä verkossa toimiva palvelin on suojattu yhtiön omilla palomuureilla. Lisäksi ulkopuolisen palvelun tulee olla luotettava. Monista vaihtoehdoista valikoitui Knack.

Knack tarjoaa mukautuvan tietokantaohjelman, joka ei vaadi käyttäjältä ohjelmointitaustaa. Palvelu maksaa 39-999 dollaria kuukaudessa, riippuen kuinka kattavana palvelun haluaa. Tarve on sen verran pieni, että mahdollisesti käyttöön otettavan palvelun hinta jää haarukan alempaan luokkaan. Selkeä dokumentaatio auttaa yhdistämään sovelluksen automaatioon. Rajapintaa varten Knack tarjoaa API avaimen ja tunnisteen.

4.2 MS Power Apps

Microsoftin Office E3 lisenssiin kuuluu Power Apps -sovelluskehitysympäristö. Se on tarkoitettu henkilöille, joilta ei vaadita varsinaista koodaustaitoa. Power Appsilla voi luoda puhelinsovelluksen sekä webbisovelluksen samalla kerralla (kuva 2). Tarvetta puhelinsovellukselle kuitenkaan ei ole. Kokeilin sovelluksen tekoa Excel-taulukon pohjalta. Tiedoston syöttäminen onnistuu suoraan Microsoftin pilvipalveluiden kautta. Power Appsilla ei itsellään ole tiedostojen versionti työkaluja, joita tarvitaan päivittäisten tietojen tallennukseen. Tämä voisi toimia käyttöliittymänä, mutta ongelmaa se ei sinällään ratkaise.

Kuva 2. Kuvakaappaus Power Apps sovelluksen tekovaiheesta.



Power Appsin sai nopeasti päällisin puolin toimimaan, mutta varsinaista tietokantaan liittyvää ongelmaa se ei ratkaissut. Microsoft on huhujen mukaan sulkemassa Access palvelunsa, jota olisi voinut käyttää Power Appsin tietokantana Excelin sijasta. Tästä oli vaikea löytää virallista tietoa. Alun perin huhut lähtivät liikkeelle jo 2017, mutta palvelu oli vielä käytettävissä. Microsoft ei ole julkaissut Office 365 blogiin vuoden 2016 jälkeen mitään liittyen Access palveluun (Microsoft, n.d.), joka herättää epäilyksiä palvelun tulevaisuudesta. Microsoft on julkaissut tuen sivuille Access Servicen päättymisestä. Access Service on sama asia kuin Access, mutta sille Microsoft tarjoaa myös palvelimen. Vapaa käännös artikkelista ”Emme enään suosittele Access Serviceä uusia sovelluksia ja verkkotietokantoja varten”. Samassa artikkelissa on ohjeet kuinka ladata palvelimella oleva tietokanta omaan talteen (Microsoft, n.d.).

4.3 Ohjelmistokehykset

Kolmas vaihoehto on tehdä sovellus itse. Web-ohjelmointikehyksiä on alettu kehittämään vuodesta 1993, jolloin keksittiin metodi HTML dokumenttien dynaamisten muutosten tekemiseen. Nykyisin sovelluksia varten on erilaisia Full Stack tapoja käyttötarkoitusten mukaan. Web-

sovelluksien yleistyttyä on ohjelmoinnissa tehtävien toistojen oikomiseksi luotu ohjelmistokehyksiä, joilla voi nopeammin saada sovelluksen perustat valmiiksi. Hyötyjä sovelluksen itse kehittämisessä on, että siitä saa juuri sellaisen kuin haluaa. Palvelimen ohjelmiston kehittämisen jälkeen tarvitaan alusta, jossa palvelin toimii. Alustana voidaan käyttää tietokonetta, virtualisoitua tietokonetta tai konttia.

Osastolla on käytössä sisäinen työkalu, joka on simulaattoreiden testausdatan analysointiin. Tämän käytössä olevan sovelluksen käyttöliittymä on tehty React-sovelluskehyksellä. Näin ollen ei koettu tarpeelliseksi pohtia muita vaihtoehtoja käyttöliittymälle. Jotta voidaan pienentää kehittämiseen kuluvia kustannuksia, on järkevää pyrkiä uudelleenkäyttämään olemassa olevaa ohjelmistoa. Lisäksi tarvitaan tietokantaohjelma ja siihen väliin rajapinta. Tietokannan ja rajapinnan toteuttamiseen monia vaihtoehtoja. Tätä sovellusta varten harkittiin NodeJS ja Django -ohjelmistokehyksiä. NodeJS on JavaScript ohjelmien ajoympäristö, joten sen lisäksi tarvitaan erilliset palvelin- ja tietokantakirjastot. Toisaalta Django on täysin sovelluksia varten kehitetty palvelinratkaisu. Django on helpompi ottaa käyttöön ja siinä on enemmän sisäänrakennettuja turvallisuusratkaisuja (Vats, 2020).

5 Sovelluksen teoriaosuus

Tietokannan hallintaan tarkoitetusta sovelluksesta käytetään nimitystä tietokantasovellus. Termillä viitataan usein ohjelmistoon, joka toimii tietokannan käyttöliittymänä. Tämän sovelluksen taustalla toimii myös tiedon varastointiin erillinen ohjelmisto, jota kutsutaan tietokannan hallintajärjestelmäksi (”Tietokannan hallintajärjestelmä”, 2020).

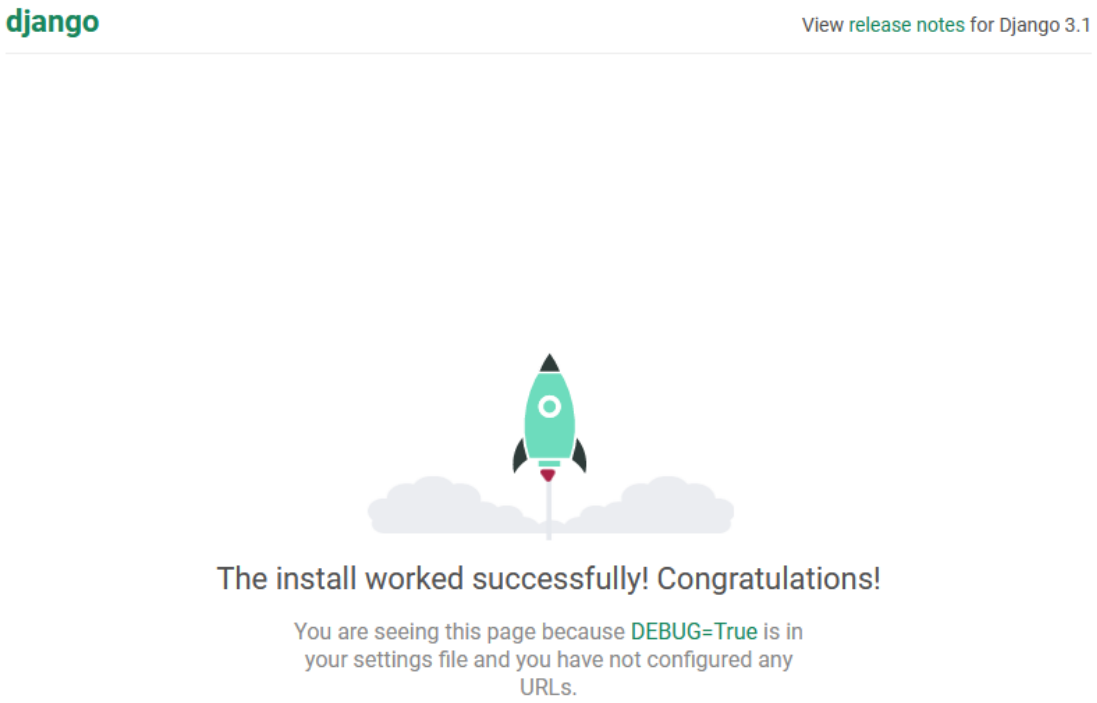
Websovellus tietokannan käyttöön tullaan tekemään osaksi olemassa olevaa sovellusta. Toteutustavaksi todettiin käyttöliittymän tehtäväksi ReactJS -kirjastolla ja tietokannan hallintajärjestelmä totetutetaan Django-ohjelmistokehyksellä. Perusteluita valinnoille on olemassa olevan järjestelmän teknologiaratkaisut ja Djangon tietoturvallisuus.

