Lisensiointisovelluksen kehittäminen Djangolla

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

Riihimäki, tieto- ja viestintätekniikan insinööri

Kevät 2018

Jan Fincke

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tieto- ja viestintätekniikan insinööri
Riihimäki

Tekijä Jan Fincke **Vuosi** 2018

Työn nimi Lisensointisovelluksen kehittäminen Djangolla

Työn ohjaaja/t Petri Kuittinen, Tuomas Ursin

TIIVISTELMÄ

Python on ohjelmointikielenä yhdessä Djangon, websovelluskehyksen, kanssa yksi tehokkaimmista webkehitystyökaluista maailmassa, joilla luodaan kestäviä, helposti hallittavia ja optimoituja palveluita. Kerään tässä dokumentissa näistä teknologioista perusominaisuudet helposti ymmärrettävään kokonaisuuteen.

Projektin tavoitteena on toteuttaa lisensointipalveluverkkosivu, josta asiakkaat pääsevät hallinnoimaan omia lisenssejään sekä kaikkia kyseiseen lisenssiin liittyviä elementtejä. Verkkosivu toteutetaan yhteistyössä Cadworks Oy:n kanssa, joka tulee myös tarjoamaan ja hallinnoimaan sivua tulevaisuudessa. Projektissa tarvitaan myös monia erinäisiä lisäpalveluja, kuten versionhallintapalvelu Bitbucket, jonka avulla toteutetaan viankorjauksia, päivityksiä sekä verkkosivun julkaiseminen.

Linux-pohjaista Ubuntu-käyttöjärjestelmää käytetään palvelimena, joka tarjoaa projektimme verkkosivun. Palvelin sisältää useita tärkeitä lisäominaisuuksia, jotka ovat projektin onnistumisen kannalta välttämättömiä. Tähän lukeutuu erillinen verkkopalvelin, joka tarjoaa verkkosivun sisältöineen. Verkkopalvelin itsessään sisältää lisäosan, jolla Django-projekti sekä verkkopalvelin voivat kommunikoida keskenään. Verkkosivu vaatii tämän lisäksi tietokantasovelluksen, joka sisältää kaiken tiedon esimerkiksi lisensseistä, asiakkaista ja analytiikasta, sekä tehtävänkäsittely- ja jonosovelluksen, jolla toteutamme asynkronisia operaatioita.

Omiin tehtäviini kuuluu sekä asiakas- että palvelinpuolen ohjelmointia. Keskityn työssäni eritoten asiakaspuolen ohjelmointiin eli verkkosivun ulkoasun ja toimintojen tekemiseen, joten kerron siihen liittyen erilaisista tekniikoista.

Avainsanat django, tietotekniikka, ohjelmointi, ohjelmointikielet, python, sovelluskehykset

Sivut 95 sivua

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Information and Communication Technology
Riihimäki

Author Jan Fincke **Year** 2018

Subject Design of a licensing service on Django

Supervisors Petri Kuittinen, Tuomas Ursin

ABSTRACT

Python, a programming language, is widely used together with Django, a web development framework, to build a stable, optimized and manageable service for years to come. All the basic features and methods as to both technologies are covered in this document.

The goal of this project was to design and develop a licensing service website for customers to manage licenses and all aspects related to the said license. The website was built in co-operation with CadWorks Inc., who will also provide and administrate this website in the future. Many other services were also required to complete the project. A version management service, Bitbucket, was used to patch, update and deploy the website.

Ubuntu, a Linux-based operating system, is used on the server that contains our web service. The server includes many important add-in services which all have tailored roles in the application. A web server will provide the basis for the web-site and will also serve all static content. The web server itself will contain a plugin which serves the Django web application and provides communication between the server and the application, a database service storing all the information about licenses, customers, analysis etc. and a task/job queue with an application message broker which allows an asynchronous task and task queue implementation.

My accomplishments with this project contain elements of a full-stack project, with both client-side and server-side programming. My focus was mostly on the front-end side of things, which means I will mostly through went the techniques for implementing parts of the web page.

Keywords django, frameworks, information technology, programming, programming languages, python

Pages 95 pages

SISÄLLYS

1 JOHDANTO

Opinnäytetyön aiheena on luoda websivusto, josta asiakkaat näkevät ja hallinnoivat omiin organisaatioihinsa kuuluvia lisenssejä AutomateWorks- ja CustomWorks-ohjelmistoille. Käyttäjätyyppejä ja -ryhmiä on useita, joten näkymät luodaan näitä mukaillen peruskäyttäjästä hallintatason käyttäjään. Sivuston tavoitteena on automatisoida ja yksinkertaistaa lisenssin aktivointi, antaa asiakkaalle enemmän vaihtoehtoja hallita omia lisenssejään sekä pyrkiä vähentämään nykyiseen lisensointiratkaisuun liittyviä ongelmia.

Valitsin tämän aiheen mielenkiinnostani ohjelmointiin ja sovelluskehitykseen. Aikaisempi sovelluskehityskokemukseni muodostuu C#-, Java-, ja JavaScript-ympäristöissä tehdyistä pienemmistä kokonaisuuksista opiskelu- ja harjoittelumielessä, joten tämä on ensimmäinen oikea projekti, jonka avulla voin kehittää omia ohjelmointitaitojani pidemmälle ja työskentelemään uusien tekniikoiden äärellä.

Projektin luomiseen käytetään Python-ohjelmointikielen ympärille luotua, webohjelmointiin tarkoitettua Django-sovelluskehystä, joka tarjoaa tehokkaan kehitysprosessin, sovellusten merkittävän skaalautuvuuden sekä tietoturvallisen ympäristön. Django on myös käytössä useilla suuremmille yrityksillä maailmassa sen monipuolisuuden ja suoraviivaisuuden vuoksi. Pohdin myös, olisiko projekti voitu luoda hyödyntämällä muita tekniikoita sekä vertailen projektin kannalta vaihtoehtoisia Python-sovelluskehyksiä- ja tekstieditoreja ja kerron omia mielipiteitä kehitysympäristöstä ja -työkaluista.

Opinnäytetyön teoriaosuudessa esitellään Python-ohjelmointikielen peruskäsitteitä muuttujista olio-ohjelmoinnin eri osa-alueisiin, käydään läpi Djangon rakenne ja toimintaperiaatteita sekä projektin alustamisen ja kehittämisprosessin vaiheita. Perustelen myös, miksi valitsimme projektiin käytettäväksi Djangon vaihtoehtoisten sovelluskehysten sijaan.

Toiminnallisessa osuudessa keskitytään lisenssihallintasivuston kehitysprosessiin. Projektin palvelinympäristöön kuuluu erilaisia komponentteja, kuten webpalvelin, tietokanta ja asynkroniset tehtävät, jotka esittelen ja käyn läpi esimerkkejä, joita hyödynsin projektissa. Lisensointisivuston kehitysprosessissa käydään läpi käyttäjän rekisteröinti- ja kirjautumisnäkymien luomista sekä päänäkymien, joihin kuuluu mm. lisenssi- ja aktivointiavainnäkymät, teknistä toteutusta ja toiminnallisuutta.

Yhteenveto-osiossa kerron projektin kulusta, missä osuuksissa onnistuin ja missä olisi ollut parantamisen varaa, ja kerron myös siitä, miten tiimissä työskentely kehitti minua ohjelmistokehittäjänä. Näiden lisäksi tuon mietteitä uusien tekniikoiden opettelusta ja niiden soveltamisesta projektissa.

2 MIKSI DJANGO?

Python on ohjelmointikielenä erittäin helppo omaksua. Projektiin yhdistyi erilaisia komponentteja heti alussa, ja tuotekehityksen tehokkuutta ja ohjelmoijien taitotaustaa (minä ja kollega) ajatellen ei esimerkiksi PHP sopinut, koska vaikka se on suorituskyvyltään nopeampi, se olisi vaatinut enemmän opettelua, harjoittelua ja soveltamista.

Django on yksi merkittävimmistä edustajista websovelluskehysten uudesta sukupolvesta. Sen avulla voidaan kehittää korkeatasoisia websovelluksia säästäen aikaa. Django tarjoaa suoraviivaisuutta sovellusten rakentamiseen, ovelia menetelmiä toistuviin ohjelmointitehtäviin sekä selkeää ohjeistusta yleiseen ongelmanratkaisuun, antaen samalla kuitenkin tilaa omaan soveltamiseen kehitystyön aikana.

Testasimme ennen varsinaisen projektin alkua tekemällä demoprojektin verkkokauppasivustosta, jossa listattiin tuotteita ja tuotekategorioita. Demoprojektin tuloksena Django asettui tämän projektin kohdalla sen toteutusta ajatellen selkeäksi vaihtoehdoksi johtuen sen pääominaisuuksista, sillä projektimme kattaa monin paikoin monimutkaisia tilanteita, eikä pienempiin ohjelmallisiin ongelmiin haluta perehtyä, vaan halutaan suoraviivaistaa tuotekehitystä ja saada sovellus nopeasti käyttöön. Ohjelmistokehityksen rutiinityö Djangon kanssa muodostui minulle nopeasti ympäristön selkeyden vuoksi.

Suuren tukiyhteisön myötä Djangoon on saatavilla Python-pakettihallinnan kautta erittäin monipuolinen ohjelmapakettitarjonta. Ohjelmapaketit ovat Python-tiedostoja, jotka sisältävät tiettyyn tarkoitukseen soveltuvia koodipätkiä. Jos jokin tietty ongelma on yleinen, se on todennäköisesti jo ratkaistu ja siitä on olemassa paketti, jonka saa ladattua omaan projektiin. Esimerkiksi kokonaisen verkkokaupan rakentamista varten on olemassa paketti, josta löytyy tarvittavat perusmallit ja näkymät valmiina, joka tehostaa sovelluskehitystä.

Myös Djangon sisäänrakennettu tietokantatuki vaikutti valintaan. Tietokannan teko onnistuu Djangon kanssa, sillä se sisältää PostgreSQL-kannan oletuksena, eikä sen asentamiseen tai konfiguroimiseen mennyt liian kauan aikaa, vaan tietokantaa aloitettiin heti suunnittelemaan. Sovelluksen tietokantarakenteen pohja saatiinkin hahmoteltua jo muutamassa päivässä.

3 MINKÄ TEKSTIEDITORIN VALITSEN?

Tekstieditorin valinta ei ollut projektin alussa itsestään selvää. Tutustuimme demoprojektin aikana Visual Studio Code -nimiseen avoimen lähdekoodin omaavaan editoriin. Se tarjoaa lukuisia käyttökelpoisia ominaisuuksia, kuten koodin testaustyökalut, sisäänrakennetun Git-versiohallintasovelluksen työkalut sekä Microsoftin kehittämän IntelliSense-ominaisuuden, joka ennakoi syöttämääsi tekstiä ja tarjoaa automaattista syöttöä perustuen siihen, mitä olet kirjoittamassa. Editori tarjoaa muokattavuutta, sillä siihen on myös olemassa erilaisia lisäosia esimerkiksi JavaScriptille tai mille tahansa muulle nykyaikaiselle ohjelmointikielelle. Vaikka ominaisuudet olivat huippuluokkaa, editori kuitenkin lakkasi ajoittain toimimasta virtuaalikoneeni kanssa, jolloin työnteosta tuli hankalaa.

Päädyin vaihtamaan jo entuudestaan tuttuun Atom -nimiseen editoriin, joka on GitHubin kehittämä avoimen lähdekoodin tarjoava tekstieditori. Suorituskyvyltään Atom on myös hyvin lähellä Visual Studio Codea, eikä se myöskään kaatuillut läheskään yhtä paljon. Atom tarjoaa hieman suppeamman sisäänrakennetun tarjonnan, mutta lisäosatarjonta on myös tässä huikea. Tämä editori oli käytössäni pidempään, mutta lisäosien ajoittainen kaatuilu haittasi työskentelyä huomattavasti ja editori myös hidasteli tästä syystä. Atomiin ei myöskään voinut tallettaa profiilia, joka olisi sisältänyt kaikki jo käytössä olleet lisäosat, vaan uudelleenasennuksen yhteydessä kaikki lisäosat ja teemat tuli asentaa uudelleen.

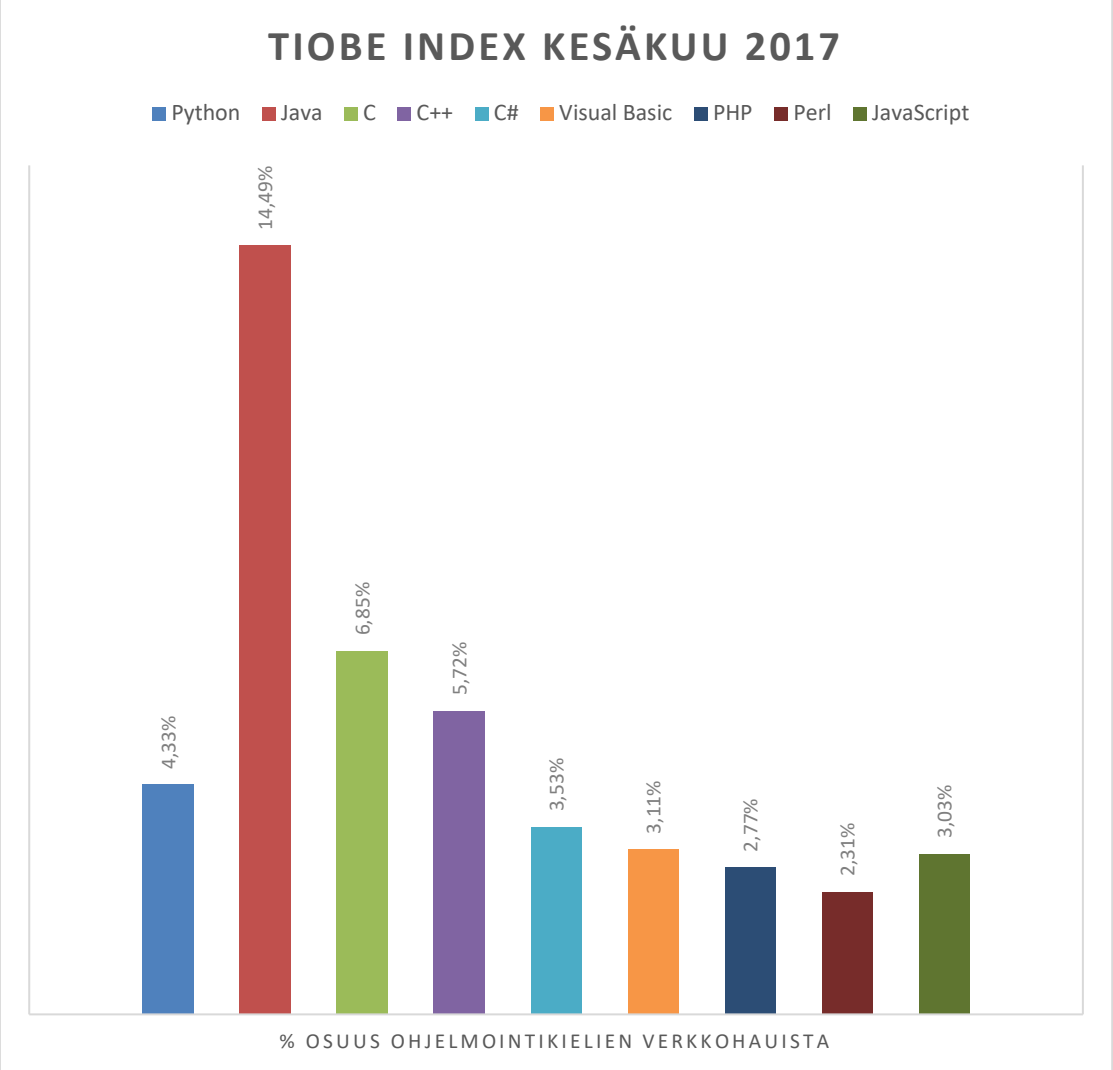
Kyllästyneenä epäluotettavuuteen päädyin vaihtamaan JetBrainsin kehittämään PyCharm-ohjelmointiympäristöön, joka tarjoaa kaiken tarpeellisen muokattavuuden ja luotettavan työskentely-ympäristön ja on ylivoimainen, mitä tulee Python- ja Django -sovellusten kehittämiseen. PyCharm on keskittynyt ainoastaan Python-sovelluskehitykseen. Se pohjautuu IntelliJ-nimisestä Java-tekstieditorista.

PyCharm sisältää myös täyden Django-tuen, joka tehostaa työskentelyä entisestään. Tähän kuuluu Django-näkymien syntaksin tunnistaminen ja automaattinen täyttö, integroitu terminaali ja testaustyökalu, visuaalisen Git-työkalun, sekä lukuisia ladattavia lisäosia. PyCharm tarjoaa asetukset eri sovellusten välillä; voin vaihtaa lennosta lisenssisivuston testauksesta lataussivumme testaamiseen. Tähän editoriin ilmestyi myös uutena ominaisuutena REST API -testaustyökalu, joka tuli myös tarpeeseen projektissamme.

Näistä kolmesta vaihtoehdosta ainoastaan PyCharm on maksullinen ja myös melko kallis, mutta saatavilla on kuukauden lisensioimaton versio sekä vuoden opiskelijaversio, joissa ei ole rajoituksia. Atom ja Visual Studio Code ovat ilmaisia, mukaan lukien ladattavat lisäosat. Python ja Django ovat kuitenkin korostetussa asemassa PyCharmin kehityksessä, jolloin myös kaikki asetukset ja vaihtoehdot ovat niiden ympärillä, mikä tekee siitä tällä hetkellä selkeän vaihtoehdon lisenssisivuston ja muiden Django-sovellusten kehittämiseen.

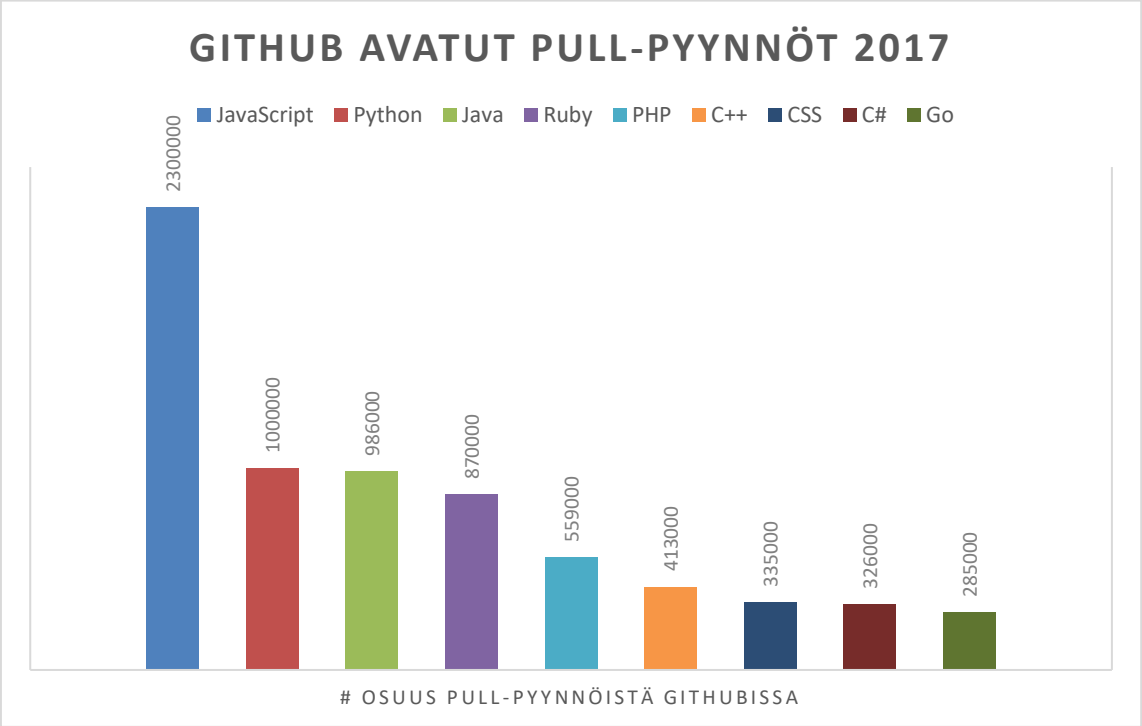
4 PYTHON

Python on ohjelmointikieli, joka yhdistää elementtejä olio-ohjelmoinnista sekä funktionaalisista ja proseduurillisista ajatusmalleista. Se mielletään usein ns. skriptauskieleksi, mutta tosiasiassa se on yleiskieli, joka kykenee skriptaustoimintojen lisäksi nykyään myös käyttöliittymä- ja tietokantaohjelmointiin sekä asiakas- ja palvelinpuolen ohjelmointiin. (Lutz 2013, 36-39.) Se oli kesäkuussa 2017 yksi maailman suosituimmista dynaamisista ohjelmointikielistä, kaiken kaikkiaan 5 suosituimman ohjelmointikielen joukossa (TIOBE Software BV 2017). Kuvassa 1 on esitetty TIOBE:n kesäkuun 2017 indeksin 9 suosituinta ohjelmointikieltä.