5.1 Django ohjelmistokehys

Django ohjelmistokehys on Python-ohjelmointikielellä kehitetty web ohjelmistokehys. Se sai alkunsa 2005 ja sitä ylläpitää voittoa tavoittelematon säätiö Django Software Foundation. Ohjelmistokehys korostaa komponenttien uudelleenkäytettävyyttä ja liitettävyyttä. Djangon tarkoitus on yksikertaistaa monimutkaista tietokantaan perustuvaa nettisivua. Django on maksuton ja se on ladattavissa projektin viralliselta kotisivulta (”Django (web-application”, 2021).

Uuden web sovelluksen voi aloittaa asentamalla Djangon. Sen voi joko hakea nettisivuilta tai asentamalla käyttäen Python ympäristöön komennolla `python -m pip install Django`. Tämän jälkeen haluttuun kansioon tietokoneella voidaan alustaa uuden projektin komennolla `django-admin startproject projektin_nimi`. Tässä vaiheessa sovellusten kehitystä nopeutetaan luomalla valmiiksi tietokanta ja alustamalla palvelimen asetukset. Käytännössä komennolla saadaan tietokannan hallintajärjestelmä, mutta sillä ei ole vielä mitään toimivuutta (kuva 3). Ohjelmiston toimivuuden voi todeta käynnistämällä palvelimen komennolla `python manage.py runserver`. Ohjelmisto toimii jos komentoriville ei tule virheilmoitusta. Jos palvelimen asentamisessa ei ilmennyt ongelmia, oletuksena sovellus toimii osoitteessa 127.0.0.1:8000. Kirjoittamalla tuo osoite selaimeen aukeaa Djangon oletusnäkymä (Django, n.d.).

Kuva 3. Kuvakaappaus Djangon oletusnäkymästä.

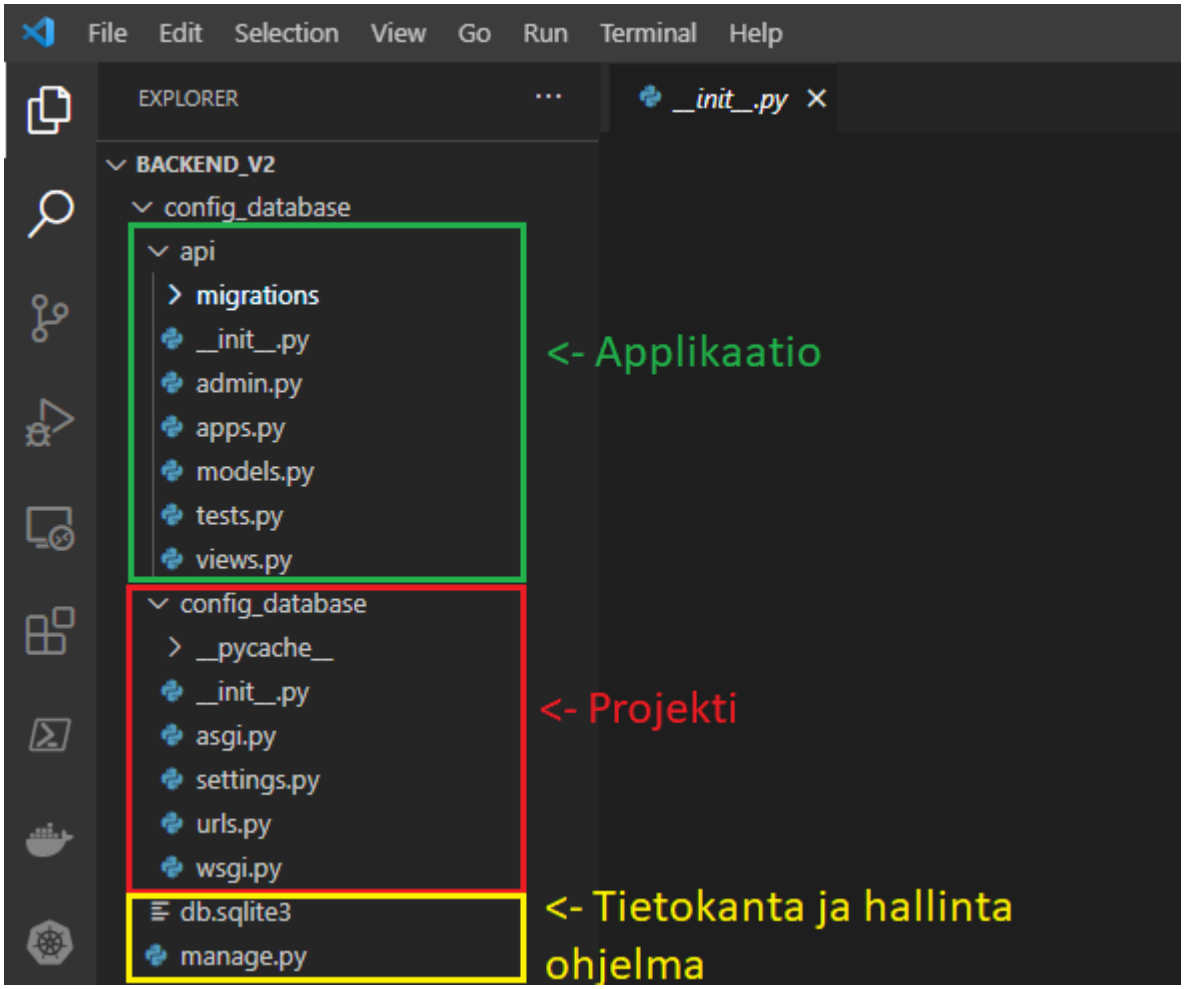


Komennolla `python manage.py startapp applikaation_nimi` luodaan ulospäin näkyvälle sovellukselle pohja. Kuvassa 4 on vihreällä värillä kehystettynä applikaation rakenne.