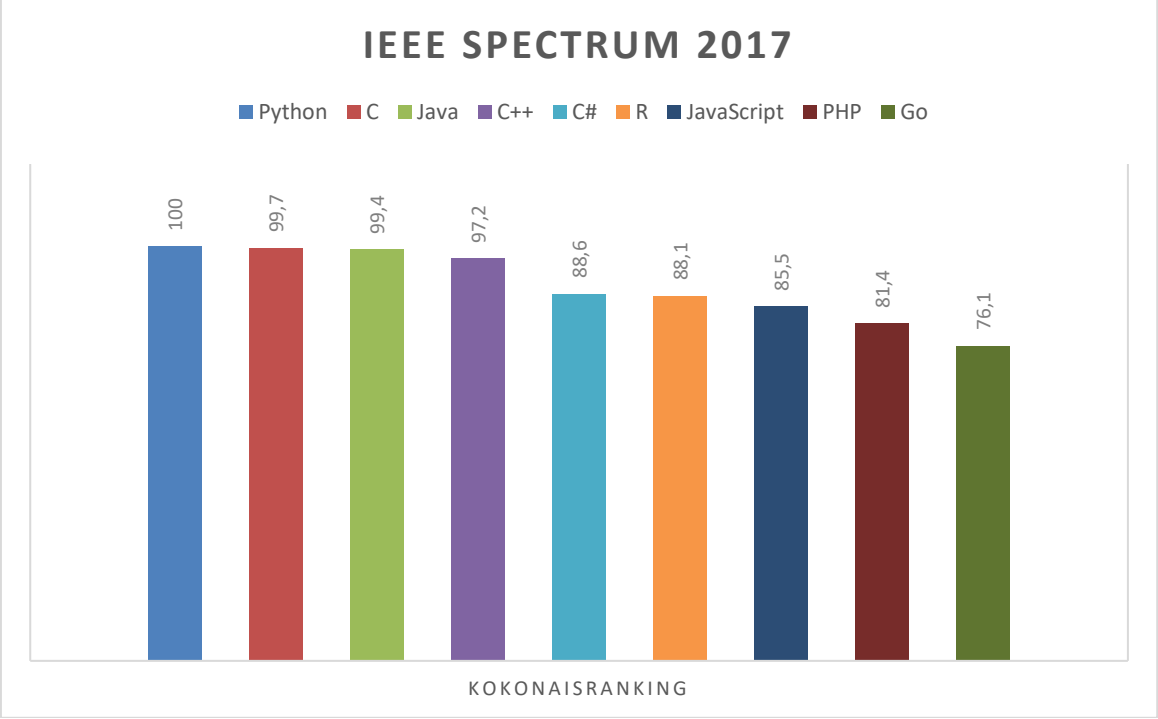
Kuva 1. TIOBE Index kesäkuussa 2017.

Python on sijoittunut myös hyvin kuvan 2 GitHubin (2018) vuoden 2017 tilastoon, jossa mitattiin osuutta avattuja ”pull-pyyntöjä”, jotka liittyvät versionhallintaan.



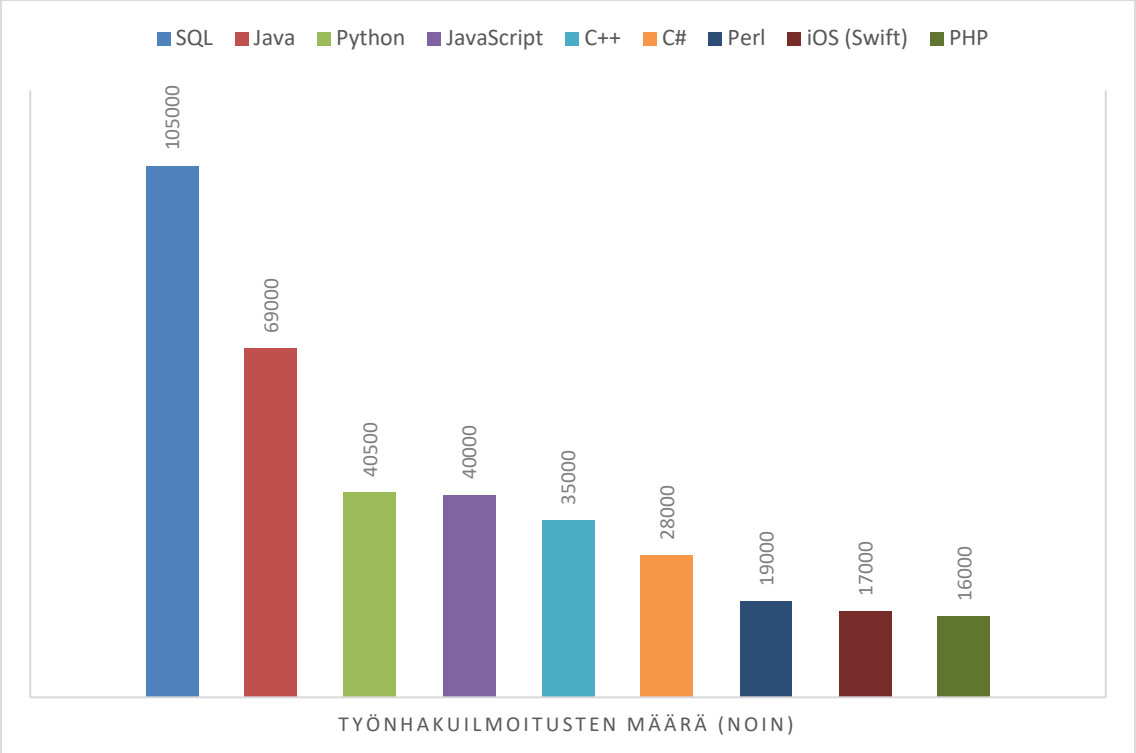
Kuva 2. Ohjelmointikielien osuus Githubissa avatuista pull-pyynnöistä vuonna 2017.

Kansainvälinen tekniikan alan järjestö IEEE (2018) on tilastoinut usean eri lähteen kautta yleistaulukon, jossa Python ylsi sijalle 1. Pisteytys kuvassa 3 koostuu mm. nopeammin nousevista kielistä, työnhakuilmoituksista eniten esiintyvistä kielistä ja suosituimmista kielistä avoimissa tietolähteissä.



Kuva 3. IEEE:n tilastoima ohjelmointikielien kokonaisranking vuonna 2017.

LinkedIn-bloggaaja Abuzhar Hussain (2018) listasi 9 suosituinta ohjelmointikieltä työnhakuilmoituksissa. Kuvassa 4 Python ylsi kolmanneksi. SQL on yllättävän korkeassa suosiossa työmaailmassa edelleen Javan ohella.



Kuva 4. Ohjelmointikielien osuus työnhakuilmoituksissa vuonna 2017.

Python on vahvasti mukana eniten nousevien ohjelmointikielien listoilla ja sen osaajilla on työmarkkinoilla kysyntää.

Python erottuu edukseen erityisesti vasta-alkajille johtuen sen helppolukuisesta kirjoitusasusta ja käyttömukavuudesta, jonka vuoksi myös koodin ylläpidettävyys ja uudelleenkäytettävyys ovat huomattavasti helpompaa kuin esimerkiksi perinteisimmissä alemman tason ohjelmointikielissä kuten C ja C++ (Lutz 2013, 34). Näille kielille se kuitenkin häviää suoritusnopeudessa, koska Pythonia ei käännetä binääritasolle samaan tapaan kuin alemman tason kieliä.

Pythonia voi tästä huolimatta kuitenkin käyttää samaan tapaan vaativimpiin laskutoimituksiin, robotiikkaan tai peliohjelmointiin käyttämällä *laajennuskirjastoja*. Hyvä esimerkki tästä on NumPy-laajennuskirjasto, mikä mahdollistaa tehokkaan numeerisen ohjelmoinnin Pythonilla. NumPy on yksi sadoista, ellei jopa tuhansista, Pythonin työkalukirjastoista.

Pyrin opinnäytetyössä esittelemään karkeasti Pythonin (versio 2.7) perusominaisuuksia ohjelmointiin liittyen. Kerron kehitystyökalujen ja -laajennuksien osalta Djangosta, erittäin tehokkaasta webkehitysympäristöstä, jolla pääosin kehitämme lisensointisivua ja siihen liittyviä elementtejä. Esimerkeissä etuliite >>>

tarkoittaa Pythonin komentokehotteeseen kirjoitettavaa käyttäjän syötettä ja ilman tätä etuliitettä oleva teksti on syötteen tuloste.

4.1 Muuttujat ja tietotyypit

Muuttujat ovat yksinkertaisesti selitettynä eräänlaisia säiliöitä, joihin voi sijoittaa muuttuvaa tai muuttumatonta tietoa myöhempää käyttöä varten, joka voi olla tyypiltään erilainen riippuen sen käyttötarkoituksesta. Näitä tyyppejä voivat olla esimerkiksi kokonaisluku (*integer*) ja merkkijono (*string*). Yksinkertaisia muuttujia kirjoitetaan seuraavan esimerkin mukaisesti:

```
>>> kilometers = 500
>>> statement = "shrubbery"
```

Muuttujan arvo asetetaan sille yhtäsuuruusmerkillä, eikä sille kirjoiteta tyyppiä, vaan ohjelmoijan täytyy itse tietää se. Yhtäsuuruusmerkin vasemmalla puolla on muuttujan nimi ja oikealla puolella sen arvo. Kun muuttuja luodaan, sille varataan rekisteristä muistipaikka, johon myös arvo sijoitetaan. Muuttujaa ei tarvitse erikseen kirjoittaa, vaan Pythonin tulkki tekee tämän automaattisesti, kun jonkin arvon sijoittaa muuttujaan. Pythonin kääntäjä päättää datatyypin perusteella sille varatun muistipaikan suuruuden. (TutorialsPoint 2017.)

Python sallii myös saman arvon sijoittamisen useaan eri muuttujaan samanaikaisesti. Myöskään muuttujan tyyppi ei ole pysyvä, vaan sitä voidaan muuttaa asettamalla se uudestaan:

```
>>> x = y = z = 5
>>> print "x: {}, y: {}, z: {}".format(x, y, z)
x: 5, y: 5, z: 5
>>> z = "five"
>>> print "x: {}, y: {}, z: {}".format(x, y, z)
x: 5, y: 5, z: five
```

Muuttujan nimeämiseen käytetään vain tiettyjä sallittuja menetelmiä. Siinä ei saa olla tiettyjä erikoismerkkejä (huutomerkki, kysymysmerkki, piste ym.). Kirjaimet Ä ja Ö eivät saa sisältyä nimeen, ellei ohjelmaan määritä erikseen käytettävää merkistöä. Nimi ei saa alkaa numerolla, mutta saa sisältää sellaisen. Myöskään välilyönti ei ole sallittu, sillä se erottaa komennot toisistaan. Välilyönti tulee korvata joko alaviivalla tai esimerkiksi ”*camelCase*”-tyylillä (Vihavainen, Paksula, Luukkainen, Mikkola, Laurinharju & Pärtel 2017). Muuttuja, jossa on yksi merkkijono, kirjoitetaan seuraavan esimerkin tapaan:

```
>>> firstProgram = "Hello World"
```

Nimen ensimmäisen sanan alkukirjain kirjoitetaan pienenä. Korpelan (2017) mukaan ”nimitystä *camel case* käytetään englannin kielessä sellaisista kirjoitusasuista kuin fontSize ja getElementById. Ajatuksena on, että sanaliitosta (kuten font size ja get element by id) tehdään yksi sana, mutta rakenne osoitetaan aloittamalla muut osat paitsi ensimmäinen versaalilla”.

Nimeämiseen on olemassa muitakin tyylejä, kuten mm. PascalCase, mikä eroaa ainoastaan siinä, että ensimmäinen sana kirjoitetaan isolla alkukirjaimella. Käytän projektissa ”*snake case”*-tyyliä muuttujissa, funktioissa ja metodeissa, jolloin sanat erotetaan toisistaan alaviivalla. Päädyin käyttämään tätä tyyliä, koska lähes kaikki Djangon sisäänrakennetut funktiot ja metodit käyttävät tätä samaa kirjoitustyyliä:

```
>>> file_name = "README.txt"
```

Muistissa olevaan muuttujatietoon sisällytetään aina jokin arvo. Tämä arvo voi olla jokin seuraavista standardoidusta tyypistä:

- merkkijono,
- lukuarvo,
- lista,
- monikko tai
- sanasto

Kerron seuraavissa osissa yksinkertaisia esimerkkejä ja peruspiirteitä edellä mainituista tyypeistä, sekä niiden käyttötarkoituksista.

4.1.1 Merkkijono

Pythonin merkkijonot (engl. *strings*) ovat jatkuva lista kirjaimia tai merkkejä lainausmerkkien sisällä. Merkkijonot muistuttavat hyvin paljon lista-tietotyyppiä. Suurin ero näiden välillä on, että merkkijonoa ei voi muuttaa yhtä lailla kuin listan alkioita. Python sallii merkkijonossa käytettäväksi lainaus- tai heittomerkit, mutta itse käytän jälkimmäistä. Esimerkkinä yksinkertainen merkkijono:

```
>>> greeting = "Welcome, citizen!"
```

Merkkijonoja voidaan manipuloida indeksiviittaustyylin `lista[indeksi]` avulla, jota käytetään listarakenteissa. Ensimmäinen indeksi alkaa nollasta, joten esimerkiksi 5. merkki on tällöin indeksissä 4:

```
>>> print  "The last letter is: " + greeting[15]
The last letter is: n
```

Merkkijonoja voidaan jakaa osiin paloittelemalla (engl*. slice*) käyttämällä operaattoreita [ ], jota käytetään edellisessä esimerkissä, ja `[:]`. Jälkimmäisen operaattorin kaksoispisteen vasemmalle puolelle tulee aloitusindeksi ja oikealle puolelle jonon lopetusindeksi. Kun alkuindeksiä ei määritä, paloittelu alkaa alusta ja vastaavasti, jos loppuindeksiä ei määritä, suoritetaan se loppuun asti. Tulostetaan muuttujan `greeting` ensimmäinen sana:

```
>>> greeting[:7]
Welcome
```

Tulostetaan myös toinen sana:

```
>>> greeting[9:16]
citizen
```

Jälkimmäisen esimerkin merkkijonon pilkkominen alkaa indeksistä 9 ja päättyy indeksiin 16. Viimeinen indeksi ei tulostu, vaan todellisuudessa se päättyy indeksiin 15.

Merkkijonoon voi myös sisältää plus- ja kertomerkkejä, joilla voidaan tehdä merkkijonon toisto- ja yhdistämistehtäviä:

```
>>> string = "This" + " is " + "a string"
>>> string
This is a string

>>> string * 2
"This is a stringThis is a string"
```

Merkkijonoille on myös olemassa omia sisäänrakennettuja funktioita. Esimerkiksi `len()` palauttaa merkkijonon kokonaispituuden ja `upper()` palauttaa sen kaikki merkit isolla:

```
>>> len(greeting)
17

>>> greeting.upper()
WELCOME, CITIZEN!
```

Python tukee myös usean rivin merkkijonoja, jotka muodostetaan sijoittamalla merkkejä kolmen heittomerkin sisään. Rivit päättyvät automaattisesti, mutta ne voi pakottaa käyttämällä kenoviivaa. Esimerkiksi:

```
>>> lorem =  """\
Lorem ipsum dolor sit amet,
consectetur adispiscing elit.
Nunc dictum, ex quis varius euismod
"""
>>> print lorem
Lorem ipsum dolor sit amet,
consectetur adispiscing elit.
Nunc dictum, ex quis varius euismod
```

4.1.2 Lukuarvo

Lukuarvodataksi luetaan Pythonissa kaikki numeroita sisältävät muuttujat. Python tukee 4 eri numerotyyppiä:

- kokonaisluku *int*,
- pitkä kokonaisluku *long*,
- liukuluku *float* ja
- kompleksiluku *complex*

Näiden lisäksi totuusarvomuuttuja *boolean* on kokonaisluvun alityyppi. Myös yksinkertaiset lukuarvot, kuten binääriarvot ja heksadesimaalit palautetaan kokonaislukuina, paitsi jos niiden arvo on liian suuri esitettäväksi ainoastaan kokoanislukuna, jolloin niistä muodostuu pitkiä kokonaislukuja. (Python Software Foundation 2017.)

Lukuarvot ovat muuttumattomia tietotyyppejä. Lukuarvo luodaan Pythonissa joko puhtaasti numeerisesti kirjoitettuna tai sisäänrakennetusti funktion tai jonkin toiminnan tuloksena. Kokonaisluku on toteutettu käyttäen `long` -tyyppiä C-kielessä, joten se on tarkkuudeltaan vähintään 32 bittiä. Pitkä kokonaisluku on tarkkuudeltaan rajaton. Liukuluku on toteutettu `double` -tyypillä, ja se sisältää kokonaisluvun sekä desimaaliosan. (Python Software Foundation 2017.)

```
>>> n = 123  # kokonaisluku
>>> type(n)
int

>>> n = 55.0  # liukuluku
>>> type(n)
float

>>> n = 68431256908658753253  # pitkä kokonaisluku
>>> type(n)
long

>>> n = 24j  # kompleksiluku
>>> type(n)
complex
```

Pythonin tulkki toimii ikään kuin laskimena, koska se sallii laskutoimituksien suorittamisen; kirjoitat sille lausekkeen ja tulkki tulostaa lausekkeen arvon. Samaan tapaan kuin missä tahansa muussa ohjelmointikielessä, lausekkeeseen voi sisällyttää merkit `+`, `-`, `*` ja `/`, ja sulkeilla `( )` voi tehdä ryhmittelyjä. Muutamia yksinkertaisia esimerkkejä:

```
>>> 3 + 3
6
>>> 5 * 6
30
>>> 25 + (5 * 6)
55
>>> 50 / 2 * 3
75
```

Jos lausekkeeseen sisällyttää liukulukuja sekä kokonaislukuja, lausekkeen tulos tulkitaan liukulukuna:

```
>>> 3 * 0.75 / 5
0.45
```

Samaan tapaan kuin merkkijonoja, myös lukuarvoja voidaan kirjoittaa muuttujiin. Muuttujan asettamisen jälkeen tulostusta ei tule ennen kuin tätä muuttujaa käytetään lausekkeessa:

```
>>> x = 5
>>> y = 10 * x
>>> y + x
55
```

4.1.3 Lista

Lista on tietotyyppi, joka sisältää jonkin määrän alkioita tietyssä järjestyksessä. Alkiot voivat muodostua esimerkiksi merkkijonoista tai lukuarvoista. Jokaiselle listassa olevalle alkiolle annetaan järjestyksessä indeksiarvo (alkaen indeksistä 0), mikä mahdollistaa listan alkioiden viittaamisen listan muuttujan nimellä. Tyhjä lista tehdään seuraavasti:

```
>>> list = []
>>> list
[]
```

Lisätään listaan muutamia lukuja ja merkkijonoja:

```
>>> list = [1, "two", "three", 4]
>>> list
[1, "two", "three", 4]
```

Alkioihin voidaan viitata edellä mainittuun tapaan listan muuttujan ja indeksin avulla:

```
>>> list[1]
"two"

>>> list[3]
4

>>> list[-2]
"three"
```

Listan perusoperaatiot, kuten lisäys, toisto ja iterointi ovat tehtävissä hyvin samaan tapaan kuin merkkijonoissa, esimerkiksi:

```
>>> len(list)
4

>>> list + list
[1, "two", "three", 4, 1, "two", "three", 4]

>>> for i in list: print i
1
two
three
4
```

Lista sisältää myös sisäänrakennettuja metodeja sen käsittelyä varten; `pop()` poistaa listan viimeisen arvon, `append()` lisää listan perään halutun arvon sekä `reverse()` kääntää listan toisin päin:

```
>>> list.pop()
4
>>> list
[1, "two", "three"]

>>> list.append(5)
>>> list
[1, "two", "three", 5]

>>> list.reverse()
>>> list
[5, "three", "two", 1]
```

Lista on yksi joustava tietotyyppi erilaisten vaihtelevien arvojen lyhytaikaiseen tallentamiseen. Sen sisältämän tiedon käsittely on yksinkertaista. Listat voivat vaihtaa sisältöä, kasvaa tai kutistua koodin sitä vaatiessa sekä ne voi sisällyttää mihin tahansa objektiin.