Django-ohjelmistokehyksessä projektilla ja applikaatiolla on eri tarkoitus. Projekti on kokoelma applikaatioista ja niiden asetuksista. Applikaatio on tarkoitettu käyttäjän rajapinnaksi. Projekti voi sisältää useamman applikaation ja applikaatio voi olla useammassa projektissa.

Projektin hallinnoitiin tarvitaan pääkäyttäjä, joka voi kirjautua hallintasivulle. Pääkäyttäjän voi luoda komennolla `python manage.py createsuperuser`. Hallintasivu löytyy oletuksena osoitteesta 127.0.0.1:8000/admin/. Jotta pääkäyttäjän voi luoda, pitää olla tietokanta, johon käyttäjät tallennetaan. Komento `python manage.py migrate` alustaa tietokannan. Migrate komento käy tarkistamassa mitä projektin asetusten INSTALLED_APPS taulukosta löytyy ja luo tai päivittää tietokannan sen mukaisesti. Käytettävä tietokantaohjelma määritellään DATABASES muuttujaan joka on oletuksena SQLite. Kuvassa 4 näkyy keltaisella tiedostorakenteessa paikka, jossa tietokanta sekä manage.py hallintaohjelma sijaitsee (Django, n.d.).

Kuva 4. Djangon oletus rakenne.



5.2 Tietokannan mallinnus

SQLite on relaatiomallia noudattavan tietokannan hallintajärjestelmä. SQLite ei tarvitse erillistä ohjelmointirajapintaa, koska tietokanta linkitetään suoraan sitä käyttävään sovellukseen. Relaatiotietokanta rakentuu tauluista. Taulu on kooste tietueista, jotka ovat tiedon tallennuspaikkoja. Yksi tietue on taulun yksi rivi. Tietue koostuu kentistä. Kenttä pienin yksittäinen osa tietokantaa. Tietokannassa taulut on varustettu uniikeilla tunnisteilla, joita kutsutaan perusavaimiksi. Samassa taulussa ei voi olla kahta identtistä perusavaimen arvoa. SQLite on julkaistu public domain -ohjelmistona, joten sitä saa käyttää, levittää ja muokata vapaasti ilman erillistä lupaa (”SQLite”, n.d.).

Applikaation alustuksessa luotava tiedosto models.py määrittelee tietokannan taulut. Django käyttää SQLiteä sulautetusti tämän tiedoston kautta. Tämä tarkoittaa, että varsinainen tietokanta näkyy kehittäjälle, mutta ei käyttäjälle, koska tietokanta on rakennettu osaksi ohjelmistoa. Tietokantaa mallintaessa ei tarvitse tuntea tietokannan syntaksia, koska tietokantaa käytetään Djangon kautta.

Tietokannasta haettavalle tiedolle suoritetaan serialisointi ennenkuin sitä käytetään Djangossa. Serialisointi tarkoittaa tiedon muuttamista muotoon, jossa se voidaan siirtää (”Serialization”, 2021). Serialisoitava tieto tallennetaan puskurimuistiin. Serialisoinnissa voidaan määritellä tietokannasta haettavien taulujen sisältö. Koko taulun sisältöä ei näin tarvitse hakea, vaan sen ulosannin yksittäiset kentät voidaan määritellä. Eri serialisointimalleja voidaan luoda samaan sovelluksiin eri käyttötarkoitukseen. Djangossa on valmiina tuki tiedon serialisointiin JSON, YAML ja XML formaatteihin (Django, n.d.).

5.3 Tietokannan käyttö

Jotta tietokantaa päästään käyttämään, pitää määritellä palvelimelle käsittelijä pyynnöille. Tämä käsittelijä on määritelty views.py tiedostoon, joka ottaa vastaan tehdyt pyynnöt ja suorittaa sen mukaisen toiminnon. Tietokantaa käytetään varsinaisesti näkymien kautta. Näkymät käyttävät serialisoinnissa määriteltyjä tauluja. Suodattimet määritellään näkymissä. Tietokannan käsittelyyn on Djangolla käyttössä CRUD-metodit. CRUD lyhenne tulee englannin kielisistä sanoinsa Create, Read, Update ja Delete. Näillä metodeilla voidaan luoda, lukea, päivittää tai poistaa taulujen mukaisia objekteja (Sankalp, n.d.) .

Näkymät saadaan käyttöön URL-osoitteiden kautta. URL on lyhenne englannin kielisestä nimikkeestä Uniform Resource Locator. Tällä osoitteella voidaan osoittaa palvelimen sijaintiin (”URL”, 2021). Palvelimen tehtävä on ymmärtää tämä pyyntö ja siihen vastataan sijainnissa olevalla tiedostolla. Palautettava tiedosto on serialisoinnin tekemässä muodossa. Osoitteet määritellään urls.py tiedostoon. Näitä osoitteita kutsutaan end pointeiksi, joita voidaan käyttöliittymään sijoittaa.

Django palvelin ottaa vastaan käyttöliittymän pyyntöjä, jotka ovat käyttäjän määrittelemiä. Pyynnöt ovat HTTP protokollan mukaisia. HTTP protokollaan kuuluu erilaisia metodeita, joita palvelin voi hyöndyntää. Tämän sovelluksen tarpeisiin ovat käytössä metodit GET, DELETE, POST ja PUT käytössä (Mozilla, n.d.). Näkymien tehtävä on kääntää nämä metodit tietokannan käsittelyyn kelpaavaan CRUD muotoon.

5.4 ReactJS

React on Facebookin kehittämä front-end-kirjasto JavaScriptille. React käyttää virtuaalista DOM ohjelmointikonseptia. Tällä tarkoitetaan dokumenttioliomallia, jossa dokumentin virtuaalista näkymää pidetään muistissa ja se synkronoidaan käyttäjän nähtäville. Näkymää päivitetään dokumentin komponenteilla, joita käsitellään olioina JavaScriptin metodeita käyttäen. Näitä komponentteja voidaan lisätä, muokata sisältöä tai poistaa reaaliajassa. React päättelee mikä sisältö on muuttunut ja muuttaa vain sen mikä on tarpeen. Tätä kutsutaan sovitteluksi. Käyttäjä komentaa komponenttia olemaan tietyssä tilassa ja React huolehtii sen toteuttamisesta. Lisäksi käytetään React-Bootstrap-kirjastoa, joka tarjoilee valmiita komponentteja.

Web-sovelluksen pohjana käytettävä sovellus määrittelee uuden käyttöliittymän teeman. Sovelluksen käyttäjät ovat yrityksen omaa henkilöstöä, joten sovelluksen käyttöliittymää ei tarvitse suunnitella erityisen houkuttelevaksi. Tarkoitus ei ole kerätä suurta käyttäjämassaa, vaan kehittää työn tehokkuutta.

5.5 Rajapinnan käyttö Reactilla

Käyttöliittymän komponenttien sisältö voidaan sitoa muuttujiin. Tämä sisältö voidaan hakea erillisestä tietokannasta. Tähän on tarjolla React API Hook. Sen kaksi työkalua useState ja useEffect mahdollistavat React-elementtien luonnin ilman luokan määrittelyä. useEffectiä käytetään kun käyttäjä suorittaa toiminnon. Esimerkiksi, kun painetaan nappia, useEffect tunnistaa tilamuutoksen ja suorittaa napille osoitetun function tai toiminnon. useState ottaa muuttujan edellisen arvon parametrina ja palauttaa sen päivitettynä. Näin muuttujien tila voidaan taas

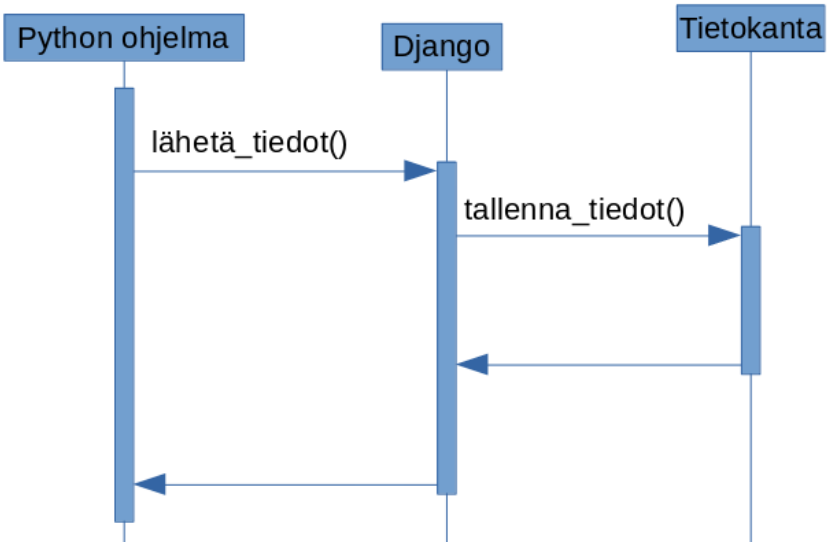
uudelleen sitoa funktioon. Näitä funktioita käyttäjä käskee ja React tunnistaa mahdollisen muutoksen muuttujan tilassa.

6 Sovelluksen tekeminen

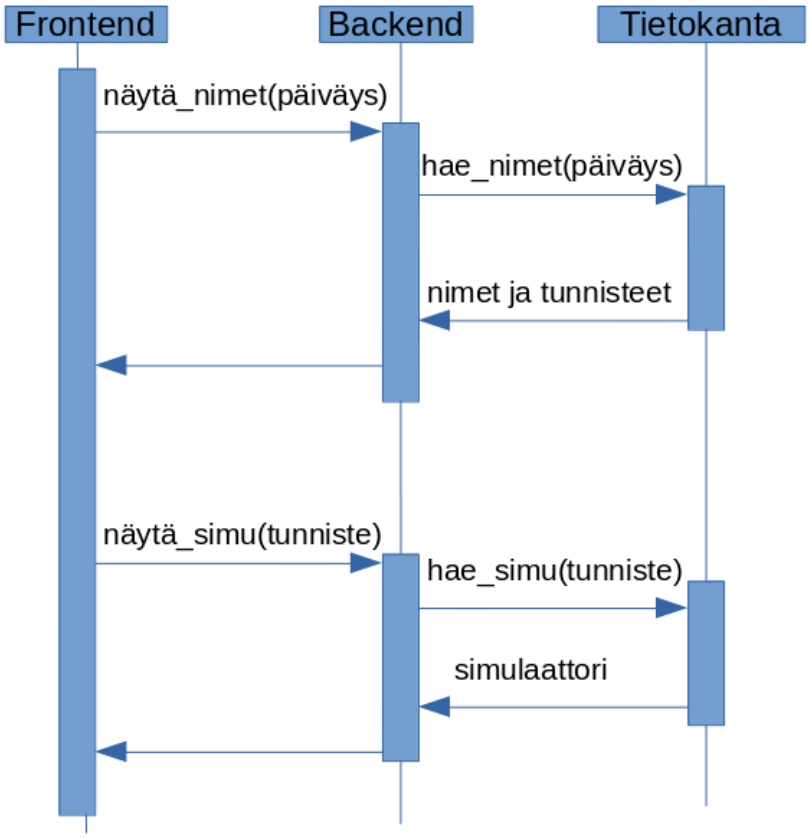
Sovelluksen perusajatus oli tallentaa kokoonpanotiedot joka päivältä ja näitä tietoja pitää päästä tarkastelemaan selaimen kautta. Sovelluksen teko tapahtui työn ohessa ja aiheeseen liittyvä opiskelu suoritettiin muulla ajalla.

Simulaattoreiden kokoonpanotiedot kerätään erillisellä ohjelmalla. Tiedot tallennetaan väliaikaisesti tekstimuotoon, jotka ovat Python ohjelmalla käytettävissä. Pythonilla saadaan yhteys Djangoon url-osoitteella ja pythonin requests-kirjastoa käyttämällä. Tämä tapahtumaketjun käynnistys on helposti automasoitavissa esimerkiksi unix-järjestelmän ajastettuna ohjelmana. Tietojen tallennus tapahtuu Djangon kautta. Kuvassa 5 on yksinkertainen kaavio tästä tapahtumasta. Jotta tietoja voidaan tarkastella myös selaimella, tarvitaan myös siihen toinen Djangon url-osoite. Kuvassa 6 on esitetty tiedonkulku käyttäjän nähtäville.

Kuva 5. Tietojen tallennus.



Kuva 6. Kuvaus tiedonkulusta.