4.1.4 Sanasto

Sanasto tai sanakirja (engl. *dict*) on listojen tapaan erittäin joustava tietotyyppi, joka yhdistää avaimen johonkin arvoon. Yksi sanasto-olio koostuu yhdestä tai useammasta avain-arvo-parista. Toisin kuin sekvenssissä tai jaksossa, jotka indeksoidaan numerojoukon perusteella, sanastot indeksoidaan avaimen perusteella, joka voi olla muodoltaan mitä tahansa muuttumatonta tyyppiä. (Python Software Foundation 2017.) Jokainen avain erotetaan arvostaan kaksoispisteellä, ja avain-arvo-pari erotetaan toisistaan pilkulla:

```
>>> dict = {"name": "Mikko", "age": 18}
>>> dict
{"name": "Mikko", "age": 18}
```

Suoraan muuttujaan asettamisen lisäksi Pythonin `dict()`-konstruktori luo sanaston automaattisesti avain-arvo-monikoista:

```
>>> dict([("name", "Mikko"), ("age", 18)])
{"name": "Mikko", "age": 18}
```

Sanaston jokaisen avaimen tulee olla muuttumatonta tyyppiä, joten avaimeksi ei käy toinen sanasto-olio tai lista, eikä sama avain saa toistua kahta kertaa.

Sanasto-olio on muuttuva, ja sen avain-arvo-parien käsittely on yksinkertaista; esimerkiksi avaimen arvoon pääsee käsiksi hakasulkeilla `[]`:

```
>>> dict["name"]
Mikko
>>> dict["age"]
```

```
18
```

Tietoa voidaan päivittää sanastossa edellä esitettyyn tapaan avainviittauksella:

```
>>> dict["name"] = "Janne"
>>> dict["name"]
"Janne"
>>> dict["school"] = "HAMK"
>>> dict
{"age": 18, "name": "Janne", "school": "HAMK"}
```

Edellä esitetyt operaatiot ovat sanaston tärkeimpiä toimintoja, sillä sanastoa käytetään paljon muuttuviin tietotyyppeihin. Avain-arvo-pareja voidaan myös poistaa `del()`-operaatiolla:

```
>>> del dict["school"]
>>> dict
{"age": 18, "name": "Janne"}
```

Sanasto on moneen käyttötarkoitukseen taipuva yksinkertainen työkalu tiedon tallentamiseen, jota käytetään paljon projektimme funktioissa ja niitä kutsuttaessa.

4.1.5 Monikko

Monikko (engl. *tuple*) on lista- ja sanastotyypistä poiketen muuttumaton tietotyyppi, joka sisältää alkioita. Ne muodostavat ryhmän objekteja.

```
>>> tuple = ("cats", "dogs", "cows", "sheep")
```

Monikot ovat suorituskyvyltään nopeampia kuin listat, sillä ne eivät sisällä listan tapaan operaatioita (kuten `sort()` tai `append()`) sen muuttamiseksi, eikä niitä voida paikka-arvoa käyttäen muokata. Niitä voidaan kuitenkin käsitellä samaan tapaan. Monikkoa ei tarvitse erikseen kirjoittaa suluilla, vaan se muodostuu automaattisesti pilkulla erotetuista arvoista, jolloin tulkki pakkaa arvot monikoksi. Tälle toiminnolle on olemassa termi "*tuple packing*". Esimerkkinä yksinkertainen monikko:

```
>>> tuple = "cats", "dogs', "cows", "sheep"
>>> tuple
("cats", "dogs", "cows", "sheep")
```

Monikko voi myös sisältää sisäkkäisiä monikkoja sekä erilaisia tietotyyppejä:

```
>>> many_tuples = [1, 2, 3], tuple, "six", "seven"
>>> many_tuples
([1, 2, 3], ("cats", "dogs", "cows", "sheep"), "six", "seven")
```

Pythonissa lista ja monikko ovat käyttötarkoitukseltaan lähes samanlaisia. Listaan voidaan kuitenkin tallettaa muuttuvaa tietoa ja siinä on enemmän operaatioita tiedon käsittelyä varten. Miksi sitten valita monikko listan sijaan?

Monikon muuttumaton rakenne tarjoaa tiedon kannalta eheyttä, mikä voi olla tietyissä käyttötarkoituksissa parempi vaihtoehto. Mikään muu ulkopuolinen operaatio ei voi muuttaa monikon tietoa, toisin kuin listaa. Hyvänä sääntönä on käyttää listaa, kun halutaan käyttää järjestyksellistä kokoelmaa, joka saattaa muuttua ajan kuluessa, ja monikkoa, kun tieto on pysyvää.

4.2 **Funktiot**

Funktiot ovat keskeinen osa mitä tahansa nykyajan ohjelmointikieltä. Niillä tuodaan ohjelmaan lisää tarvittavaa toiminnallisuutta ja organisoidaan ohjelmakoodia luettavampiin osiin. Sen sijaan, että kopioisimme samaa koodinpätkää uudelleen ja uudelleen, mikä tekisi tiettyä operaatiota useita kertoja, voimme supistaa sen yhteen funktioon. Funktio sisältää uudelleenkäytettävää ja aikaa säästävää koodia, jota voidaan myös ajaa useita kertoja eri vaiheessa ohjelman ajoa.

Funktion rakenne koostuu avainsanasta `def` sekä haluamastaan nimestä. Nimen perään kirjoitetaan kaarisulut, joiden sisään kirjoitetaan haluttuja parametreja, niin monta kuin itse haluaa määritellä. Sulkujen jälkeen aloitetaan sen toiminnallisuuden määrittäminen kaksoispisteen jälkeen.

Python käyttää funktioiden ja lausekkeiden kanssa sisennyksiä, joista ensimmäinen sisentämätön rivi sisältää funktion määrityksen, nimen ja parametrit, ja funktion sisään kirjoitettava sisennetty osa sisältää funktion toiminnallisuuden. Toiminnallisuusosan tulisi sisältää funktion kommenttikentän, jossa kuvataan sen toiminnallisuus, eli on toisille käyttäjille apuna, kun halutaan tietää funktion käyttötarkoitus. Funktion rakenteen jälkeen halutaan pysäyttää funktio ja palauttaa sille jokin arvo avainsanalla `return`. Sille voidaan antaa määritelmä, josta palautettava arvo tulee. Jos tälle ei anneta argumentteja, funktio ei palauta mitään. Se ei myöskään ole pakollinen funktion sisällä, vaan funktio palautuu, kun funktio loppuu. Tällöin funktio käytännössä palauttaa `None` – tyyppisen objektin, mutta se ei näy funktion kutsussa.

Funktiolle on olemassa myös avainsana `yield`, jonka tarkoitus on palauttaa arvoja tietyn ajan kuluessa (Lutz 2013, 763-766). Tätä avainsanaa käytetään *generaattorifunktioissa*, jolloin funktiosta ei palauteta vain yhtä arvoa, vaan `yield` keskeyttää funktion ajamisen, palauttaa yhden arvon takaisin kutsujalle ja säilyttää funktion tilan jatkaakseen siitä, mihin funktion ajaminen keskeytyi. Tämä sallii funktion arvojen palautumisen ajan kuluessa sen sijaan, että kaikki arvot palautuisivat yhdellä kerralla esimerkiksi listaan. (Lutz 2013, 942-945.)

Funktion rakenne muodostuu seuraavasti:

```
>>> def funktion_nimi(parametrit):
...   "funktion kommenttikenttä"
...   funktion_rakenne
...   return [määritelmä]
```

Luodaan yllä olevan rakenteen mukaisesti toimiva funktio, joka palauttaa uuden arvon annetulle argumentille. Funktio ottaa yhden parametrin ja palauttaa sen arvon potenssiin 2:

```
>>> def calc(n):
...     "Palauttaa annetun arvon n neliön"
...     return n ** 2
```

Funktiota voidaan kutsua kirjoittamalla funktion nimen sekä kaarisulkeet, joiden sisälle tulee kirjoittaa funktion argumentti eli arvo tai arvot, joille halutaan jokin uusi palautettava arvo funktiosta. Jos funktion parametreja olisi useita, niille tulisi jokaiselle antaa argumentti funktiota kutsuttaessa. Funktio voidaan kutsua myös toisesta funktiosta. Funktion sisältämä koodi ajetaan vasta kun funktiota kutsutaan:

```
>>> calc(4)
16
```

Python käsittelee kaikki funktion parametrit ja argumentit viittauksina alkuperäisiin muuttujiin. Tämä tarkoittaa, että jos parametria, johon se viittaa, muutetaan funktion sisällä, muuttaa se myös lopputulosta funktiota kutsuttaessa. (TutorialsPoint 2017.)

```
>>> def modify(list):
...     "Muuttaa listan sisältöä"
...     list.append([8, 7, 6, 5])
...     print "Arvot funktion sisällä: {}".format(list)
...     return

>>> list = [0, 2, 4, 6]
>>> modify(list)
>>> print "Arvot funktion ulkopuolella: {}".format(list)

Arvot funktion sisällä: [0, 2, 4, 6, [8, 7, 6, 5]]
Arvot funktion ulkopuolella: [0, 2, 4, 6, [8, 7, 6, 5]]
```

Tämä voidaan todeta luomalla sama funktio, mutta sen sijaan, että funktio muuttaa viittausarvoa niin kuin edellisessä esimerkissä, se luo kokonaan uuden listan paikallisesti:

```
>>> def modify(list):
...     "Muuttaa listan sisältöä"
...     list = [8, 7, 6, 5]
...     print "Arvot funktion sisällä: {}".format(list)
...     return

>>> list = [0, 2, 4, 6]
>>> modify(list)
>>> print "Arvot funktion ulkopuolella: {}".format(list)

Arvot funktion sisällä: [8, 7, 6, 5]
Arvot funktion ulkopuolella: [0, 2, 4, 6]
```

Parametri `list` on funktiolle paikallinen. Sen muuttaminen ei vaikuta funktion ulkopuolella olevaan listaan `list`. Funktio ei näin ollen käytännössä saavuta mitään.

Funktioille on eri tyyppisiä argumentteja. Ne ovat funktiolle järjestyksessä annettavia yksittäisiä arvoja, joiden lukumäärän tulee vastata funktiossa olevaa parametrien määrää. Seuraava esimerkki havainnollistaa pakollista argumenttia:

```
>>> def print_this(word):
...     "Tulostaa funktiolle argumentiksi annetun sanan"
...     print word
...     return
>>> print_this()

Traceback (most recent call last):
 File ”<stdin>”, line 1, in <module>
TypeError: print_this() takes exactly 1 argument (0 given)
```

Syntyy virheilmoitus, jossa ilmoitetaan virheen tyyppi sekä viesti, josta näemme, että funktio vaatii vähintään yhden argumentin, kun sitä halutaan kutsua. Tälle argumentille voidaan antaa myös vakioarvo:

```
>>> def print_this(word=’Hello World’):
...     "Tulostaa funktiolle argumentiksi annetun sanan"
...     print word
...     return
>>> print_this()
Hello World
```

Näiden argumenttimuotojen lisäksi funktion otsikossa oleva parametri muotoa `*nimi` kerää kaikki paikkakohtaiset argumentit monikkoon ja muoto `**nimi` kerää *avainsana-argumentit* sanastoksi (Lutz 2013, 846). Nämä argumenttimuodot ovat suunniteltu funktioille, jotka voivat ottaa vastaan minkä tahansa määrän argumentteja. Molemmat voivat esiintyä funktion määrittelyssä tai sen kutsussa, ja niillä on samoja käyttötarkoituksia. (Lutz 2013, 854.)

Seuraava funktio tarkastelee paikkakohtaisia argumentteja, joilla ei ole vertaisia funktiossa. Funktion parametriksi asetetaan `*` -muotoa oleva arvo:

```
>>> def func(*args):
...     "Tulosta argumentit monikossa"
...     print args
```

Tätä funktiota kutsuttaessa argumentit kerätään uuteen monikkoon, ja tälle annetaan funktion sisällä muuttujanimeksi `args`:

```
>>> func()
()
>>> func(5)
(5,)
>>> func(2, 4, 6, 8)
(2, 4, 6, 8)
```

Avainsana-argumenteille tämä toiminto on sama, mutta argumentit kerätään uuteen sanastoon:

```
>>> def func(**args):
...     "Tulostaa avainsana-argumentit sanastossa"
...     print args
```

Funktio on käytännössä sama kuin aikaisempi, mutta parametrina on `**` -muodossa oleva arvo. Palautettua sanastoa voidaan käsitellä samaan tapaan kuin mitä tahansa sanastoa.

```
>>> func()
{}
>>> func(a=1, b=2)
{"a": 1, "b": 2}
```

Argumenttityyppien yhdistäminen on myös mahdollista. Se on harvinaista, mutta sitä esiintyy funktioissa, jotka ovat esimerkiksi taaksepäin yhteensopivia aikaisempien versioiden kanssa:

```
>>> def func(x, *args, **kwargs):
...     """
...     Tulosta argumentit erikseen
...     omassa annetussa muodossaan
...     """
...     print a, args, kwargs
```

Funktio ottaa ensimmäisen argumentin paikkakohtaisesti, kerää seuraavaksi muut argumentit monikkoon ja muut argumentit tulee olla avainsanoja, jotka kerätään sanastoon:

```
>>> func(1, 2, 3, x=1, y=2)
1, (2, 3), {"y": 2, "x": 1}
```

Funktiot ovat projektimme selkäranka. Niitä esiintyy moduuleissamme jatkuvasti eri muodoissa, ja sekä Pythonissa että Djangossa niitä on sisäänrakennettu toiminnallisuuden lisäämiseksi. Myös eri tietotyypeille on olemassa niiden käsittelyä varten erilaisia funktioita. Funktioiden käyttäminen korostuu projektimme kaltaisessa isommassa kokonaisuudessa, jossa ohjelman luettavuuden, suorituskyvyn ja pitkäikäisyyden merkitys kasvaa.

4.3 **Olio-ohjelmointi Pythonilla**

Pythonin olio-ohjelmointimenetelmä eroaa proseduurillisesta ja funktiopohjaisesta ohjelmoinnista. Se käsittelee luokkia, joiden avulla luodaan rakenteellista ohjelmaa ja tarjoaa käytettäväksi uudenlaisia objekteja.

Luokka luo pohjarakenteen sen tulevaa käyttötarkoitusta varten, ja se sisältää tiedon lisäksi myös toiminnot eli metodit tietoon käsiksi pääsemiseen. Metodit ovat käytännössä vain funktioita sisältäen luokalle ominaisia argumentteja. Metodeihin liittyy myös vahvasti *periytyminen*, jolla tarkoitetaan käytännössä jonkin

luokan koodin muuttamista ja uudelleenkäyttöä. Periytymismekanismin avulla voidaan uudella luokalla periä esimerkiksi jonkin emoluokan ominaisuudet ja metodit.

Avainsanaa `class` käytetään uuden luokan luomiseksi. Esimerkiksi henkilöllä on peruskäsitteistä yhteistä nimi ja ikä. Luodaan emoluokka, joka sisältää kaikille aliluokille ominaisen metodin `getPerson()`:

```
>>> class Person:
...  def __init__(self, name, age):
...             self.name = name
...             self.age = age
...  def getPerson(self):
...             print "Person named {} is
...                    {} years old".format(self.name, self.age)
```

Voimme tehdä aliluokan, joka perii metodin `getPerson()` Person-superluokalta, mutta myös sisältää yhden tälle aliluokalle ominaisen metodin:

```
>>> class Employeee(Person):
...  def __init__(self, name, age, salary):
...             self.name = name
...             self.age = age
...             self.salary = salary
...  def getEmployee(self):
...             print "Person named {}, {}
...                 has a salary of {}".format(self.name,
...                                            self.age,
...                                            self.salary)
```

Näistä luokista voidaan luoda olioita, jotka luodaan luokan määritelmän mukaisesti eli ovat luokan ilmentymiä. Edellä olevissa esimerkeissä luokan metodi `__init__()` on luokan konstruktori, joka määrittelee sen, miten luokan olio luodaan, ja sitä käytetään yleisemmin olion muuttujien arvojen määrittelemiseen.

```
>>> jan = Person("Jan", 24)
>>> jan.name
Jan
>>> jan.age
24
```

Edellä oleva esimerkki luo uuden olion luokasta `Person` annetuilla parametreilla, ja ominaisuudet voidaan tarkistaa esimerkin mukaisesti. Olio voi kutsua suoraan luokan metodia:

```
>>> jan.getPerson()
Person named Jan is 24 years old
```

Työntekijä voidaan luoda samaan tapaan käyttämällä `Employee`-luokkaa:

```
>>> jorma = Employee("Jorma", 45, 35000)
>>> jorma.name
Jorma
... jorma.age
45
```

```
>>> jorma.salary
35000
>>> jorma.getEmployee()
Person named Jorma, 45, has a salary of 35000
```

Koska luokka `Employee` perii `Person`-luokan metodin, voimme kutsua myös sitä juuri luomallani oliolla:

```
>>> jorma.getPerson()
Person named Jorma is 45 years old
```

Voimme näin ollen luoda emoluokan, joka sisältää perusmetodeja, jotka kaikki aliluokat voivat käyttää. Aliluokissa voidaan määritellä kyseiselle luokalle ominaisia metodeja, jotka vain kyseinen aliluokka käyttää.

5 DJANGO

5.1 Historiaa

Django kehittyi työmaailmassa jo käytössä olleista sovelluksista, jotka oli kehittänyt joukko ohjelmoijia Kansasissa, Yhdysvalloissa. Se syntyi syksyllä 2003, jolloin uutistoimisto Lawrence Journal-World:in webohjelmoijat Adrian Holovaty ja Simon Willison alkoivat käyttää Pythonia websovelluskehityksessä. (Holovaty & Kaplan-Moss 2009.)

The World Online -tiimi, joka oli vastuussa muutaman paikallisen uutistoimiston verkkosivujen kehityksestä ja hallinnoimisesta, työskenteli ympäristössä, jossa kehitystyötä ohjasi ja määräsi journalismin deadlinet. Muutos alkoi tapahtua, kun uutistoimiston toimittajat alkoivat vaatia kolmen nettisivun, LJWorld.com, Lawrence.com ja KUSports.com, lisäominaisuuksien ja kokonaan uusien sovellusten kehittämistä päivien, jopa tuntien, varoitusajalla. Tämän vuoksi Holovaty ja Willison kehittivät tiimin tarpeen vuoksi sovelluskehyksen ohjelmointiin kuluvan ajan säästämiseksi. Se oli ainoa tapa, jolla sovellukset saatiin valmiiksi ajoissa. (Holovaty & Kaplan-Moss 2009.)

Kesällä 2005, kun kaikki kehitystiimin verkkosivut olivat enimmäkseen muutettu käyttämään tätä uutta sovelluskehystä, tiimiin liittyi uusi jäsen, Jacob Kaplan-Moss. Kolmikko päätti saman vuoden heinäkuussa julkaista kehittämänsä työkalun lähdekoodin avoimeksi, jonka nimeksi annettiin Django. Se sai nimensä kuuluisan jazzkitaristin, Django Reinhardtin, mukaan. (Holovaty & Kaplan-Moss 2009.)

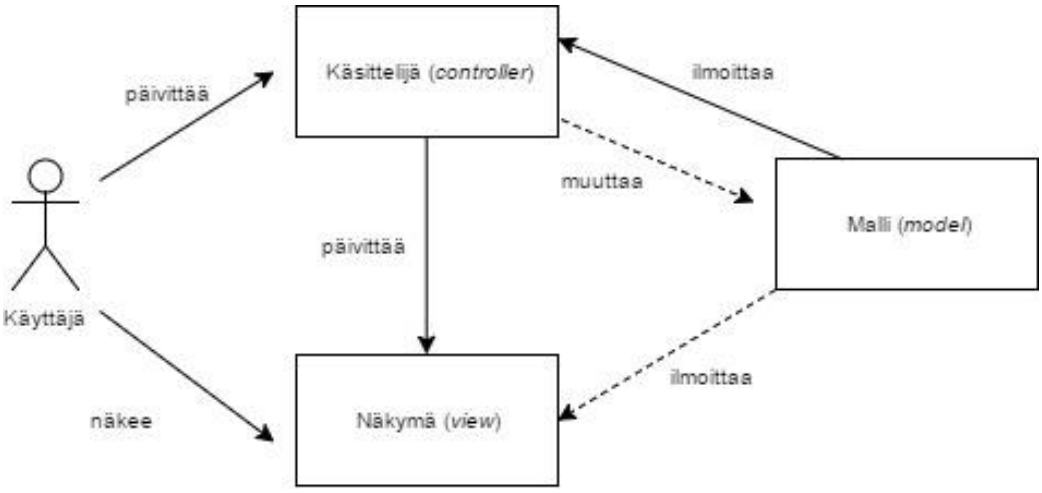
Tänä päivänä Django on menestyvä avoimen lähdekoodin projekti, jota käyttävät ja kehittävät useat kymmenet tuhannet kehittäjät ympäri maailmaa. Djangosta tekee uniikin se, että se kehitettiin työympäristössä, eikä ole esimerkiksi akateeminen projekti tai kaupallinen tuote, ja sen tavoite on ratkaista webkehitykseen liittyviä ongelmia, joita esiintyy nyt ja tulevaisuudessa. Tästä johtuen se kehittyy joka päivä enemmän, ja sen kehittäjillä on yhteinen kiinnostus ja päämäärä pitääkseen Djangon aikaa säästävänä, helppona hallittavana ja suorituskykyisenä. (Holovaty & Kaplan-Moss 2009.)