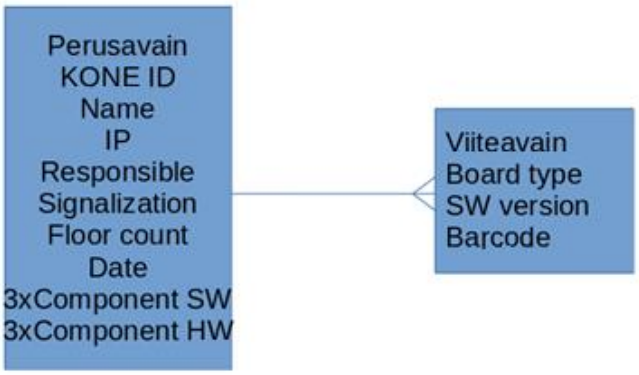


6.1 Tietokanta

Jokaiselta simulaattorilta tallennetaan konfiguraatiot joka päivältä. Simulaattori alustetaan tietokantaan luomalla taulu Djangon käyttöliittymän kautta. Simulaattorilta tuleva datapaketti on jo itsessään sen verran pieni, että monimutkaisen järjestelmän luonti todettiin tarpeettomaksi. Yksinkertaisesti joka päivä luodaan uudet taulut, jolloin erillistä muutoslokia ei tarvitse ylläpitää, koska tauluilla on päiväys ja perusavain. Lopullinen tietokanta käsittää äititaulun, johon tallennetaan päiväyksen lisäksi simulaattorin pysyvämmät tiedot. Näitä tietoja ei voida hakea simulaattorilta, koska ne ovat simulaattorin toiminnan kannalta merkityksettömiä. Äititauluun

kuuluu myös kolmen tärkeimmän komponentin tuotenimi ja ohjelmistoversio, jotka haetaan simulaattorilta. Aluksi ajattelin eriyttää tämän omaksi taulukseen, mutta se monimutkaisti järjestelmää turhaan. Yksittäiset kokoonpanoon kuuluvat komponentit tallennetaan kolmen tietueen lapsitauluina ja ne on sidottu äititauluun viiteavaimilla. Jos äititaulun poistaa, häviävät myös lapsitaulut. Lapsitaululla voi olla vain yksi äititaulu, mutta äititaululla saa olla useampi lapsitaulu.

Kuva 7. Tietokannan malli.



Äititaulu ja sen sidokset ovat simulaattorista riippuen vain muutamia kilotavuja. Jotta tietokanta ei kasva liian suureksi, määritellään myöhemmin tietojen elinikä. Jokaiselta virallisesti julkaistavalta ohjelmistolta tallennetaan testausraportit ikuisesti. Näiden mukana voitaisiin tallentaa myös sille raportille suunnattu konfiguraatio yhtä pitkäksi ajaksi. Muut versiot ovat vähemmän tarpeellisia. Tiedon eliniästä sovittiin toimeksiantajan kanssa kokouksessa 1.12.2020.

6.2 Tietokantaohjelma

Jotta tietokantaa päästään hyödyntämään, tarvitaan tietokantaohjelma. Sen tehtävä on muuttaa tietokannan data ihmisen luettavaan muotoon. Djangon avulla URL-osoite toimii pääsynä tietokannan sisältöön. Osoitteet määritellään urls.py nimiseen tiedostoon.

Tähän tiedostoon tarvitaan Djangon urls-kirjasto. Tällä kirjastolla käytössä on url-niminen funktio, joka ottaa muuttujana osoitteen ja funktion. Kuvassa 8 on esitelty työssä käytetty osoitetaulukko. Osoitteiden alkupää on korvattu r'^-merkkijonolla.

Taulukon kaksi ensimmäistä osoitetta ovat tietokannan lukemiseen. Name-osoite hakee päivämäärän mukaan simulaattorit NameView.as_view() metodia käyttäen. Id-osoite hakee perusavaimen mukaan yhden simulaattorin IDView.as_view() metodia käyttäen. Items- ja board-osoitteet ovat tietokantaan kirjoittamista varten. Items-osoite käyttää CreateElevatorItemView funktiota ja board-osoite käyttää CreateBoardView funktiota. Osoitteille annetut metodit ja funktiot ovat tuotu views.py tiedostosta.

Kuva 8. Osoitteiden määrittely.

```
1  from django.conf.urls import url
2  from .views import NameView, IDView, CreateElevatorItemView, CreateBoardView
3
4  urlpatterns = [
5      url(r'^name/(?P<date>.+)/$', NameView.as_view()),
6      url(r'^id/(?P<id>.+)/$', IDView.as_view()),
7      url(r'^items/$', CreateElevatorItemView),
8      url(r'^board/$', CreateBoardView),
9  ]
10
```

Tietokannan lukemiseen tarkoitetut luokat on esitetty kuvassa 9. Ne ovat perittyjä luokkia Djangon REST-kehitysympäristöstä. NameView-luokka palauttaa muuttujan date mukaan tietokannan taulut. IDView-luokka palauttaa perusavaimen mukaan yhden tietokannan taulun.

REST-kehitysympäristöstä löytyy luokat myös tietokantaan kirjoittamiseen, mutta kokeilun jälkeen päädyin tekemään toteuttamaan ne funktioina. Funktio ottaa http-pyynnön vastaan muuttujana.

Jos muuttujan metodi on POST, voidaan suorittaa kirjoitustapahtuma. Muuten osoite palauttaa virheilmoituksen vastauksena. Ohjelma ottaa kirjoitettavan tiedon JSON-muodossa. CreateElevatorItemView-funktio palauttaa lähettäjälle uuden taulun perusavaimen. CreateBoardView-funktio luo lapsitaulut vastaavalla tavalla, mutta käyttää CreateElevatorItemView-funktion palauttamaa perusavainta taulujen viiteavaimena.

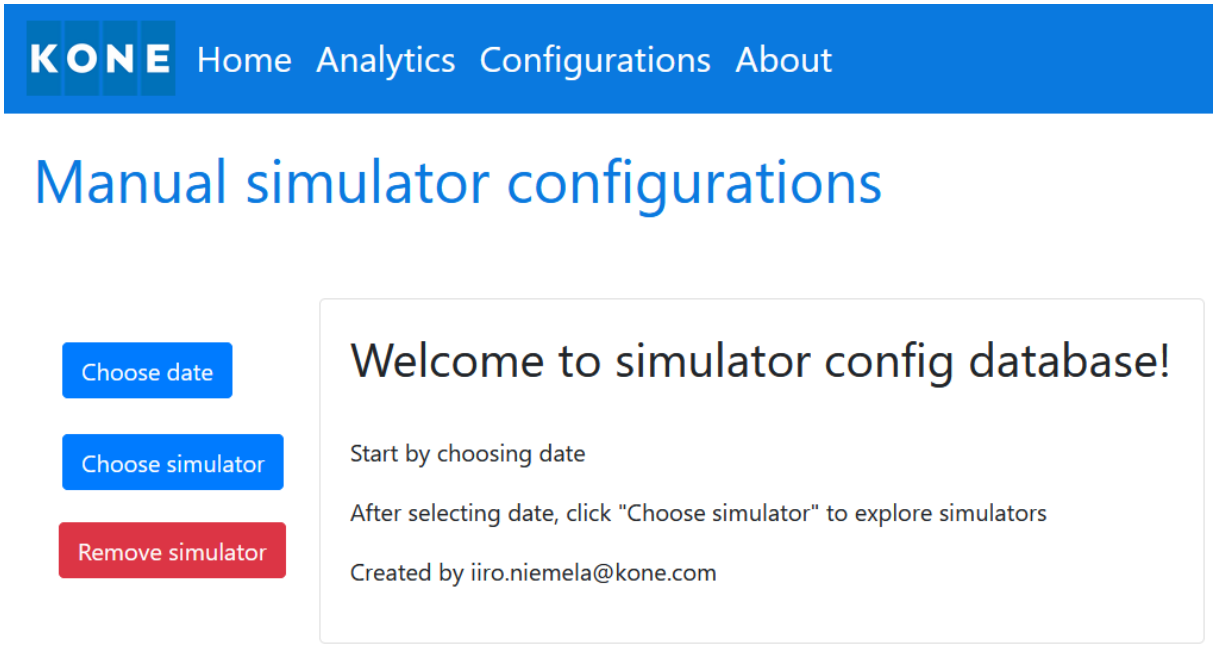
Kuva 9. Osoitteisiin liitettyjä toimintoja.

```
86
87  class NameView(generics.ListAPIView):
88      serializer_class = ConfNameSerializer
89
90      def get_queryset(self):
91          date = self.kwargs['date']
92          return Simulator.objects.filter(date=date)
93
94
95  class IDView(generics.ListAPIView):
96      serializer_class = ConfIDSerializer
97
98      def get_queryset(self):
99          identifier = self.kwargs['id']
100         return Simulator.objects.filter(id=identifier)
101
```