Koska Django on kehitetty uutistoimistoympäristössä, se soveltuu sisältölähtöisille websivustoille, esimerkiksi Amazon, tai Instagram, jotka tarjoavat dynaamista ja tietokantalähtöistä sisältöä. Tästä huolimatta Django kuitenkin soveltuu hyvin myös muihin dynaamisiin verkkosivuprojekteihin. (Holovaty & Kaplan-Moss 2009.)

5.2 Rakenne ja toiminta

Djangon rakenne pohjautuu löyhästi malli-näkymä-käsittelijä (MVC) -arkkitehtuuriin, jonka tarkoituksena on jakaa sovellus kolmeen, toisiaan vuorovaikutta-

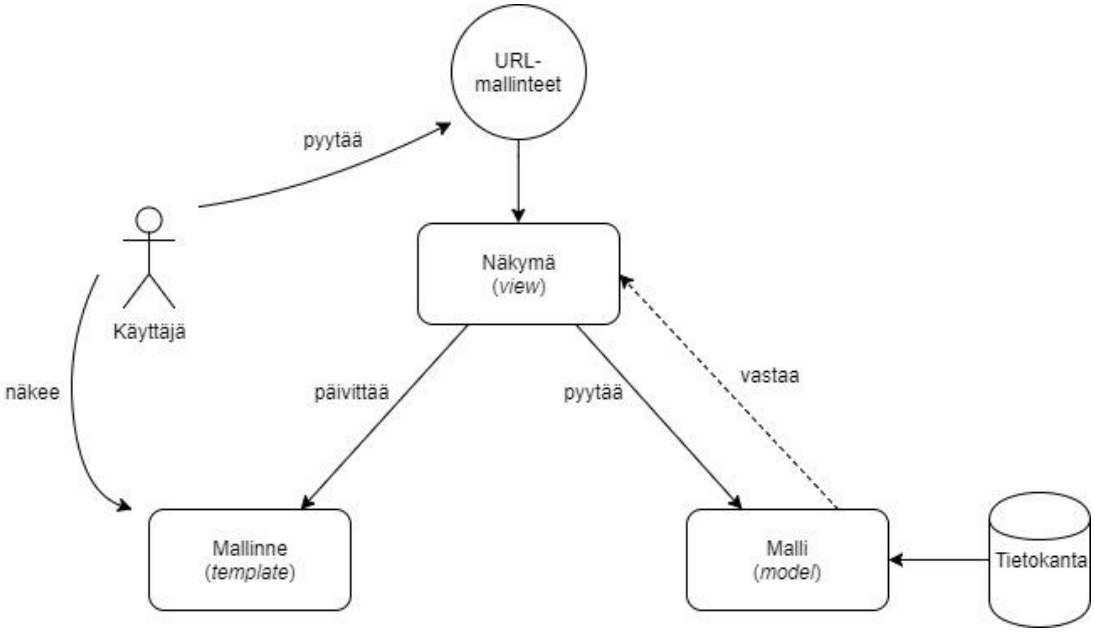
vaan osaan tiedon käsittelyn helpottamiseksi. Malli (*model*) sisältää tiedon rakenteen, tietokantakommunikoinnin sekä sovelluksen logiikan. Näkymät (*views*) tarjoaa loppukäyttäjälle sovelluksen käyttöliittymän käyttäen mallia. Näkymissä voidaan myös muuttaa mallissa olevaa tietoa käsittelijän (*controller*) avulla. Käsittelijä ohjaa käyttäjän pyyntöjä näkymän ja mallin välillä, palauttaen pyyntöä vastaan vastauksen mallin tiedon perusteella. Kuva 5 sisältää MVC-arkkitehtuurin perusrakenteen.



Kuva 5. MVC-arkkitehtuurin rakenne.

Kun käyttäjä tekee päivityspyynnön mallille, käsittelijälle tulee ilmoitus muutoksesta. Tällöin käsittelijä pyrkii vastaamaan pyyntöön päivittämällä näkymän sekä ilmoittamalla mallille muutoksen tilasta. Myös malli voi ilmoittaa tilapäivityksestä näkymälle.

Djangossa tätä arkkitehtuuria tulkitaan MVT-arkkitehtuurina (*model-view-template*). Siinä näkymä edustaa tietokannasta pyydettyä tietoa, tiedon muotoa. Näkymä toimii funktiona URL-linkin kutsulle, jonka avulla haetaan haluttu tieto. Se ei hallitse loppukäyttäjälle näkyvää osaa, vaan tiedon esitystapa luodaan hahmotelmassa tai sivupohjassa (*template*), joka vastaa MVC-arkkitehtuurin näkymää käyttäjälle. Malli (*model*) toimii samalla periaatteella, kuin MVC-arkkitehtuurissa, sisältäen sovelluksen logiikan sekä tietorakenteen tietokantamuodossa. Käsittelijän roolia edustaa tässä tapauksessa sovelluskehys itsessään; se lähettää käyttäjän pyynnön oikeaan näkymään URL-linkin perusteella. Kuva 6 havainnollistaa MVT-arkkitehtuurirakenteen eroja esikuvaansa.



Kuva 6. MVT-arkkitehtuurin rakenne.

MVC- tai MVT- mallit ovat sovelluksen kannalta helppoja ylläpidettäviä, sekä osa-alueet toimivat itsenäisesti; vaikka esimerkiksi mallia muuttaisi, se ei vaikuta käyttäjän näkymään. Tämä tekee sovelluksesta joustavan erillisten komponenttien vuoksi.

5.2.1 Djangon asentaminen

Djangon asentaminen tapahtuu useassa eri vaiheessa. Koska Django on kirjoitettu Pythonilla, tulee se asentaa ensin ja varmistaa, että molempien versiot ovat yhteensopivat. Djangon versio 1.11 LTS, jota oma projektimmekin hyödyntää, on viimeinen versio, jolla on Python 2.7 -tuki. Päädyimme käyttämään Pythonista myös versiota 2.7, koska sitä on helpompi oppia ja useat hyödylliset lisäpaketit tukevat eniten tätä versiota. Tiimimme ei aikaisemmin ole ohjelmoinut työelämässä Pythonilla, pois lukien joitain yksittäisiä skriptejä.

Djangosta tarjotaan ladattavaksi kaksi erityyppistä pakettia: viimeisin virallinen versio sekä kehityksessä oleva testiversio. Valinta on vakaan ja testatun tai epävakaamman, viimeisimmät ominaisuudet tarjoavan version välillä. Tämä valinta riippuu täysin projektin tavoitteesta. Oma projektimme hyödyntää viimeisintä vakaata versiota, jolloin voimme keskittyä projektin tuottamiseen, eikä Djangon vikoihin. On hyvä kuitenkin tietää, että testiversio on olemassa, koska siihen saatetaan viittaa dokumentaatiossa ja yhteisön keskusteluissa.

Djangon asennuksen helpottamiseksi käytämme Pythonin paketinhallintaa, Pipiä. Pip on Pythonin asennuspaketin mukana tuleva lisäosa, joka helpottaa pakettien asentamisen lisäksi niiden versionhallintaa. Djangon asennus tapahtuu Ubuntulla seuraavasti:

```
pip install –upgrade pip
pip install django
```

Ensimmäinen komento päivittää Pipin ja toinen asentaa viimeisimmän Djangon version, tässä tapauksessa version 1.11 LTS. Asennuksen todentaminen tapahtuu Pythonin komentokehotteessa:

```
>>> import django
>>> django.VERSION
(1, 22, 1, u'final', 0)
```

Kun asennus on suoritettu onnistuneesti, voidaan alustaa ensimmäinen projekti. Projekti tarkoittaa kokoelmaa asetuksia yhdelle Djangon instanssille, sisältäen tietokantakonfiguraatiot sekä sovelluskohtaiset asetukset (Holovaty & Kaplan-Moss 2009, 47-48). Ennen projektin luomista kannattaa luoda erillinen hakemisto, jossa se elää. Projektin alustaminen tapahtuu komennolla

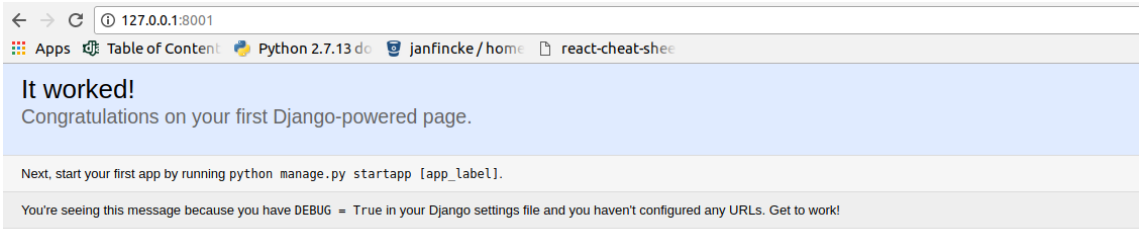
```
django-admin.py startproject site
```

Komento luo hakemistoon sovelluksen perusrakenteen, joka sisältää neljä tiedostoa:

```
site/
     __init__.py
     manage.py
     settings.py
     urls.py
```

`__init__.py` kertoo Pythonilla omasivu-hakemiston olevan paketti, ryhmä Python-moduuleja. Se on tyhjä tiedosto, johon ei lisätä mitään. `Manage.py` on komentorivityökalu, jonka avulla hallitaan juuri luotua projektia. Se sisältää useita hyödyllisiä komentoja projektin ja tietokannan hallintaan. `Settings.py` sisältää sovellukselle ja tälle Djangon instanssille ominaisia asetuksia. `Urls.py` -tiedostossa on kaikki tälle projektille kuuluvat URL-linkit. Se toimii ikään kuin sisällysluettelona projektille, joka yhdistää näkymät ja sivupohjat keskenään (Holovaty & Kaplan-Moss 2009, 48-49).

Projekti sisältää myös Djangoon sisäänrakennetun testipalvelimen, jolla ajetaan testiversiota omasta projektista ja jonka avulla on helppo kehittää sivuja. Käynnistämisen jälkeen se tarkkailee koodia ja päivittää websivua, jolloin työskentely helpottuu, eikä tarvitse keskittyä tuotantopalvelimen hallitsemiseen (Holovaty & Kaplan-Moss 2009, 48-49). Testipalvelin käynnistetään komennolla `python manage.py runserver`. Komento käynnistää testipalvelimen paikallisesti porttiin 8000, johon pääsee käsiksi vain omalta tietokoneelta. Testipalvelinta käytetään ainoastaan paikalliseen testaamiseen, eikä sisällä tuotantoympäristön kaltaista tietoturvaa ja se pystyy käsittelemään vain yksinkertaisia pyyntöjä. Kuvassa 7 on esitetty testipalvelimen ensimmäisen onnistuneen käynnistyksen tulos.



Kuva 7. Djangon testipalvelimen ensikäynnistys.

Jotta saamme projektin täysin alustettua, tulee tälle projektille luoda vielä sovellus, joka on Djangossa erillinen käsite projektin sisäiselle koodille. Djangon *app* eli sovellus edustaa liikkuvaa kokoelmaa koodia, joka sisältää näkymät, mallit ja sivupohjat. Ne elävät omassa paketissa projektin sisällä ja ovat yhteydessä Django-projektin asetuksiin ja tietokantaan. Tämä sovellusrakenne helpottaa isojen projektien jakoa pienempiin palasiin antamalla samalla mahdollisuuden sovellusten uudelleenkäyttämiseen. Yksinkertaisiin websivuihin riittää yksi Djangon sovellus. Sovellusten luominen projektin sisälle on pakollista ainoastaan, kun käytössä on jokin tietokantarakenne. Tällöin myös mallien on sijaittava sovelluksen sisällä. Sovellusten luonti on siis projektin suunnittelun kannalta merkittävää.

Uusi Django-sovellus luodaan projektihakemistoon esimerkin mukaisesti komennolla `python manage.py startapp books`. Projektin sisälle syntyy uusi hakemisto, johon sovelluksen tiedostot on kopioitu. Rakenne muuttuu seuraavasti:

```
site/
     movies/
                __init__.py
                migrations/
                           __init__.py
                models.py
                tests.py
                views.py
                admin.py
                apps.py
     __init__.py
     manage.py
     settings.py
     urls.py
```

Uudet tiedostot ovat muutamia kommenttirivejä lukuun ottamatta tyhjiä. `Models.py` tulee sisältämään sovelluksen mallit ja `views.py` sovelluksen näkymämäärittelyt. `Tests.py` -tiedostoon voidaan kirjoittaa sovellukselle ominaisia kooditestejä. `Admin.py` sisältää hallinnointisivulle tulevien mallikohtaisten näkymien ominaisuuksien asetuksia ja `apps.py` sisältää sovelluskohtaisten asetusten metatiedon `AppConfig`-luokan instanssissa. Sovellus on tästä eteenpäin valmis kehitettäväksi ja jalostettavaksi.

5.2.2 Mallit

Nykyajan verkkosivut ovat monimutkaisia, ja vaativat usein ohjelmalogiikan ja tietokannan välistä vuorovaikutusta. Tällaisten verkkosivujen taustalla toimii jokin tietokanta, josta etsitään tietoa ja esitetään se käyttäjälle jossain muodossa. Tästä on hyvänä esimerkkinä *Amazon.com*, jossa näkymät hoitavat tietokantapyynnöt eri tuotteille. Djangon rakenne on suunniteltu Amazonin kaltaisten tietokantapohjaisten verkkosivujen ja -palveluiden tuottamiseen, koska se tarjoaa tietokantaohjelmoimista varten mallitason, jossa yksi malli luodaan vastaamaan yhtä tietokannassa olevaa taulua. Malli määrittää tietokantaan yksittäisen tietorakenteen sisältäen tiedolle ominaiset kentät ja toiminnallisuuden.

Mallin luominen aloitetaan luomalla tai käyttämällä jo valmiiksi olemassa olevaa tietokantaa. Django tukee useita erityyppisiä tietokantoja, kuten MySQL, PostgreSQL, SQLite tai Oracle. Palvelimelle tulee asentaa adapteri tietokantaa varten, joka asennetaan Pythonin pakettihallinnan kautta. Käytimme projektissa PostgreSQL-tietokantaa, josta kerron myöhemmässä luvussa. Käytän tämän luvun esimerkeissä Djangon oletuskantaa SQLite3.

Sovelluksen mallit luodaan Django-sovelluksen alustamisen jälkeen `models.py` -nimiseen tiedostoon. Kaikki mallit ovat `django.db.models.Model` -luokan aliluokkia. Niiden määrittely muistuttaa jonkin verran Pythonin luokkamäärittelyä. Yksinkertaisimmillaan luokka voidaan toteuttaa seuraavan esimerkin mukaisesti:

```
from django.db.models import Model

class Movie(Model):
     title = models.CharField(max_length=50)
     director = models.CharField(max_length=50)
     publishing_date = models.DateField()
     country = models.CharField(max_length=50)
```

Tämä määrittely luo SQLite3 -tietokannassa vastaavan taulun kyselyllä

```
CREATE TABLE movies_movie (
     "id" serial NOT NULL PRIMARY KEY
     "title" varchar(50) NOT NULL
     "director" varchar(50) NOT NULL
     "publishing_date" date NOT NULL
     "country" varchar(50) NOT NULL
);
```

Djangolle tulee vielä kertoa käyttää juuri tehtyä mallia. Asetustiedostossa `settings.py` on asetus `INSTALLED_APPS`, joka sisältää kaikki tälle Djangon instanssille ominaiset sovellukset, jotka se lataa käyttöönsä. Sen oletusarvot ovat määritetty seuraavasti:

```
INSTALLED_APPS = [
     "django.contrib.auth",
     "django.contrib.sites",
     "django.contrib.contenttypes",
     "django.contrib.sessions",
```

```
    "django.contrib.messages",
    "django.contrib.staticfiles",
]
```

Tähän asetukseen lisätään kaikki ne projektin sovellukset, joiden malleja halutaan hyödyntää. Sovellukset tässä listassa ovat monikkoja, joten ne tulee lopettaa pilkulla, vaikka sovelluksia olisikin vain yksi. Lisätään sovellus asetuksen viimeiselle riville:

```
INSTALLED_APPS = [
    ...
    "movies",
]
```

Sovelluksen lisäämisen jälkeen synkronoidaan tietokanta, jotta mallit tulevat siellä näkymään omina tauluinaan. Synkronointi tapahtuu migraatiotiedostojen avulla, jotka tallentavat tietokannan tilatiedot sisältäen mallit sekä esimerkiksi niiden relaatiot muiden mallien ja sovellusten kesken. Komento `python manage.py migrate` tarkistaa ja toteuttaa uudet migraatiot. Jos syntyy uusi malli, käytetään komentoa `python manage.py makemigrations`, jotta uusista muutoksista syntyy migraatiotiedosto.

Django tarjoaa sisäänrakennetun QuerySet API-sovelluksen mallien käyttöä ja hallintaa varten. Tätä ominaisuutta voidaan helposti käyttää komentokehotteesta. Luodaan kirjamallista muutama uusi objekti ja listataan ne:

```
>>> from movies.models import Movie
>>> m1 = Movie(title="The Godfather", director="Francis Coppola",
publishing_date="1972-09-29", country="USA")
>>> m1.save()
>>> m2 = Movie(title="Fight Club", director="David Fincher",
publishing_date="1999-11-12", country="USA")
>>> m2.save()
>>> movie_list = Movie.objects.all()
>>> movie_list
<QuerySet [<Movie: Movie object>, <Movie: Movie object>]>
```

Uusi objekti luodaan muuttujaan asettamalla sille attribuutit, jonka jälkeen kutsutaan `save()` -metodia, jotta objekti tallentuu tietokantaan. Ilman sitä tieto ei tallennu. Edellisessä esimerkissä luotiin tähän tapaan kaksi eri objektia ja listattiin ne. Listauksesta ei käy vielä ilmi, kumpi elokuva on kumpi. Muutetaan mallin esitystapaa sen määrittelyn sisällä niin, että elokuvan nimi näkyy kyselyjen tuloksista. Muokataan mallia lisäämällä siihen metodi `__unicode__()`, joka palauttaa objektin nimen, joka sille on asetettu:

```
class Movie(Model):
    ...
    def __unicode__(self):
            return self.title
```

Metodin toiminnallisuus voidaan testata tekemällä uusi kysely komentokehotteesta:

```
>>> from movies.models import Movie
```

```
>>> Movie.objects.all()
<QuerySet [<Movie: The Godfather>, <Movie: Fight Club>]>
```

Objekteja voidaan myös suodattaa tietyn kriteerin perusteella. Metodi `filter()` ottaa avainsana-argumentteja ja muuttaa ne SQL:ää vastaaviksi `WHERE`-lausekkeiksi (Holovaty & Kaplan-Moss 2009, 116-117). Voimme suodattaa elokuvat nimen perusteella seuraavasti:

```
>>> from movies.models import Movie
>>> Movie.objects.filter(title="The Godfather")
<QuerySet [<Movie: The Godfather>]>
```

, joka vastaa SQL-lauseketta:

```
SELECT id, title, director, publishing_year, country
FROM movies_movie
WHERE name = "The Godfather";
```

`Filter()`-metodiin voidaan myös sijoittaa useita argumentteja, jotka muunnetaan eteenpäin SQL:n `AND`-lausekkeiksi.

```
>>> Movie.objects.filter(title="Fight Club", country="USA")
<QuerySet [<Movie: Fight Club>]>
```

Kun kysely ajetaan, SQL käyttää yhteneväisyyksien löytämiseen yhtäsuuruusmerkkiä vakioarvona. Tätä voidaan muokata ja tehdä erilaisia, monimutkaisempia kyselyjä. Esimerkiksi elokuvan nimen voi hakea jollain merkkijonolla käyttämällä `__contains` -hakua sen nimessä. SQL tulkitsee nämä lausekkeet `LIKE`-lausekkeina:

```
>>> Movie.objects.filter(title__contains="father")
<QuerySet [<Movie: The Godfather>]>
```

Tiettyinä hetkinä haluamme hakea listan, tai Djangon tapauksessa QuerySet:in, sijaan vain yhtä objektia. Tämä onnistuu käyttämällä `get()`-metodia:

```
>>> Movie.objects.get(name="Fight Club")
<Movie: Fight Club>
```

Koska `get()` hakee vain yhtä objektia, se asettaa poikkeuksen, jos se löytää yhdelle kriteerille monta eri objektia.

```
>>> Movie.objects.get(country="USA")
Traceback (most recent call last):
...
MultipleObjectsReturned: get() returned more than one Movie – it
returned 2! Lookup parameters were {"country": "USA"}
```

Myös kysely, josta ei löydy objektia, asettaa poikkeuksen:

```
>>> Movie.objects.get(name="Titanic")
Traceback (most recent call last):
...
DoesNotExist: Movie matching query does not exist.
```

Poikkeus `DoesNotExist` on mallin luokan attribuutti. Tämän kaltaiset poikkeukset hallitaan yleensä `try/except/finally` -lausekkeiden avulla.