6.3 Käyttöliittymä

Sivun avautuessa ensimmäisen kerran ohjelman toivottaa käyttäjän tervetulleeksi. Yksinkertaiset ohjeet kertovat miten käyttäjän tulee toimia. Lisäksi on painike, jolla voidaan poistaa simulaattori tietokannasta, tämä toiminto on lähinnä ohjelman kehitystä varten, ei niinkään varsinaiseen käyttöön.

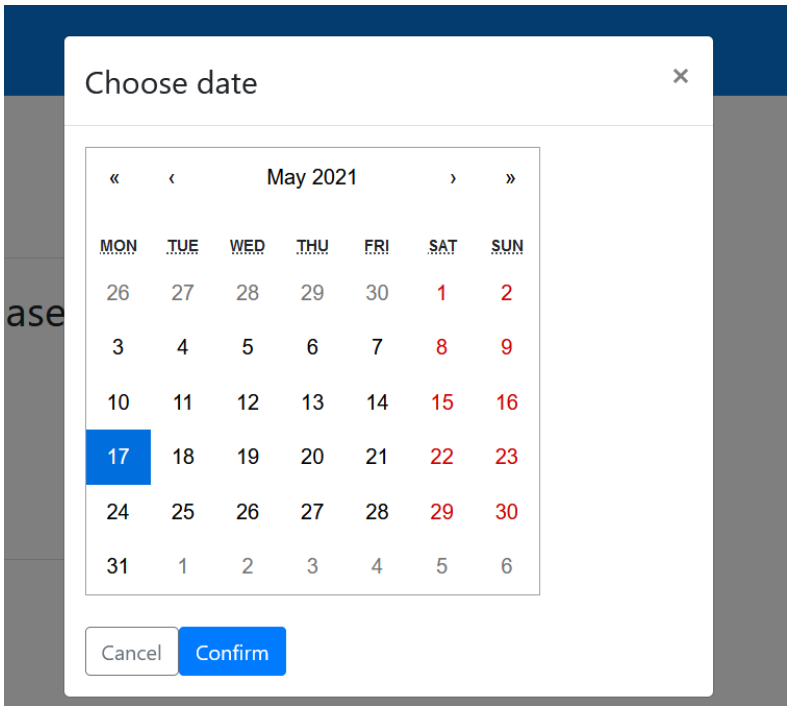
Kuva 10. Etusivu.



Päivämäärän voi kirjoittaa monella tapaa. Pidetään mielessä sovelluksen mahdollisuus skaalautua, asiakkaan toimistot eri maanosilla saattavat kirjoittaa päiväyksen eri tavoin. Jotta saadaan yhtenäisempi käyttökokemus, päivämäärän hakuun on luonnollista käyttää kalenteria. Kirjoitettava päivämäärä altistaa myös käyttäjän kirjoitusvirheisiin. React-yhteensopivia kalenteri löytyi valmiina. Kalenterin sidoin moduuliin, jonka näkyminen on yhden muuttujan varassa.

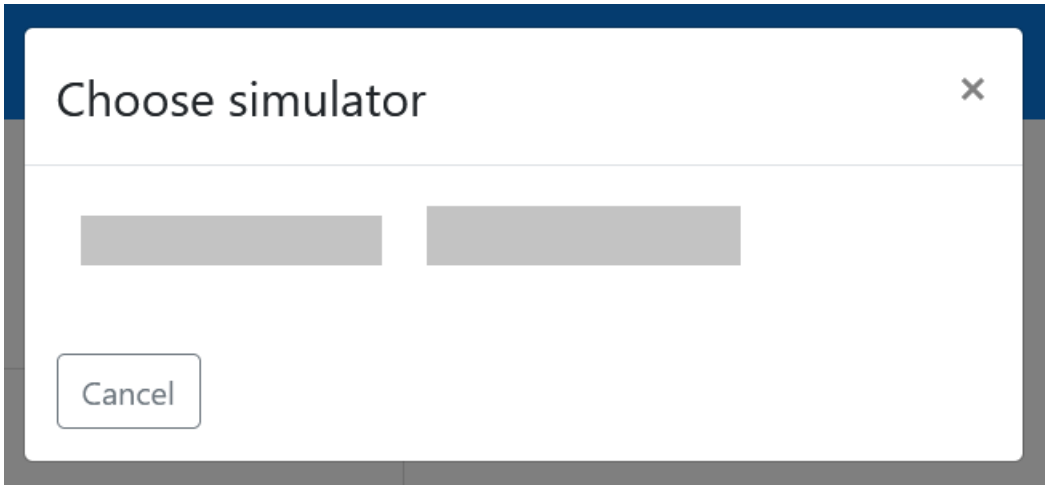
Confirm-painike tallentaa väliaikaisesti päivämäärän objektina. Tätä objektia käytetään parametrina tietokantaan tehtävään hakuun. Painike sulkee kalenterinäkymän.

Kuva 11. Kalenteri.