Kyselyiden tieto palautuu tietokannasta aina mielivaltaisesti, järjestämättömänä. Tähän voidaan vaikuttaa metodilla `order_by()`, joka järjestää objektit halutun arvon perusteella, esimerkiksi aakkosjärjestykseen:

```
>>> Movie.objects.order_by("title")
<QuerySet [<Movie: Fight Club>, <Movie: The Godfather>]>
```

tai käänteiseen aakkosjärjestykseen käyttäen argumentin edessä miinusmerkkiä:

```
>>> Movie.objects.order_by("-title")
<QuerySet [<Movie: The Godfather>, <Movie: Fight Club>]>
```

Vaikka `order_by()` on yksittäisiin kyselyihin erinomainen vaihtoehto, sen kirjoittaminen joka kerta saattaa olla työlästä. Taulujen järjestyksen voi määrittää mallin sisällä `Meta`-nimisellä luokalla, jolla voidaan tehdä mallikohtaisia asetuksia. Käyttämällä `Meta`-luokassa asetusta `ordering` saamme taulut järjestettyä heti kyselyn yhteydessä halutun arvon perusteella, jos niille ei erikseen ole kerrottu `order_by()`-metodia käyttäen järjestystä. (Holovaty & Kaplan-Moss 2009, 119-120).

```
class Movie(Model)
...
     class Meta:
                ordering = ["title"]
```

Myös metodien ketjuttaminen kyselyissä on mahdollista, kun halutaan suodattaa tietoa ja samanaikaisesti järjestää se:

```
>>> Movie.objects.filter(country="USA").order_by("title")
<QuerySet [<Movie: Fight Club>, <Movie: The Godfather>]>
```

Tietoa voidaan poistaa tietokannasta käyttämällä objektin `delete()`-metodia.

```
>>> m = Movie.objects.get(title="Fight Club")
>>> m.delete()
>>> Movie.objects.all()
<QuerySet [<Movie: The Godfather>]>
```

Usean objektin poistaminen voidaan tehdä sisällyttämällä poistokutsuun objektin `filter()`-metodi, jolloin tietokannasta poistuu kaikki sitä argumenttia vastaavat nimikkeet:

```
>>> Movie.objects.filter(country="USA").delete()
>>> Movie.objects.all()
<QuerySet []>   # tyhjä, koska kaikkien elokuvien maa on "USA"
```

5.2.3 Näkymät

Malleja ja niiden ominaisuuksia voidaan tuoda käytettäväksi tietokannasta Djangon näkymien (*views*) avulla. Näkymä tekee tietokantaan kyselyn annetuilla kriteereillä, ja käsittelee tiedon niin, että siitä muodostuu verkkosivun käyttäjälle haluttu lopputulos. Yksinkertaisimmillaan näkymä voidaan tehdä funktioiden avulla. Näkymiin voidaan suoraan tuoda viittauksia tietokannasta käyttämällä Djangon QuerySet API-rajapintaa, joka esitettiin aikaisemmassa luvussa. Avataan tiedosto `views.py` ja kirjoitetaan funktio `movie()`:

```
from django.http import HttpResponse
from models import Movie
...

def movie(request):
     return HttpResponse(Movie.objects.get(id=1)
```

Funktiossa määritettävä QuerySet-tietokantahaku etsii objektin, jonka ID on arvoltaan 1, ja funktio palauttaa elokuvan nimen, koska objektin esitystapa määritettiin sen `__unicode__`-metodissa palauttamaan sen nimi.

Django prosessoi URL-pyynnöt katsomalla `ROOT_URLCONF`-nimistä asetusta, josta se löytää tarvittavat sovelluksen URL-määritykset. Django hakee asetuksessa määritetyn sovelluksen `urls.py`-tiedoston, josta se etsii ensimmäisen pyyntöä vastaavan määrityksen ja näkymäfunktion. Näkymäfunktio ajetaan ja käyttäjälle muodostuu selaimeen pyyntöä vastaava näkymä. Näkymälle tulee näin ollen ensiksi asettaa URL-linkki, joka tehdään projektin `urls.py` -nimisessä tiedostossa:

```
...
from movies import views

urlpatterns = [
...
url(r'^movie/$', views.movie),
]
```

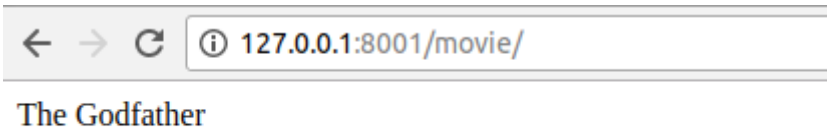
Näkymä tuodaan sovelluksen näkymämoduulista Pythonin `import` -lausekkeella, joka sijoitetaan `urlpatterns`-monikkoon. Ensimmäinen `url()`-funktion argumentti on URL-kaavain, joka kertoo linkin polun ja toinen argumentti on näkymämoduulissa sijaitsevan funktion nimi.

URL-kaavaimen kirjoitusasuun liittyy säännöllinen lauseke, joka sisältää linkin merkkijonomuodossa, mutta sen erikoismerkit tekevät siitä mielenkiintoisemman. Siihen sisältyy potenssimerkki (`^`) ja taalamerkki (`$`); potenssimerkki tarkoittaa, että kaavaimen tulee päteä merkkijonon alussa, ja vastaavasti dollarimerkki tarkoittaa, että sen tulee päteä merkkijonon lopussa. Jos käyttäisin edellisen esimerkin tapaan samanlaista merkkijonoa, jossa ei olisi taalamerkkiä, mikä tahansa linkki *alkaen* merkkijonolla `/hello/` täsmää kaavaimeen, esimerkiksi `hello/foo/bar`. Jos jättäisin pois potenssimerkin, mikä tahansa linkki *päättyen* sanaan `hello/` täsmää kaavaimeen, esimerkiksi `foo/bar/hello/`. Näillä merkeillä siis rajoitetaan linkin riippuvuutta näkymään; edellisen esimerkin kaavaimella rajoitan funktion toimivuuden vain ja ainoastaan linkkiin `/hello/`.

Kaavaimet loppuvat tyypillisesti kyseisiin erikoismerkkeihin, mutta niiden jättämisellä pois lisätään linkkien joustavuutta ja syvyyttä. (Holovaty & Kaplan-Moss 2009, 53-54.)

Erikoismerkkien lisäksi Django käsittelee merkkijonot jättäen pois ensimmäisen vinoviivan, näin ollen URL-kaavaimeen kirjoitetaan haluttu linkki ilman kyseistä vinoviivaa, niin kuin aikaisemmassa esimerkissä. Tämä vaikuttaa myös linkin pyyntöön selaimessa. Jos selaimeen kirjoittaa kaavaimen merkkijonosta eroten linkin *ilman* viimeistä vinoviivaa, Django tulkitsee, että kaavain ei täsmää. Tällöin linkki uudelleenohjataan vakioarvoon, joka sisältää vinoviivan. (Holovaty & Kaplan-Moss 2009, 57-58.)

Jos testipalvelin on päällä, se päivittyy automaattisesti, kun sovelluksen tiedostoja tallentaa, näyttäen samalla mahdolliset virheet koodissa. Jos näkymän kutsu onnistuu, kuva 8 havainnollistaa onnistunutta lopputulosta.



Kuva 8. Movie-näkymä.

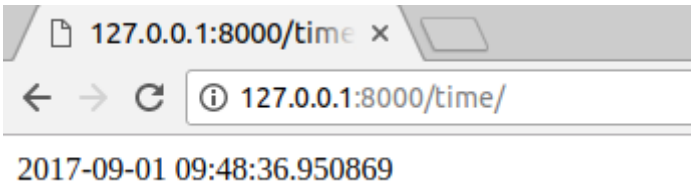
Sivulla ei tällä hetkellä ole muuttuvaa sisältöä, vaan tieto viittaa tietokannan tauluun. Tämä lähestymistapa ei ole aina hyväksyttävää, vaan sivuille halutaan luoda dynaamisesti päivittyviä sivuja, jonka tieto päivittyy, kun sivun lataa uudelleen. Funktion avulla voidaan luoda näkymä, jossa on itsestään päivittyvä digitaalinen kello.

```
...
from datetime import datetime

def current_time(request):
     now = datetime.now()
     html = "<html><body>{}</body></html>".format(now)
     return HttpResponse(html)
```

Muuttujaan `now` sidotaan `datetime` -objektin `now()` -metodin palautusarvo, joka kertoo tämänhetkisen paikallisen kellonajan ja päivämäärän. Tästä tiedosta tehdään HTML-rakenne, jossa käytetään kyseistä aikatietoa. Muuttuja `html` annetaan `HttpResponse` -konstruktorin argumentiksi, joka lähettää tiedon näkymälle, ja tietoa voidaan tarkastella testipalvelimen avulla. Tätä ennen tulee luoda funktiolle URL-kaavain näkymän käsiksi pääsemiseen.

```
urlpatterns = [
...
url(r'^time/$', views.current_time),
]
```

Sivun voi todeta toimivaksi avaamalla testipalvelin komennolla `python manage.py runserver` ja avataan sivu osoitteessa 127.0.0.1:8000. Kuva 9 havainnollistaa toimivaa näkymää.



Kuva 9. Dynaaminen aikatietonäkymä.

Sivu näyttää nykyisen päivämäärän oletusformaatissa ja kellonajan millisekunnin tarkkuudella. Funktiossa on myös ”kovakoodattua” HTML-sisältöä, joka on edellisten esimerkkien kannalta hyvä vaihtoehto, mutta ei todellisuudessa ole hyväksi seuraavia asioita ajatellen:

- Jokainen sivun muutos sen muotoilun kannalta vaatisi funktion muuttamista. Websivun rakenne muuttuu enemmän kuin sen taustalla ajettava Python-koodi, joten rakennetta tulisi pystyä muuttamaan ilman taustaja lähdekoodin muuttamista. (Holovaty & Kaplan-Moss 2009, 67-68.)
- Ohjelmointi Pythonilla ja HTML:llä ovat kaksi eri käsitettä, ja ne tulee myös pitää sellaisenaan. Websuunnittelijoiden ei tulisi joutua koskemaan Python-koodiin. (Holovaty & Kaplan-Moss 2009, 67-68.)
- HTML-sisältöä ja ohjelmakoodia tulisi pystyä tuottamaan samanaikaisesti. On myös tuottamisen kannalta tehokkaampaa, että ohjelmoijat eivät joutuisi vuoron perään muokkaamaan samaa tiedostoa. (Holovaty & Kaplan-Moss 2009, 67-68.)
- Tiedostojen hallinta helpottuu, kun ohjelmakoodin ja HTML-sisällön jakaa erilleen.

5.2.4 Sivumallinteet

Django tarjoaa näkymän käsittelemän tiedon esittämiseen sivumallinteet (*templates*). Mallinne voi olla mitä tahansa tekstipohjaista formaattia, kuten HTML tai XML. Mallinteiden avulla tekstisisältö hajautetaan näkymistä, ja niiden tuottaminen ja hallinnointi helpottuvat merkittävästi. Mallinteiden avulla verkkosivun tiedon esitystapaa voidaan muuttaa rakenteellisesti ilman, että taustalla ajettavaan ohjelmakoodiin tarvitsee koskea. Mallinteet ovat näin ollen erillinen käsite Django-ympäristössä, ja ne antavat projektille muokattavuutta.

Projektissamme mallinteet ovat HTML-tiedostoja, jotka luodaan samalla periaatteella kuin perinteiset HTML-sivut, mutta ne sisältävät muuttujia, jotka edustavat jotain arvoa suoraan tietokantakyselystä tai jostain muusta koodin puolella käytetystä arvosta. Muuttuja muodostetaan seuraavalla tavalla:

```
{{ muuttujan_nimi }}
```

Kun koodi ajetaan ja jokin vastaava muuttuja prosessoidaan, Django etsii näkymän kontekstista vastaavan muuttujan, evaluoi sen ja asettaa muuttujan paikalle sen arvon. Pisteellä voidaan erotella muuttujasta attribuutteja. Esimerkiksi elokuvan nimi on elokuvamallin attribuutti:

```
{{ movie.title }}
```

Tällöin kyseisen muuttujan tilalle tulee arvoksi tietokannasta kysellyn elokuvan nimi.

Yksinkertaisimmillaan mallineita voidaan luoda Pythonin komentokehotteessa ilman tekstitiedostojen luomista. Mallinne tarvitsee `Template`-luokan olion `t`, jonka konstruktoriin asetetaan merkkijonomuodossa mallinteen rakenne, sekä `Context`-luokan olion `c`, jonka konstruktoriin asetetaan muuttujat ja niille jokin arvo:

```
>>> from django.template import Template, Context
>>> t = Template("{{ name }} is {{ age }} years old.")
>>> c = Context({"name": "Jan", "age": 24})
```

`Template`-luokka sisältää `render()` -metodin, jonka avulla mallinne voidaan renderöidä eli tulostaa antamalla sille argumentiksi muuttujat kontekstista.:

```
>>> t.render(c)
u'Jan is 24 years old.'
```

Django käsittelee koko ympäristössä kaikki merkkijonot Unicode-tyyppisinä objekteina, jotka tunnistetaan merkkijonon edessä olevasta `u` -merkistä. Tämä käsittely auttaa huomattavasti, jos projekteissa joutuu käsittelemään erikoismerkkejä, joita ei löydy englannin kielestä.

Monirivisellä merkkijonolla voidaan muodostaa Pythoniin hahmotelma HTML-mallineesta. HTML-mallinteet luodaan yleensä omiin tiedostoihin, mutta esimerkin valossa luodaan uusi merkkijonotyyppinen muuttuja ja asetetaan siihen raaka versio sivusta:

```
>>> html_template = """<p>Hei {{ person }},</p>
...
... <p>Kiitos tuotteen {{ product_name }} tilauksesta!
...    Tuotteen lähetyspäivämääräksi
...    arvioidaan {{ ship_date|date:'d.m.Y' }}.</p>
...
... {% if warranty %}
...    <p>Takuukuitti sekä -tiedot löytyvät pakkauksesta.</p>
... {% else %}
```

```
...    <p>Tälle tuotteelle ei ole takuuta
...    tai et tilannut tuotteelle takuuta.</p>
... {% endif %}
... <p>Ystävällisin terveisin,<br>{{ company }}.</p>"""
```

Muuttujiin voidaan asettaa suodattimia | -merkin jälkeen. Asetin päivämäärämuuttujaan suodattimen, jolla muotoilen päivämäärän eurooppalaiseen muotoon, sillä `datetime.date` palauttaa oletusarvona päiväyksen amerikkalaisessa muodossa. Jotta mallinne saadaan käyttökelpoiseksi, `html_template` -muuttuja asetetaan `Template`-olion `t` konstruktorin argumentiksi, jolloin siitä muodostuu mallinne, joka edustaa HTML-sivua:

```
>>> t = Template(html_template)
```

`Context`-olion `c` konstruktoriin asetetaan kaikille mallineessa oleville muuttujille arvot:

```
>>> c = Context(
...     {
...      "person": "Jan",
...      "product_name": "Sony Playstation 4",
...      "ship_date": datetime.date(2018, 4, 10),
...      "warranty": False,
...      "company": "ScamYou Ltd."
...     }
... )
```

Lopuksi mallinne voidaan renderöidä:

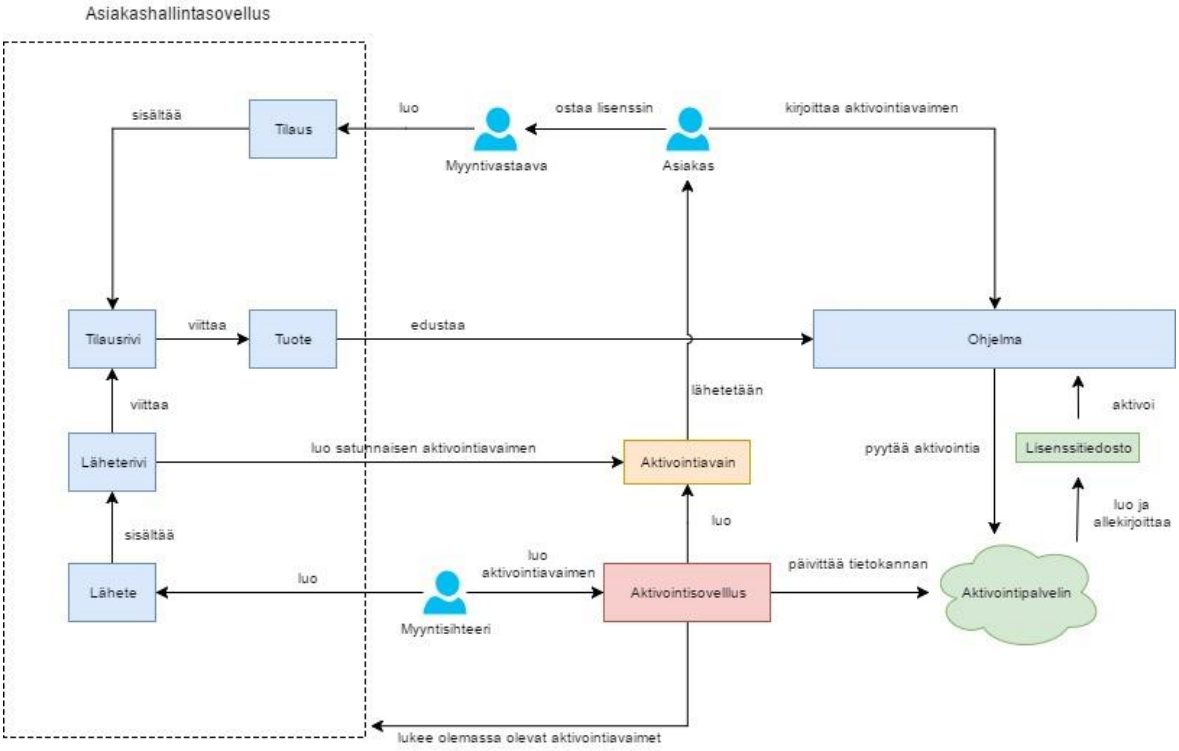
```
>>> t.render(c)
... u'<p>Kiitos tuotteen Sony Playstation 4 tilauksesta!
... Tuotteen lähetyspäivämääräksi
... arvioidaan 10.4.2018.</p>
... \n\n\n<p>Tälle tuotteelle ei ole takuuta
... tai et tilannut tuotteelle takuuta.</p>\n\n\n
... <p>Ystävällisin terveisin,<br>ScamYou Ltd.</p>'
```

Projektin puolella mallinteet luodaan hyvin samaan tapaan kuin näissä esimerkeissä. Helppoa ylläpitoa varten mallinteet ja näkymämäärittelyt on jaettu projektissa eri tiedostoihin.

6 PROJEKTIN TOTEUTUS

6.1 Sovelluksen käyttötarkoitus

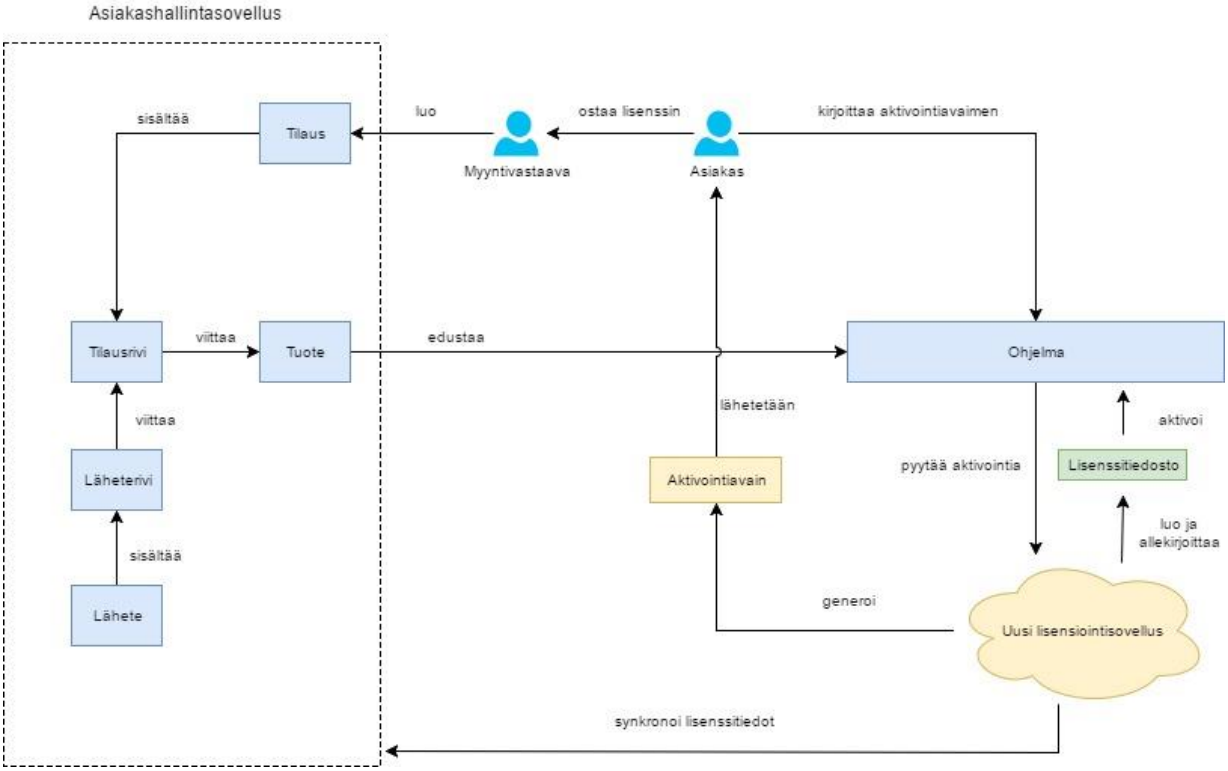
Sovelluksen tarkoituksena on suoraviivaistaa ja automatisoida CadWorks Oy:n tuotteiden lisenssien toimittamista asiakkaalle. Lisenssin aktivointi on tätä projektia ennen vaatinut manuaalista lisenssiavaimen luontia käyttämällä aktivointisovellusta, joka kommunikoi asiakashallintaohjelmiston kanssa. Kuva 10 esittää nykyistä aktivointiprosessia.



Kuva 10. Nykyinen lisenssin aktivointiprosessi.

Asiakkaan ostaessa lisenssin, myyntivastaava luo tästä uuden tilauskentän. Tähän luodaan vastaavasti tilauksen lähete, jonka läheteriviltä luodaan aktivointiavaimelle asiakashallintasovelluksen ominainen avain. Myyjän tulee tämän lisäksi manuaalisesti luoda aktivointisovellukselle oma avain, jonka avulla päivitetään uusi lisenssi tietokantaan sekä päivitetään asiakashallintatietokanta. Aktivointisovellus generoi lopullisen asiakkaalle toimitettavan aktivointiavaimen. Oleellinen ongelma on ylimääräisen sisäisen ”aktivointiavaimen” generoituminen, joka ei varsinaisesti liity muuhun, kuin aktivointisovelluksen käyttöön ja hidastaa prosessia turhaan.

Uusi lisensiointiprosessi automatisoi asiakkaalle toimitettavan aktivointiavaimen luomisen sekä nopeuttaa toimitusprosessia. Kuva 11 esittää yksinkertaisempaa prosessia.



Kuva 11. Uusi lisensiointiprosessi.

Myyjä käyttää uutta sovellusta aktivointiavaimen toimittamiseen. Sovellus huolehtii oman tietokannan, vanhan lisensiointipalvelimen ja asiakashallinnan kanssa synkronoinnin sekä tuotteiden kanssa kommunikoimisen. Näin ollen kyseinen prosessi tehostuu ja samalla helpottaa myymisprosessia sekä parantaa asiakaskokemusta- ja tyytyväisyyttä.

Lisensiointisovellukseen rakennetulla websivustolla voidaan hallinnoida asiakkaiden AutomateWorks© ja CustomWorks© -lisenssejä. Sovelluksen avulla asiakas voi selata käyttöoikeuksista riippuen oman organisaationsa käyttäjä-, työasema-, ja aktivointiavaintietoja sekä lisenssitietoja, joihin kuuluu tiedot aktivoinnin ja ylläpitosopimuksen tilasta. Lisäksi asiakas näkee, mille työasemalle lisenssi on aktivoitu ja kenelle se kuuluu. Asiakas näkee myös lisenssin aktivointitapahtumat eli toteutumistaulun, josta näkee onnistuneet aktivointipyynnöt ja niiden tilan.

Sovellukseen on rakennettu yksinkertainen REST API-rajapinta asiakkaiden käyttöön, jolla voidaan listata käyttäjän saatavilla olevia lisenssejä, aktivoida tai vapauttaa jo aktivoituja lisenssejä sekä tarkastella aktivoinnin aikana generoituvaa lisenssitiedoston tilaa.

6.2 **Versionhallinta**

6.2.1 Git

Projektin kannalta on erittäin tärkeää pysyä mukana tiiminsä työskentelystä ja ohjelmakoodin tilasta sekä pystyä korjaamaan ja päivittämään koodia hajautetusti. Otimme ensimmäistä kertaa käyttöön Git-versionhallinnan, joka mahdollistaa kooditiedostojen muutosten seuraamisen ja näiden kooditiedostojen hajautetun muokkaamisen.

Ohjelmakoodi kopioidaan aluksi tietolähteeseen (*repository)*, josta kopioidaan paikalliselle työasemalle kopio. Tietolähteessä koodia muokataan haaroja (*branch)*, joiden tarkoituksena on jakaa ohjelman kehitystä erillisiin, suljettuihin ympäristöihin. Eri haaroista muodostuu näin ollen puurakenne jotka projektissamme jakautuu kolmeen pääoksaan, testi-, alustus- ja tuotantohaaraan (*dev, staging* ja *master)*. Kehitystyö jakaantuu viikoittaisiin sprintteihin, jolloin testihaarasta luodaan sprinttihaara, jossa varsinainen työskentely tapahtuu.

Viikon päätteeksi sprinttihaara yhdistetään alustushaaraan. Alustushaara on tuotantoympäristön kopio, jossa testataan viimeisen kerran tehdyt koodin muutokset ja muutosten vaikutus tuotantohaaraan. Alustushaaran ideana on, että viimeistään tässä vaiheessa ilmenisi mahdolliset tuotantohaaraan liittyvät ongelmat. Lopuksi yhdistetään tuotantohaaraan.

Ohjelmakoodi voidaan kopioida käyttämällä Git:in tarjoamia komentoja. Tietolähteen kopiointi tapahtuu seuraavan esimerkin tapaan. Kun kopioiminen on suoritettu, se sijaitsee nimeään vastaavassa hakemistossa.

```
$ git clone http://github.com//testi_esimerkki
Cloning into 'testi_esimerkki'...
remote: Counting objects: 10, done.
remote: Compressing objects: 100% (8/8), done.
remove: Total 10 (delta 1), reused 10 (delta 1)
Unpacking objects: 100% (10/10), done.
```

Jos websivun tuotantopuolella (asiakkaalle toimitettava puoli) ilmenee jokin virhetilanne, voidaan sille luoda uusi haara:

```
$ git checkout -b "issue2"
```

Virhetilannetta voidaan tästä alkaa työstämään. Kun virhe on saatu korjattua, muutosten täytyy tulle näkyviin juuri luotuun uuteen haaraan. Luodaan nk. sitoutuminen tai kommitointi, jolla muutokset saadaan tallennettua nykyiseen haaraan.

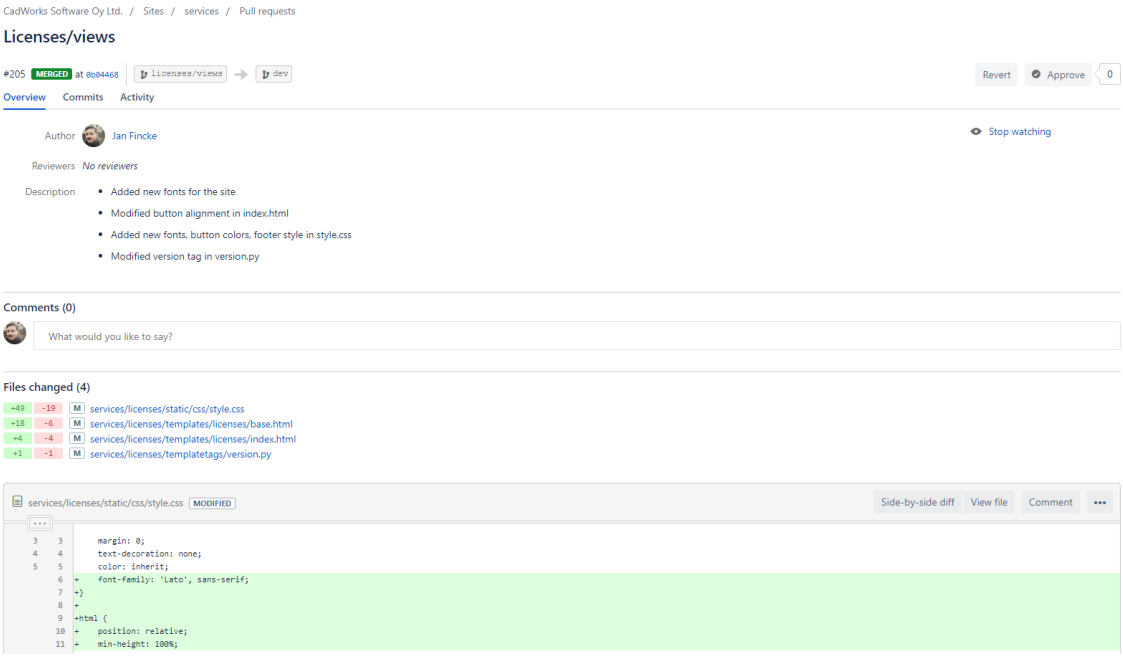
```
$ git commit -a -m "Fixed a crash on the website"
```

Muutokset tulee vielä yhdistää päähaaraan ”*master*”, jotta tehty korjaus saadaan toimitettua asiakkaalle asti.

```
$ git checkout master
$ git merge issue2
Updating f42c576..3a0874c
Fast-forward
 index.html | 2 ++
 1 file changed, 2 insertions(+)
```

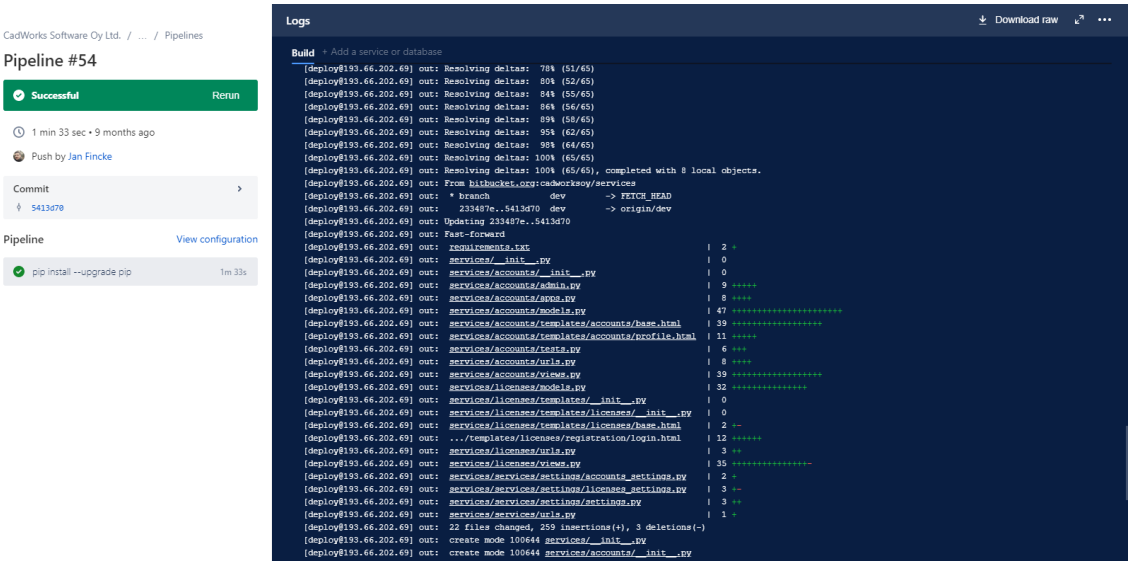
6.2.2 Bitbucket

Bitbucket ammattilaistiimeille suunnattu webpohjainen versionhallintapalvelu, jossa ohjelmakoodimme sijaitsee. Voimme Bitbucketin avulla seurata koodia, työnkulkua, hallita kehityshaaroja sekä tarkastella muiden muutoksia. Käytämme jokaisella haaran yhdistämiskerralla *pull-pyyntöjä*, koodin muutosten yhdistämisen ehdottamista, joiden avulla tiimin jäsenet voivat kuvan 12 tapaan tarkastella ja kommentoida koodia ennen sen yhdistämistä.



Kuva 12. Pull-pyyntö.

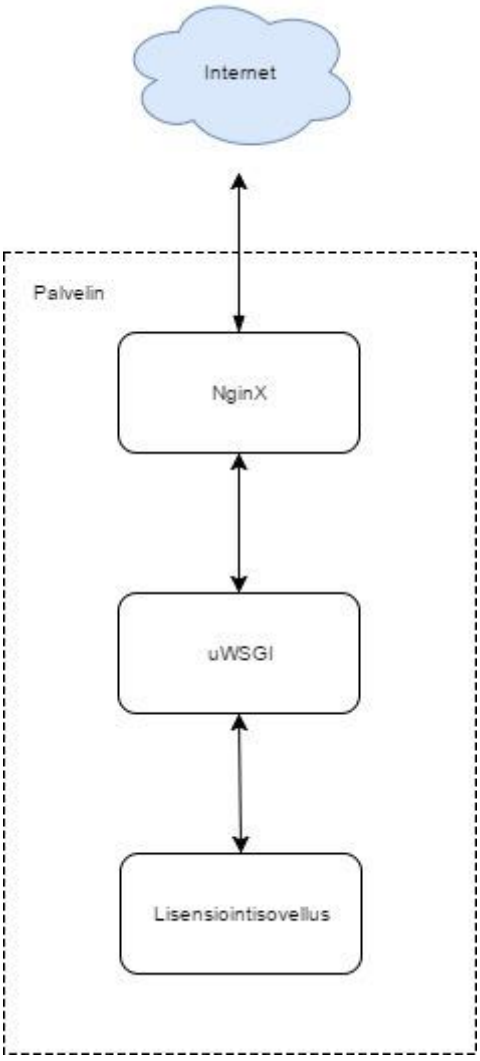
Käyttöönottoja varten Bitbucket tarjoaa Pipelines-sovelluksen, joka on tärkeässä asemassa projektissamme. Sen avulla koodin testaus ja käyttöönotto testi-, alustus-, ja tuotantopalvelimelle voidaan automatisoida ja se tekee koodin toimittamisesta jatkuvaa prosessia. Kuvassa 13 on esitetty testipalvelimen koodin käyttöönotto.



Kuva 13. Muokatun koodin käyttöönotto testipalvelimelle.

6.3 **Alusta**

Projektimme on moniosainen sovellus, joka vaatii alleen webpalvelimen ajamaan Django-sovellustamme. Verkosta yhdistetään webpalvelimelle, joka lähettää liikenteen eteenpäin Python-sovelluksille tarkoitetun rajapinnan kautta Django-sovellukselle, joka käsittelee palvelimelta saadut pyynnöt. Kuvassa 14 on esitetty yksinkertaistettuna projektin palvelinarkkitehtuurista.



Kuva 14. Yleiskuva sovellusta ajavasta alustasta.

6.3.1 Nginx

Palvelimemme käyttävät pohjanaan ilmaista Nginx-webpalvelinta, jonka välityksellä käyttäjät voivat yhdistää verkkosivustolle. Palvelin on konfiguroitu tarjoamaan Django-sovellukset sekä niiden kiinteät tiedostot, sekä esimerkiksi lataussivuston tiedostot. Django-sovellusta varten teimme kolme eri virtuaalipalvelinta, joihin koodi julkaistaan sen eri kehitysvaiheissa, esimerkiksi testihaaran koodi Bitbucketissa julkaistaan testipalvelimelle. Näin mahdollistetaan koodin testaaminen julkaistussa ympäristössä ennen, kuin toimiva koodi esitetään asiakkaalle.

Jokainen Django-sovellus sisältää yhden Nginx-konfigurointitiedoston. Tiedosto sisältää palvelinmääritykset, kuten nimen ja IP-osoitteen, sekä hakemistomääritykset kiinteille tiedostoille ja mediatiedostoille. Kuva 15 sisältää otteen palvelimelle luodusta lisensiointisovelluksen konfiguroinnista.

```
server {
    listen 80;
    server_name licenses.test.sites.cadworks.fi;
    access_log /var/log/nginx/licenses.access.log;
    error_log /var/log/nginx/licenses.error.log;

    set_real_ip_from 10.0.0.0/8;
    real_ip_header X-Real-IP;
    real_ip_recursive on;
```

Kuva 15. Ote Nginx-konfiguroinnista lisensiointisovellukselle.

6.3.2 uWSGI

uWSGI on sovellus, joka prosessoi Python-koodia ja tarjoaa WSGI (Web Service Gateway Interface) -rajapinnan, jonka avulla webpalvelin voi kommunikoida Djangon kanssa, eli käytännössä Nginx tarjoaa uWSGI:lle liikenteen ja Django-sovellus käsittelee saadun liikenteen. WSGI on standardisoitu rajapinta palvelimen ja sovellusten kommunikointiin.

Jokaiselle Django-sovellukselle konfiguroidaan uWSGI-asetukset, jolla voidaan määritellä erilaisia palvelimen kommunikointiin liittyviä asetuksia. Kuvassa 16 on esitetty lisensiointisovelluksen uWSGI-asetustiedostossa tehdyt määrittelyt.

```
1   [uwsgi]
2   chdir = /srv/services/services/
3   wsgi-file = licenses/wsgi.py
4   processes = 4 # number of cores on machine
5   logger = file:/var/log/uwsgi/vassals/licenses.log
6   virtualenv = /home/deploy/virtualenvs/services
7   max-requests = 5000
8   chmod-socket = 666
9   master = True
10  vacuum = True
11  socket = /var/run/uwsgi/licenses.sock
12  stats = /var/run/uwsgi/licenses.stats.sock
```

Kuva 16. Lisensiointisovelluksen uWSGI-määrittely.

WSGI antaa myös lisämuokattavuutta Django-sovelluksen ajamiseen. Se tarjoaa esimerkiksi mahdollisuuden määrittää, kuinka monta prosessia palvelimella käsittelee Djangon instansseja samanaikaisesti. Nginx-konfiguroinnin avulla muokattavuutta saadaan edelleen. Sivustomme on asetettu välittämään liikenne HTTPS:n kautta, eikä Django tiedä tästä mitään. Webpalvelin on myös täysin tiedoton siitä, mitä uWSGI tarjoaa, eli se ei tiedä, että taustalla on Django-sovellus.

6.3.3 PostgreSQL

Djangon asennuksen yhteydessä asentuu myös SQLite-tietokanta kevyttä datan tallentamista varten. Sovelluksemme tulee käsittelemään suurehkon määrän dataa, joten tarvitsimme tietokantaratkaisun, joka soveltuu tällaisen datan käsittelyyn ja olisi tarpeeksi hyvä suorituskyvyltään, kun kyseessä on usean käyttäjän sovellus, jonka vuoksi palvelin joutuu käsittelemään useita erilaisia tietokantapyyntöjä. Ratkaisuksi muodostui PostgreSQL, avoimen lähdekoodin relaatiotietokantapalvelin, jonka päälle kehitimme tietokannan. Tietokanta sisältää kaiken asiakas-, tuote-, analytiikka- ja lokitiedot, joita voi hyödyntää kaikkien Django-sovellusten kesken. PostgreSQL voidaan ottaa käyttöön jopa suurempaa datansäilöntää vaativissa projekteissa. Se on jonkin verran hitaampi kuin MySQL, mutta se on luotettavampi ja suoraviivaisempi ratkaisu Django-ympäristössä.