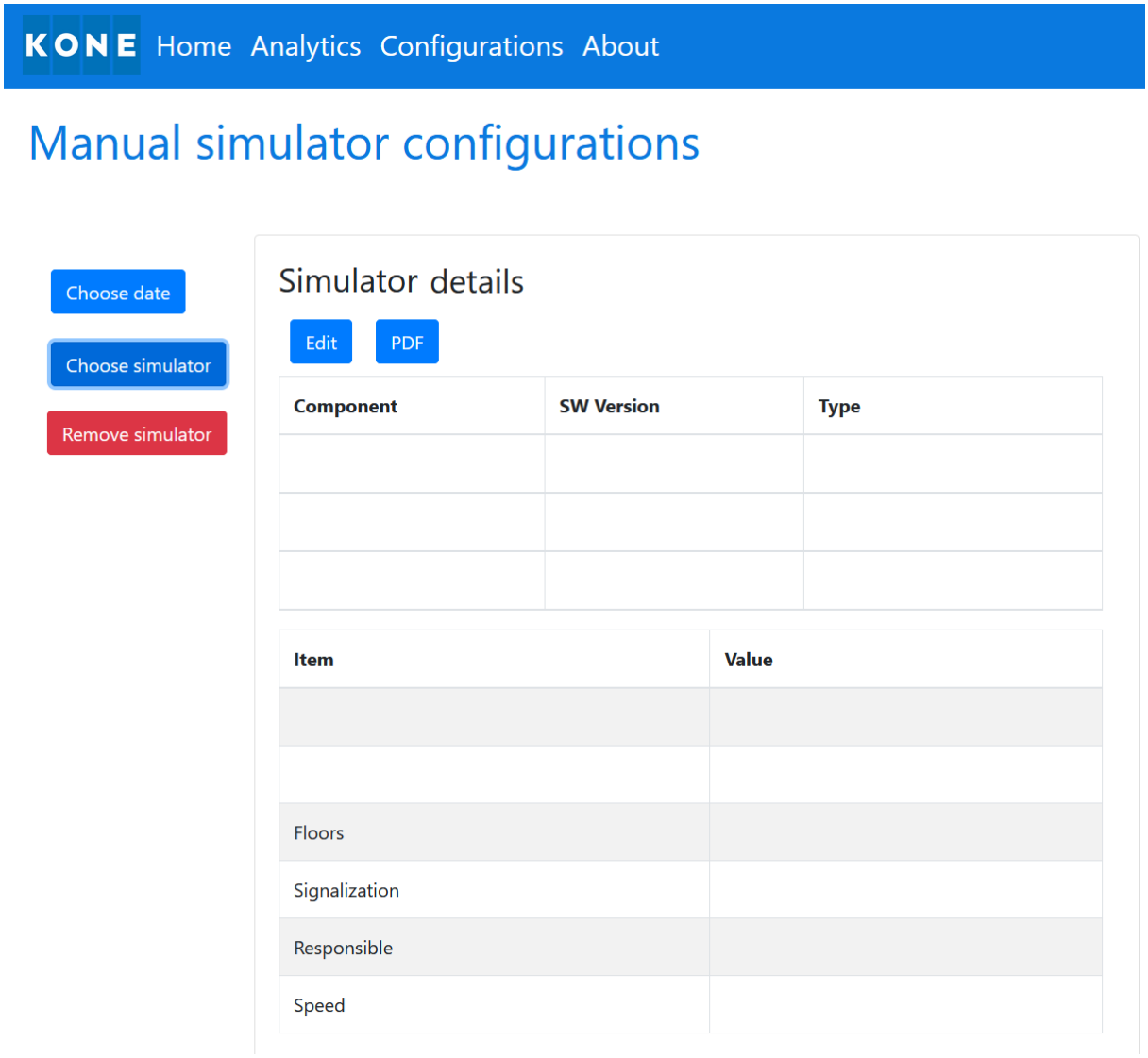
Ensimmäinen tietokantaa suoritettava haku tapahtuu painamalla Choose simulator-painiketta. Haku tehdään muodossa http://127.0.0.1/api/name/{päiväys}/?format=json ja palauttaa tietokannasta kyseiseltä päivämäärältä simulaattorien nimet ja niiden perusavaimet. Osoitteesta osio {päiväys} korvataan päivämäärällä. Käyttäjälle tulee näkyviin vain nimet kuvan 12 mukaisesti. React tekee nimistä paineltavia nappeja, joilla aktivoidaan uusi haku perusavainta parametrina käyttäen. Napin painallus sulkee nimilista näkymän ja suorittaa uuden haun. Tämä haku tehdään muodossa http://127.0.0.1/api/id/{perusavain}/?format=json ja se palauttaa perusavaimeen sidotun äititaulun sisällön ja siihen sidottujen lapsitaulujen sisällöt. Onnistuneen haun jälkeen simulaattorin tietosivu.

Kuva 12. Päivämäärältä löytyvät simulaattorit. Harmaat laatikot ovat sensurointia varten.

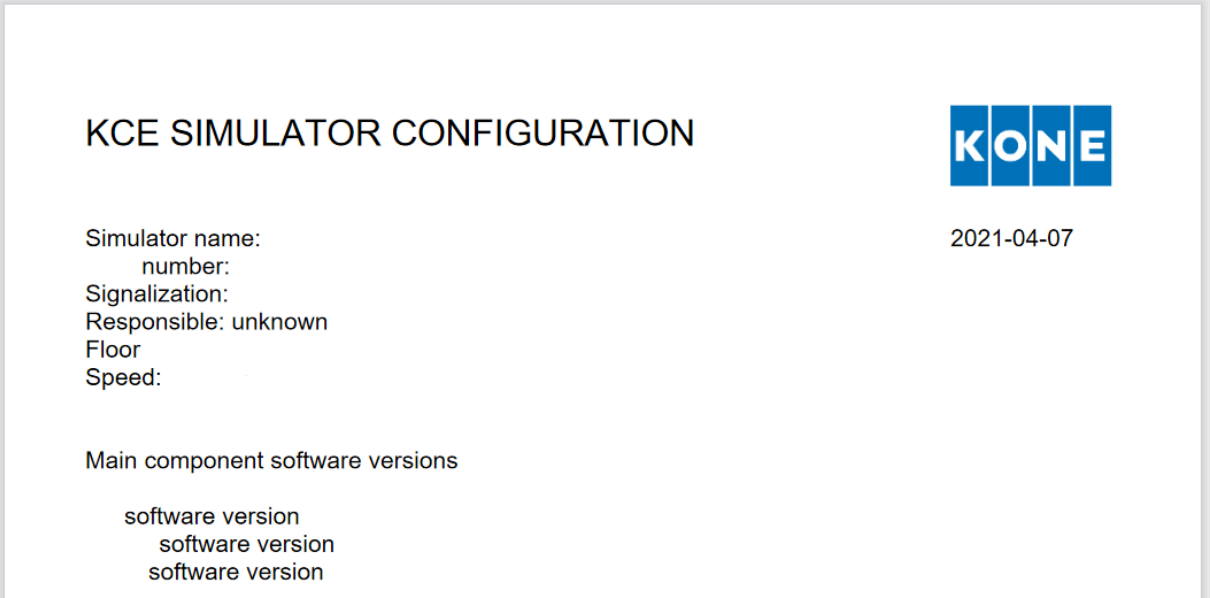


React:n vastaanoottama datapaketti pilkotaan muuttujiin. Näillä muuttujilla on oma paikkansa simulaattorin tietosivulla. Sivulla näkyy myös kaksi uutta painiketta. Edit-painikkeen taakse on myöhemmin aikomus tehdä mahdollisuus muuttaa simulaattorin tietoja, mitä ei voida hakea simulaattorista itsestään. PDF-painike avaa selaimeen kuvan 14 mukaisen lomakkeen, joka on täytetty vastaavilla tiedoilla.

Kuva 13. Simulaattorin kokoonpano. Tietueet ovat jätetty tyhjiksi kuvankäsittelyllä.



Kuva 14. Simulaattorin kokoonpano PDF muotoisena. Tietueet ovat tarkoituksella jätetty tyhjiksi.



KCE SIMULATOR CONFIGURATION

KONE

Simulator name: 2021-04-07
number:
Signalization:
Responsible: unknown
Floor
Speed:

Main component software versions

software version
software version
software version

6.4 React API rajapinta

Kun simullaattorin tiedot halutaan näyttää käyttäjälle, napin painallus kutsuu BoardData funktion joka ottaa parametrinaan simulaattorin perusavaimen. React tunnistaa muutoksen simuID muuttujan tilassa ja käynnistää kuvan 16. mukaisen funktion fectEnv.

Kuva 15. Muuttujat sidottuina useState funktioihin heti pääfunktion BoardData() alussa.

```
10    //define board data
11   const BoardData = ({ simuID }) => {
12     const [boardData, setBoardData] = useState([]);
13     const [items, setItems] = useState([]);
14     const [equipments, setEquipments] = useState([]);
15     const [contentLoaded, setContentLoaded] = useState(false);
```

Kuva 16. useEffect kutsuu fectEnv() funktiota parametrinaan simuID.

```
49
50     //API hooks
51     useEffect(() => {
52       fetchEnv(simuID);
53     }, [simuID]);
54
```

Kuva 17. Muutttujat saavat uudet arvot funktion fecthEnv() sisällä.

```
40       var board_array = [fetchedItems.data[0].board_data]
41       setBoardData(board_array)
42       setEquipments(fetchedItems.data[0].equipment)
43       setItems(fetchedItems.data[0])
44       setContentLoaded(true)
```

7 Pohdinta

Projektin kuvaus tähän opinnäytetyöhön rajattiin täysin käytettävään ohjelmaan ja siitä jätettiin pois taustalla toimiva ohjelma, joka suorittaa tietojen haun simulaattorista ja lähettää ne tietokantaohjelmalle. Oletuksena on, että tietokanta sisältää mallin mukaista tietoa, jota tarkastella. Projektia voisi jatkaa hyödyttämään myös simulaattoreiden käyttäjiä. Jos tulee tarve testata jotain tiettyä kokoonpanoa, voisi käyttäjä katsoa tietokannasta, millä simulaattorilla on kyseinen kokoopano. Demoversion tietokannan turvallisuutta pitää lisätä ennen käyttöönottoa.

Valmiina palveluna tarjottava Knack vaikutti työn jälkeen hyvältä vaihtoehdolta. Siinä on kaikki tarvittava mitä tietokannalta vaaditaan. Käytännössä Knack palvelun tarjoama API on hyödynnettävissä demoversiota varten tehtyä tiedon päivittämisohjelmalla. Vaihtamalla Knackin tarjoamat API osoitteet ja tunnistautumiset Python koodiin, ohjelma voisi toimia.

Lähteet

Django. (n.d.). *Serializing Django objects*.
https://docs.djangoproject.com/en/3.1/topics/serialization/#id2

Django (web framework). (3.6.2021). *Wikipedia-artikkeli*.
https://en.wikipedia.org/w/index.php?title=Django_(web_framework)&oldid=1026733611

Django. (n.d.). *Writing your first Django app, part 1*.
https://docs.djangoproject.com/en/3.1/intro/tutorial01/

Django. (n.d.). *Writing your first Django app, part 2*.
https://docs.djangoproject.com/en/3.1/intro/tutorial02/

Ekonoja A, Lahtonen T, Mäntylä J. (n.d.). *Relaatiotietokantojen peruskäsitteet*.
http://appro.mit.jyu.fi/doc/tiedonhallinta/tietokannat/index2.html

European Computer Manufacturers Association. (2017). *The JSON data interchange syntax* (ECMA-404). https://www.ecma-international.org/publications-and-standards/standards/ecma-404/

Full stack. (18.10.2021). *Wikipedia-artikkeli*.
https://fi.wikipedia.org/w/index.php?title=Full_stack&oldid=19241154

Harley, N. (2018). *Why building internal tools could become a costly mistake. Raygun*.
https://raygun.com/blog/building-it-yourself-vs-paying/

Hypertext transfer protocol. (28.4.2021). *Wikipedia-artikkeli*.
https://en.wikipedia.org/w/index.php?title=Hypertext_Transfer_Protocol&oldid=1020322699

KONE oyj. (2020). *Vuosikatsaus*. https://mb.cision.com/Main/18027/3275620/1364872.pdf

Knack. (n.d.) https://www.knack.com/

Microsoft. (n.d.). *Microsoft Office 365 blog*. https://www.microsoft.com/en-us/microsoft-365/blog/category/access/

Microsoft. (n.d.). *Access Services in SharePoint Roadmap*. https://support.microsoft.com/en-us/office/access-services-in-sharepoint-roadmap-497fd86b-e982-43c4-8318-81e6d3e711e8?ui=en-us&rs=en-us&ad=us

Mozilla. (n.d.). *HTTP request methods*. https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods

Ohjelmointirajapinta. (5.4.2021). *Wikipedia-artikkeli*.
https://fi.wikipedia.org/w/index.php?title=Ohjelmointirajapinta&oldid=19687132

Sankalp, J. (n.d.). Building a Django CRUD application in minutes.
https://www.sankalpjonna.com/learn-django/building-a-django-crud-application-in-minutes

Serialization. (22.1.2021). Wikipedia-artikkeli.
https://en.wikipedia.org/w/index.php?title=Serialization&oldid=1001983923

SQLite. (n.d.) Release history. https://www.sqlite.org/changes.html

SQLite. (22.3.2021). Wikipedia-artikkeli.
https://en.wikipedia.org/w/index.php?title=SQLite&oldid=1024466909

Tietokannan hallintajärjestelmä. (12.3.2020). Wikipedia-artikkeli.
https://fi.wikipedia.org/w/index.php?title=Tietokannan_hallintaj%C3%A4rjestelm%C3%A4&oldid=18788159

URL. (17.2.2021). Wikipedia-artikkeli.
https://en.wikipedia.org/w/index.php?title=URL&oldid=1007276500

Vats, R. (2020). Django vs NodeJS: Difference Between Django and NodeJS.
https://www.upgrad.com/blog/django-vs-nodejs-difference-between-django-and-nodejs/

Web-application. (31.3.2021). *Wikipedia-artikkeli*.
https://en.wikipedia.org/w/index.php?title=Web_application&oldid=1015241169

Whorton, T. (2017). Lessons learned from building Internal Tools. Agile Insider.
https://medium.com/agileinsider/lessons-learned-from-building-internal-tools-463c779af2e4