Sovellusta varten tulee luoda käyttäjä, jolla voidaan muokata tietokantaa. Teimme yksinkertaisesti yhden geneerisen tietokantakäyttäjän, jolloin tiedon haku helpottuu. Tämä ratkaisu ei ole olennainen tietoturvakysymys. Asetuksissa tulee myös määritellä kuvan 17 tapaan käytettävä tietokanta.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'services',
        'USER': 'services',
        'PASSWORD': POSTGRES_PASS,
        'HOST': 'localhost',
        'PORT': '5432',
    }
}
```

Kuva 17. Projektin tietokantamäärittely.

Djangon ominaisuuksiin kuuluu mallit. Mallit vastaavat suoraan tietokantamallia, esimerkiksi kuvan 18 ActivationKey-malli, joten Djangon mallit tulee tallentaa (*migrate*) PostgreSQL-kantaan.

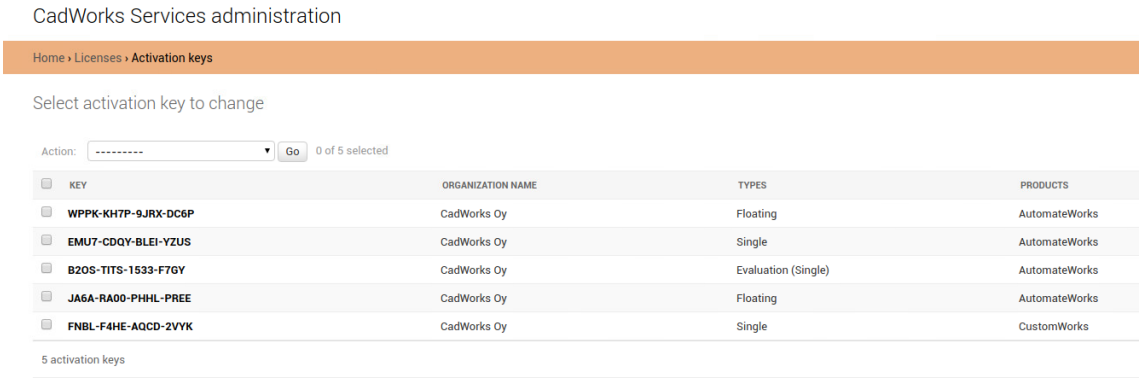
```
class ActivationKey(models.Model):
    objects = ActivationKeyQuerySet.as_manager()

    date_created = models.DateTimeField(auto_now_add=True)
    date_modified = models.DateTimeField(auto_now=True)
    key = models.CharField(default=activation_key_generator, max_length=255, unique=True)
    slug = autoslug.AutoSlugField(populate_from='key', null=True, unique=True)
    KEY_REGEX = r'(\S{4}-){3}\S{4}'
```

Kuva 18. ActivationKey-malli.

Tietokantaan tallentamista varten mallista tulee luoda migraatiotiedosto, eräänlainen tietokantamallin tilatieto, joka sisältää muutokset, jonka tietokanta tulkitsee SQL-koodiksi. Tiedosto generoituu `makemigrations`-komennolla, joka etsii uudet muutokset vertaamalla muita migraatiotiedostoja. Jos muutoksia löytyy, komennolla `migrate` voidaan tallentaa muutokset tietokantaan.

```
$ python manage.py makemigrations
$ python manage.py migrate
```

Tällöin tietokanta päivittyy ja sisältää uuden mallin, ja uusia mallin ilmentymiä voidaan luoda ja tarkastella kehitysvaiheessa esimerkiksi admin-työkalun avulla kuvassa 19.



Kuva 19. Aktivointiavaimia.

6.3.4 Celery

Celery on asynkroninen työjono, jonka avulla voidaan suorittaa taustaprosesseina tapahtuvia toimenpiteitä. Tällainen toimenpide voisi esimerkiksi olla tietokantojen synkronointi tiettyyn aikaan tai jonkin API-rajapinnan käyttö tunnin välein.

Celeryä voi myös hyödyntää verkkosivuille. Jos käyttäjät tekevät useita tietokantapyyntöjä, palvelin saattaa pian olla jumissa, jos pyyntöjä tulee useita samanaikaisesti. Ongelma hoituu asettamalla pyynnöt työjonoon, ja pyynnöt hoituvat asynkronisesti, jolloin palvelimen ei tarvitse erikseen odottaa pyyntöjen valmistumista.

Celeryn käyttö Django-sovelluksessa tapahtuu luomalla ensin uusi `celery.py` -tiedosto. Tulevia töitä varten luodaan muutamat Celeryn vaatimat asetukset.

```python
from __future__ import absolute_import
import logging
import os
import subprocess
import sys
from time import sleep
from celery import Celery
from django.conf import settings
from django.utils import timezone
from mailjet_rest import Client
import os.path
from time import strftime
from services.settings.local_settings import RABBITMQ_PASS
from django.db.models import Count
import time
import traceback
logger = logging.getLogger(__name__)

CONTACTS_LIST = 'ik3cmoei1@lists.mailjet.com'
os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'services.settings.licenses')
app = Celery('licenses', broker='amqp://services:{}@localhost:5672/licenses'.format(RABBITMQ_PASS))
app.config_from_object(settings)
app.autodiscover_tasks(['licenses'])


@app.task
def send_email():
    mailjet = Client(auth=(settings.MJ_API_PUBLIC, settings.MJ_API_PRIVATE))

    data = {
        'FromEmail': 'webmaster@cadworks.fi',
        'FromName': 'Mailjet Test',
        'Subject': 'Sending spa... I mean tests',
        'MJ-TemplateLanguage': 'true',
        'Text-part': 'This is a Celery app test!',
        'Html-part': '<h3>See your link below</h3><br /><a href={{ var:link }}>Click me!</a>',
        'Recipients': [{
            'Email': CONTACTS_LIST
        }],
        'Vars': {
            'link': 'http://www.google.fi'
        }
    }
    result = mailjet.send.create(data=data)
```

Kuva 20. Ote lisenssisovelluksen celery-tiedostosta.

Kuvan 20 `celery.py` -tiedosto sisältää lisenssisovelluksen Celery-työjonon testauksessa käytetyn sähköpostiviestin lähettämisen toteutuksen Celery-työnä. Riveillä 19-23 luodaan sähköpostia varten vastaanottajalista, joka pyydetään sähköpostisovelluksen API-rajapinnan kautta. Celery-työ alustetaan luomalla app -muuttuja, jolla kerrotaan, minkä Django-sovelluksen kanssa tämä `celery.py` -tiedosto kommunikoi. Se etsii sovellusta vastaavat asetukset, sekä etsii myös sovelluksen alta jo olemassa olevat työt. Sähköpostiviestejä varten luodaan `send_email()` -funktio, joka sisältää viestin rakenteen ja kommunikaation sähköpostisovelluksen kanssa.

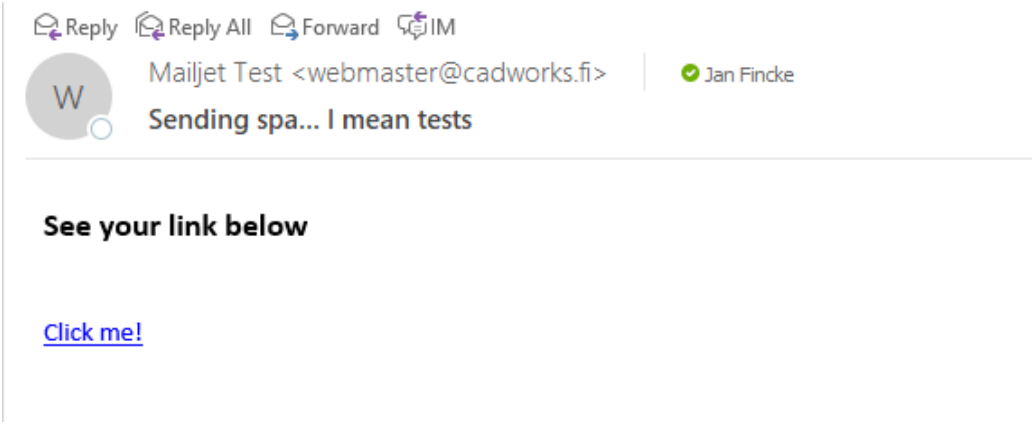
Jokaisen viestin lähetyksen käsittelee jokin Celeryn suorittaja (*worker)*, taustalla pyörivä prosessi, jonka avulla kaikki työjonossa olevat työt lopulta ajetaan.

```
$ celery -A licenses worker
```

Tällä komennolla käynnistetään uusi worker -prosessi. Prosessi jää tämän jälkeen odottamaan tulevia töitä. Kuvassa 20 luotu työ `send_email()` voidaan testata komentokehotteessa

```
$ python manage.py shell
>>> from licenses.celery import send_email
>>> send_email()
```

Suorittaja prosessoi työjonoon ilmestyvän työn, ja ajaa funktion. Kuvan 21 sähköpostiviesti lähetetään kaikille vastaanottajille, kun työ on prosessoitu.



Kuva 21. Celeryn testaukseen käytetty sähköpostiviesti.

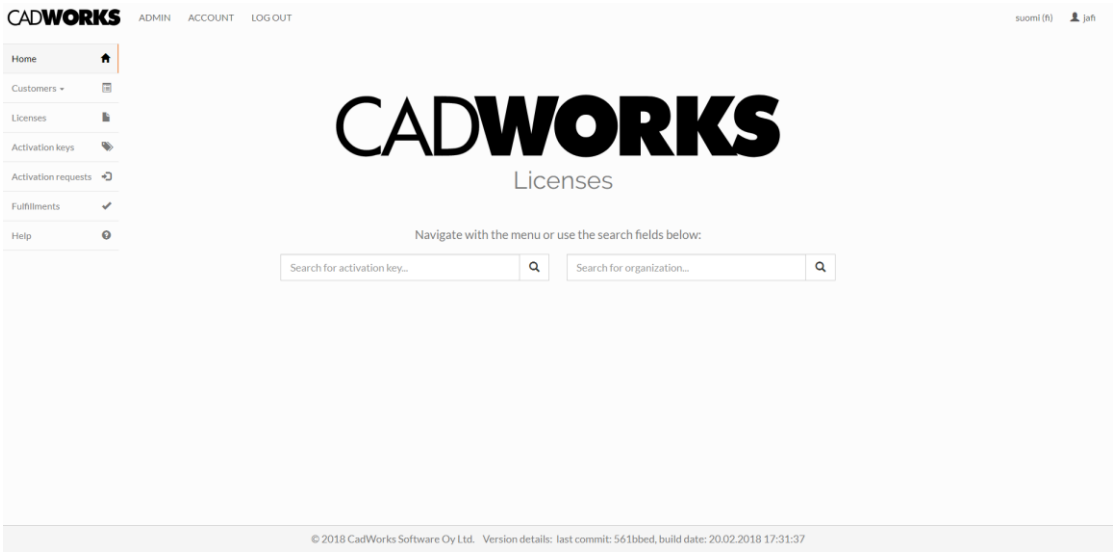
Projektin kannalta on tärkeää synkronoida olemassa oleva lisenssidata uuden tietokannan kanssa, jotta nekin voidaan siirtää uudelle alustalle. Toteutimme tietokannan synkronoinnin Celeryn avulla, jotta palvelin ei jäisi odottamaan erikseen jokaista synkronointia ja olisi käytettävissä päivittämisen aikana. Palvelimella on käynnissä aikataulutettu prosessi, joka tietyn väliajoin käy läpi olemassa olevaa dataa ja vertaa niitä oman tietokantansa riveihin, ja jos muutoksia löytyy, se luo omaan tietokantaansa vastaavat uudet rivit.

6.4 Kehitys

Sovelluksen kehityksen alkaessa päätimme, että sovelluksen logiikan tulee olla etusijalla. Sivuston tyylin suunnitteluun ei käytetty kuin muutama palaveri, jolloin tyylillinen ratkaisu oli vapaasti minun tehtävissä. Päätimme, että tietokannan kehitys, lisenssien aktivointi ja vapautus ja olemassa olevan datan tuominen järkevästi uuteen tietokantaan olivat tärkeimmässä asemassa.

Päätin luoda yksinkertaisen ja selkeän ulkoasun sivustolle, jossa tiedon hallinta on etusijalla. Hyödynnän Djangon syntaksia sekä tietokantakyselyitä HTML-sivujen luomiseen, sekä sovelluslogiikan puolella Pythonia ja JavaScriptiä sivuston vaatimiin erikoisempiin ominaisuuksiin.

6.4.1 Etusivu



Kuva 22. Etusivu.

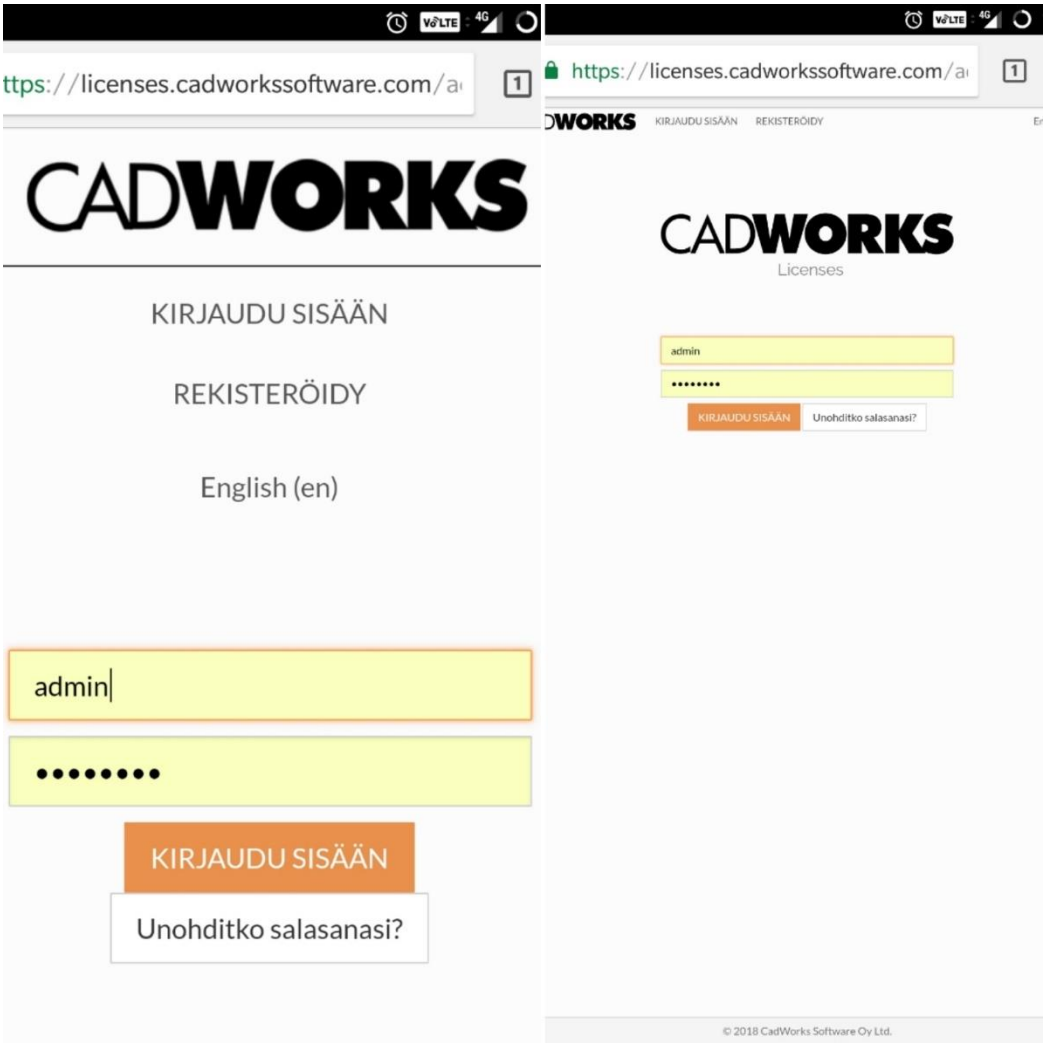
Sivuston kehitys alkoi sivuston tyylin suunnittelulla ja kuvan 22 etusivun luomisella. Djangon ominaisuuksiin lukeutuu HTML-mallinteiden perintä, jonka avulla koodin toistamista voidaan vähentää huomattavasti. Koko sivuston alla on yksi `base.html` -pohjamallinne, jossa määritellään osiot (*block)*. Osioiden sisällä olevat määrittelyt voidaan periä muissa mallineissa.

base.html

```
<!DOCTYPE html>
<head>
    {% block meta %}
        <meta name="viewport" content="width=device-width"/>
    {% endblock %}

    {% block scripts %}
        <script
                src="https://code.jquery.com/jquery-3.2.1.min.js"
                integrity="sha256-hwg4gsxgFZhOsEEamdOYGBf13FyQuiTwlAQgxVSNgt4="
                crossorigin="anonymous"></script>
        <script src="https://cdnjs.cloudflare.com/ajax/libs/bootstrap-select/1.12.4/js/bootstrap-select.min.js"></script>
        {% load static %}
        {% load pipeline %}
        {% load bootstrap3 %}
        {% load version %}
        {% load has_organization %}
        {% load svg %}
        {% load i18n %}
        {% load ga_settings %}
        {% javascript 'licenses' %}
        {% bootstrap_javascript %}
        {% bootstrap_css %}
        {% stylesheet 'licenses' %}
    {% endblock %}

    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/bootstrap-select/1.12.4/css/bootstrap-select.min.css">
    <link href="https://fonts.googleapis.com/css?family=Lato" rel="stylesheet">
    <link href="https://fonts.googleapis.com/css?family=Source+Sans+Pro" rel="stylesheet">
    <link href="https://fonts.googleapis.com/css?family=Raleway" rel="stylesheet">
    <link rel="icon" href="{% static 'images/key-32-211406.png' %}" type="image/png" sizes="32x32">

    {% url 'licenses-index' as index %}
    {% trans 'CadWorks Licenses' as main_title %}
    {% if request.path == index %}
        <title>{{ main_title }}</title>
    {% else %}
        <title>{{ main_title }} | {% block title %}{% endblock %}</title>
    {% endif %}
    {% block ga %}
        {% get_from_settings 'USE_GA' as use_ga %}
        {% if use_ga %}
            {% include 'licenses/google_analytics.html' with ga_tracking_id=ga_tracking_id %}
        {% endif %}
    {% endblock %}
</head>
```

Kuva 23. Head-määrittely.

Projektin pohjamallinne on tavallinen HTML-tiedosto. Kuvassa 23 on `base.html`-tiedoston alku. Loin `meta` -osion, johon kirjoitin mobiilialustojen näkymäoptimointia varten `viewport` -määrittelyn. Viewport edustaa websivun käyttäjälle näkyvää osaa. Elementin asetus `width=device-width` korjaa mobiililaitteen näkymän niin, että sivun koko sovitetaan käyttäjän mobiililaitteen näytön mukaan. (TutorialsPoint 2018.)



Kuva 24. Sivun viewport -vertailu mobiililaitteella.

Kuvassa 24 vasemmanpuoleisen kuvan kirjautumisnäkymässä on käytetty ko. metatietoa. Oikeanpuoleisen kuvan näkymä on tehty ilman tätä tietoa, jonka vuoksi se skaalaa kuvan samaan tapaan, kuin jos käyttäjä olisi käyttänyt sivua tietokoneen selaimessa. Käyttämällä tätä metatietoa saadaan mobiilikäyttäjille luettavampi versio sivusta, jossa elementit voidaan skaalata oikein.

Osioon `scripts` määritellään sivuston käyttämät kirjastot, kuten JQuery. Osiossa ladataan myös `{% load %}` -tagin avulla Djangon käyttämät lisäpaketit ja kaikki itse tehdyt lisäominaisuudet. Esimerkiksi rivin 13 `{% load static %}` lataa tämän mallinteen käytettäväksi kaikki staattiset tiedostot, kuten kuvat ja omat JavaScript -tiedostot. Kuvatiedoston lataaminen levyltä käy seuraavasti rivillä 31:

```
<link rel="icon" href="{% static 'images/key-32-211406.png' %}"
type="image/png" sizes="32x32">
```

Näin on mahdollista lataa kaikki tarpeelliset projektin tiedostot ilman, että tiedoston koko polkua tarvitsee muistaa. Se käyttää Djangon sisäänrakennettua staticfiles -sovellusta, jonka tarkoitus on kerätä kaikki projektin staattiset tiedostot yhteen paikkaan. (Django Software Foundation 2018.)

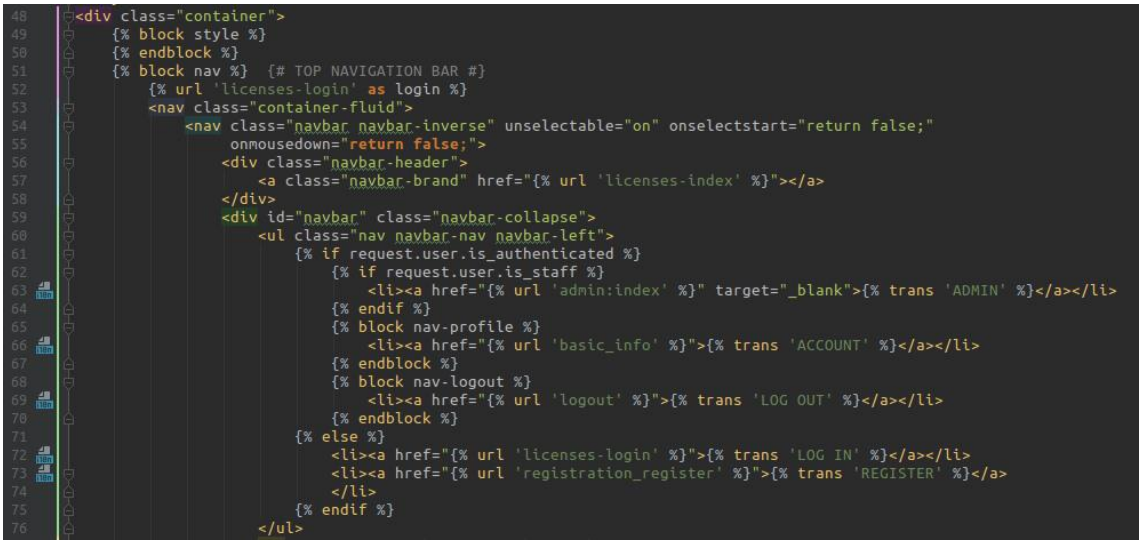
Riveillä 33-39 määritellään sivun `title` -otsikko. Määrittely tapahtuu katsomalla `request`-objektista sivun tämänhetkisen polun nimi. Rivillä 33 asetetaan etusivupolku uuteen muuttujaan `index` Djangon syntaksin tapaan. URL-linkin nimi määritellään projektin `urls.py` -tiedostossa. Rivin 34 muuttujaan `main_title` asetetaan merkkijono `{% trans %}` -tagin avulla, joka mahdollistaa sanojen kääntämisen. Sivun titteli määräytyy loogisesti niin, että jos käyttäjä on etusivulla, tittelinä näkyy vain sovelluksen nimi. Jos käyttäjä on jollani muulla sivulla, tittelimerkkijonoon sisältyy myös alisivun otsikko, joka on jaettu erikoismerkillä. Alisivujen titteli määritellään niiden omissa mallinnetiedostoissa.

Viimeinen osio, `ga`, sisällyttää koko sivustolle Googlen analytiikan seurannan, jonka avulla voimme seurata sivuston käyttäjämääriä ja liikennettä. Muuttuja `use_ga` on totuusarvo -tietotyyppiä edustava asetus projektin asetuksissa, joka haetaan käyttämällä `{% get_from_settings %}` -tagia. Jos asetus on tilassa tosi, tähän HTML-tiedostoon sisällytetään google_analytics.html -niminen mallinne, joka sisältää Google Analyticsin vaatiman skriptin. Djangon `{% include %}` lataa ja renderöi sen argumentiksi asetetun mallinteen nykyiseen kontekstiin, ja `with` –avainsanalla asetetaan vaihtoehtoisia muuttujia ladattavaan lisämallineeseen. (Django Software Foundation 2018.) Tähän sisällytetään Googlelta saatu seuranta-ID. Kuva 25 esittää tätä lisämallinetta. Mallinne sisältää ainoastaan JavaScriptillä kirjoitetun koodin.

```
1  <!-- Global Site Tag (gtag.js) - Google Analytics -->
2  <script async src="https://www.googletagmanager.com/gtag/js?id=UA-2533124-7"></script>
3  <script>
4      window.dataLayer = window.dataLayer || [];
5
6      function gtag() {
7          dataLayer.push(arguments)
8      }
9
10     gtag('js', new Date());
11
12     gtag('config', 'UA-2533124-7');
13
14     {% if request.user.is_authenticated %}
15         gtag('set', {'user_id': {{ request.user.id }}});
16     {% endif %}
17 </script>
```

Kuva 25. google_analytics.html.

Pohjasivun määritys jatkuu kuvan 26 mukaisesti HTML -rakenteen `<body>` -määrittelyllä. Sivusto ympäröidään aluksi yhtenäiseen `<div>` -ryhmään, johon käytin Bootstrap-kirjaston `container` -luokkaa. Ryhmän sisälle tein sivustokohtaista muokattavuutta varten `style` -osion, johon voisi sisällyttää ylimääräisiä sivustokohtaisia CSS-määrittelyjä.



```
<div class="container">
    {% block style %}
    {% endblock %}
    {% block nav %}  {# TOP NAVIGATION BAR #}
        {% url 'licenses-login' as login %}
        <nav class="container-fluid">
            <nav class="navbar navbar-inverse" unselectable="on" onselectstart="return false;"
                 onmousedown="return false;">
                <div class="navbar-header">
                    <a class="navbar-brand" href="{% url 'licenses-index' %}"></a>
                </div>
                <div id="navbar" class="navbar-collapse">
                    <ul class="nav navbar-nav navbar-left">
                        {% if request.user.is_authenticated %}
                            {% if request.user.is_staff %}
                                <li><a href="{% url 'admin:index' %}" target="_blank">{% trans 'ADMIN' %}</a></li>
                            {% endif %}
                            {% block nav-profile %}
                                <li><a href="{% url 'basic_info' %}">{% trans 'ACCOUNT' %}</a></li>
                            {% endblock %}
                            {% block nav-logout %}
                                <li><a href="{% url 'logout' %}">{% trans 'LOG OUT' %}</a></li>
                            {% endblock %}
                        {% else %}
                            <li><a href="{% url 'licenses-login' %}">{% trans 'LOG IN' %}</a></li>
                            <li><a href="{% url 'registration_register' %}">{% trans 'REGISTER' %}</a>
                            </li>
                        {% endif %}
                    </ul>
```

Kuva 26. Ylävalikon koodi.

Osioon `nav` loin sivuston ylävalikon. Tein tämän `<nav>` -tagin sisälle, jota käytetään sivustolinkkien määrittelyssä. Tähän ryhmään tein vielä yhden `<nav>` -ryhmän, jotta valikko saadaan ryhmiteltyä sivun ylälaitaan. Käytin Bootstrapin `navbar-` ja `navbar-inverse` -luokkia tyylittelyssä, jota muokkasin tekemällä lisää CSS-tyylittelyjä. Ylävalikon otsikko käyttää `navbar-brand` -luokkaa, joka asettaa otsikon omaksi elementikseen sivun vasempaan laitaan. Lisään otsikkoon kuvasisällön CSS-määrittelyn avulla. Estin myös tekstin maalaamisen valikosta. Linkkien URL-referenssit muodostetaan `{% url %}` -tagilla, johon annetaan argumentiksi URL-määrittelyn nimi.

Valikkojen linkit käyttävät ehtoa eritasoisille käyttäjille. Järjestelmävalvojasivun linkki ei kuulu näkyä peruskäyttäjälle, eikä myöskään tilinmuokkaussivun linkki kuulu näkyä uudelle, kirjautumattomalle käyttäjälle. Riviltä 59 alkaen määritellään valikon peruslinkit, jolla luodaan kyseinen ehto. Ehdon käyttämä `is_authenticated` -funktio etsii nykyisen mallinteen käyttäjän kirjautumistiedot ja edelleen funktio `is_staff` katsoo, onko käyttäjä tyypiltään järjestelmänvalvoja. Näin ollen Django huolehtii sisäisesti siitä, että linkit muodostetaan käyttäjän perusteella.

```
                    <ul class="nav navbar-nav navbar-right">
                        {% get_current_language as LANGUAGE_CODE %}
                        {% get_available_languages as LANGUAGES %}
                        {% get_language_info_list for LANGUAGES as languages %}
                        {% for language in languages %}
                            <li>
                                <a class="change-language"
                                   {% if language.code == LANGUAGE_CODE %}style="..."{% endif %}
                                   name="{{ language.code }}" href="{% url 'set_language' %}">
                                    {{ language.name_local }} ({{ language.code }})
                                </a>
                            </li>
                        {% endfor %}
                        {% if request.user.is_authenticated %}
                            {% block nav-currentuser %}
                                {% if user.first_name and user.last_name %}
                                    <li>
                                        <a href="{% url 'basic_info' %}"><span class="glyphicon glyphicon-user"></span>  {{ user.first_name }} {{ user.last_name }}
                                        </a>
                                    </li>
                                {% else %}
                                    <li>
                                        <a href="{% url 'basic_info' %}"><span class="glyphicon glyphicon-user"></span>  {{ user.username }}
                                        </a>
                                    </li>
                                {% endif %}
                            {% endblock %}
                        {% endif %}
                    </ul>
                </div><!--/.nav-collapse -->
            </nav>
        </nav><!--/.container-fluid -->
    {% endblock %}
```

Kuva 27. Ylävalikon oikean laidan määrittely.

Ylävalikon oikeaan laitaan tein 27 mukaan linkin käyttäjän profiilisivulle sekä linkin kielen vaihtoon. Käyttäjäprofiililinkin muodostaminen tapahtuu osiossa `nav-currentuser`, jossa linkin nimi muodostuu joko käyttäjän etu- ja sukunimen tai käyttäjänimen perusteella.

Kielen vaihto tapahtuu etsimällä listan sivustolla käytettävistä kielistä Djangon omissa asetuksissa, ja iteroimalla näitä, muodostaen linkin, joka vaihtaa sivuston käyttämän kielen. Määrittelyssä käytetty ehto piilottaa linkin, joka vastaa nykyistä käytettyä kieltä, ja näyttää vain linkin vaihtoehtoisiin kieliin. Linkin muodostaminen vaati JavaScript-koodia, jonka avulla tehdään POST-pyyntö Djangon kielen vaihdon sisäiseen koodiin. Tämän muodostamista helpotti se, että käytämme vain suomen ja englannin kieltä.

```
52      // language change function
53      var link = $('.change-language');
54      link.click(function (e) {
55          var url = link.attr('href');
56          var target = $(e.target);
57          var language = target.attr('name');
58
59          $.ajax({
60              type: 'POST',
61              url: url,
62              data: {
63                  'language': language
64              },
65              success: function (response) {
66                  location.reload();
67              },
68              error: function (rs, e) {
69                  alert(e);
70              }
71          })
72      })
```

Kuva 28. Kielen vaihtamisen funktio JavaScriptillä.

Kuva 28 esittää linkin muodostamisen funktiota. Koodissa kerätään tiedot nykyisestä linkistä ja suoritetaan AJAX-pyyntö. Jos pyyntö onnistuu, sivu ladataan uudelleen seuraavalle kielelle ja jos ei, muodostuu virheilmoitus.

Ylävalikkoon sisältyy myös jonkin verran CSS:llä tehtyjä tyylittelyjä, kuten esim. linkkien oranssi alaviiva, joka ilmestyy, kun käyttäjä vie hiiren linkin kohdalle. Linkkien teksteille tein lukuisia muutoksia johtuen Bootstrap-kirjaston jo olemassa olevasta tyylittelystä, jota piti muokata. Kuvat 29 ja 30 esittävät ylävalikon CSS-tyylittelyt kokonaisuudessaan.

```
/* --- NAVIGATION --- */

.nav {
    -moz-user-select: none;
    -webkit-user-select: none;
    -ms-user-select: none;
    user-select: none;
    cursor: default;
}

.disabled {
    pointer-events: none;
}

.navbar-nav {
    margin: 0;
}

.navbar-inverse .navbar-link {
    text-decoration: none;
    text-align: left;
    color: rgba(47, 47, 47, 0.8);
}

.navbar-inverse .navbar-link:hover {
    text-decoration: none;
    text-align: left;
    color: #000;
}

.nav-current-user {
    text-align: center;
}

.navbar .navbar-inverse {
    border-radius: 0;
    border: 0 rgba(0, 0, 0, 0.2);
    -webkit-box-shadow: none;
    -moz-box-shadow: none;
    box-shadow: none;
    text-align: center;
}

.navbar-inverse {
    background-color: rgba(252, 252, 252, 1);
    border: 0;
}

.navbar-inverse .navbar-nav > li > a {
    color: rgba(47, 47, 47, 0.8);
}

```

Kuva 29. Ylävalikon tyylittely 1.

```
.navbar-inverse .navbar-nav > li > a:hover {
    color: #000;
}

.navbar-inverse .navbar-nav > li > a::before, .navbar-inverse .navbar-nav > .active > a::before {
    content: "";
    position: absolute;
    width: 100%;
    height: 1px;
    bottom: 0;
    left: 0;
    background-color: rgba(237, 100, 3, 0.6);
    visibility: hidden;
    -webkit-transform: scaleX(0);
    transform: scaleX(0);
    -webkit-transition: all 0.2s ease-in-out 0s;
    transition: all 0.2s ease-in-out 0s;
}

.navbar-inverse .navbar-nav > li > a:hover:before, .navbar-inverse .navbar-nav > .active > a::before {
    visibility: visible;
    -webkit-transform: scaleX(1);
    transform: scaleX(1);
}

.navbar-inverse .navbar-nav > .active > a, .navbar-inverse .navbar-nav > .active > a:hover {
    color: #000;
    background-color: rgba(252, 252, 252, 1);
}

.navbar.navbar-default {
    border: 0;
    -webkit-box-shadow: none;
    box-shadow: none;
    background-color: rgba(252, 252, 252, 1);
}

.navbar.navbar-default .navbar-collapse {
    border: 0;
    -webkit-box-shadow: none;
    box-shadow: none;
    width: auto;
    margin: auto;
}

.navbar-brand {
    background: url("../images/cadworks.png") no-repeat;
    margin-top: 10px;
    margin-left: 5px;
    margin-right: 15px;
    width: 100%;
}
```

Kuva 30. Ylävalikon tyylittely 2.

Ylävalikon toteutus on käyttäjälle selkeä ja toimiva kokonaisuus, joka onnistui hyvin ajan ja omien taitojen puitteissa. Sivustolle lukeutuu myös useita tietokannan tietojen perusteella muodostuvia sivuja. Asiakkaan tulee esimerkiksi päästä etsimään tietoa lisensseistään ja aktivointiavaimista tai tietoa organisaationsa käyttäjistä ja työasemista. Tätä varten sivusto tarvitsi toisen navigointivalikon, jonka suunnittelin ja toteutin sivun vasemmalle puolelle.

```
{% block sidenav %}  {# PAGE LEFT SIDE MENU #}
    {% if user.is_authenticated %}
        <nav class="navbar navbar-default sidebar" unselectable="on" onselectstart="return false;"
             onmousedown="return false;">
            <div class="container-fluid">
                <div class="navbar-header">
                    <button type="button" class="navbar-toggle" data-toggle="collapse"
                            data-target="#bs-sidebar-navbar-collapse-1">
                        <span class="sr-only">{% trans 'Toggle navigation' %}</span>
                        <span class="icon-bar"></span>
                        <span class="icon-bar"></span>
                        <span class="icon-bar"></span>
                    </button>
                </div>
                <div class="collapse navbar-collapse" id="bs-sidebar-navbar-collapse-1">
                    <ul class="nav navbar-nav">
                        {% url 'licenses-index' as index %}
                        {% url 'licenses-organizations' as organizations %}
                        {% url 'licenses-organizations-search' as organizations_search %}
                        {% url 'licenses-organization' organization.slug as organization %}
                        <li {% if request.path == index %} class="active" {% endif %}>
                            <a href="{% url 'licenses-index' %}">
                                {% trans 'Home' %}
                                <span class="pull-right hidden-xs showopacity glyphicon glyphicon-home"></span>
                            </a>
                        </li>
```

Kuva 31. Sivuvalikon hahmottelu.

Sivuvalikko luodaan kuvan 31 mukaisesti HTML-tiedostossa lähes samaan tapaan kuin ylävalikko ja sijoitetaan omaan osioon. Sivuvalikon organisaatiolinkkien asettelussa on hieman eroja peruskäyttäjän ja järjestelmänvalvojan välillä kuvassa 32. Linkit tein sen perusteella mitä näkymiä nämä käyttäjätyypit tarvitsevat. Peruskäyttäjän ei tulisi päästä listaukseen, josta näkyisi kaikkien organisaatioiden käyttäjät ja työasemat, vaan ainoastaan ne, joihin käyttäjä itse kuuluu. Peruskäyttäjä näkee sivuvalikossa lisäksi listauksen kaikista omista organisaatioistaan.

```
                    {% if request.user.is_staff %}
                        <li
                                {% if request.path == organizations %} class="active"
                                {% elif request.path == organization %} class="active"
                                {% elif request.path == organizations_search %} class="active"
                                {% endif %}>
                            <a href="#" class="dropdown-toggle" data-toggle="dropdown">
                                {% trans 'Customers' %}
                                <span class="caret"></span>
                                <span class="pull-right hidden-xs showopacity glyphicon glyphicon-list-alt"></span>
                            </a>
                            <ul class="dropdown-menu forAnimate" role="menu">
                                <li>
                                    <a href="{% url 'licenses-organizations' %}">
                                        {% trans 'Organizations' %}</a>
                                </li>
                                <li>
                                    <a href="{% url 'licenses-workstations' %}">
                                        {% trans 'Workstations' %}</a>
                                </li>
                                <li>
                                    <a href="{% url 'licenses-users-admin' %}">
                                        {% trans 'Users' %}</a>
                                </li>
                            </ul>
                        </li>
                    {% else %}
                        <li>
                            <a href="{% url 'licenses-organizations' %}">
                                {% trans 'Organizations' %}
                                <span class="pull-right hidden-xs showopacity glyphicon glyphicon-list-alt"></span>
                            </a>
                        </li>
                        <li>
                            <a href="#" class="dropdown-toggle" data-toggle="dropdown">
                                {% trans 'Management' %}
                                <span class="caret"></span>
                                <span class="pull-right hidden-xs showopacity glyphicon glyphicon-file"></span>
                            </a>
                            <ul class="dropdown-menu forAnimate" role="menu">
                                {% for organization in request.user.organizations.all %}
                                    <li class="active">
                                        <a href="{% url 'licenses-organization' organization.slug %}">
                                            {{ organization.name }}
                                        </a>
                                    </li>
                                {% endfor %}
                            </ul>
                        </li>
                    {% endif %}
```

Kuva 32. Sivuvalikon organisaatiolinkkien eroavaisuus.

Loput linkeistä kuvassa 33 ovat mielestäni ns. yleisiä linkkejä, joita molemmat käyttäjätyypit voivat jakaa. Se, mitä kukin käyttäjätyyppi näkee linkin sivulla, riippuu näkymiin tehdyistä kyselyistä ja suodatuksesta, joka tehdään näkymälogiikan koodissa.

```
                        {% url 'all-licenses' as all_licenses_list %}
                        <li {% if request.path == all_licenses_list %} class="active" {% endif %}>
                            <a href="{% url 'all-licenses' %}">
                                {% trans 'Licenses' %}
                                <span class="pull-right hidden-xs showopacity glyphicon glyphicon-file"></span>
                            </a>
                        </li>
                        {% url 'all-keys' as keys %}
                        {% url 'all-keys-search' as keys_search %}
                        <li {% if request.path == keys %}
                            class="active" {% elif request.path == keys_search %}
                            class="active" {% endif %}>
                            <a href="{% url 'all-keys' %}">
                                {% trans 'Activation keys' %}
                                <span class="pull-right hidden-xs showopacity glyphicon glyphicon-tags"></span>
                            </a>
                        </li>
                        {% url 'activation-requests' as all_activation_requests %}
                        <li {% if request.path == all_activation_requests %} class="active" {% endif %}>
                            <a href="{% url 'activation-requests' %}">
                                {% trans 'Activation requests' %}
                                <span class="pull-right hidden-xs showopacity glyphicon glyphicon-log-in"></span>
                            </a>
                        </li>
                        {% url 'fulfillments' as all_fulfillments %}
                        <li {% if request.path == all_fulfillments %} class="active" {% endif %}>
                            <a href={{ all_fulfillments }}>
                                {% trans 'Fulfillments' %}
                                <span class="pull-right hidden-xs showopacity glyphicon glyphicon-ok"></span>
                            </a>
                        </li>
                        {% url 'licenses-help' as help %}
                        <li {% if request.path == help %}
                            class="active"
                        {% endif %}>
                            <a href="{% url 'licenses-help' %}">
                                {% trans 'Help' %}
                                <span class="pull-right hidden-xs showopacity glyphicon glyphicon-question-sign"></span>
                            </a>
                        </li>
                    </ul>
                </div>
            </div>
        </nav>
    {% endif %}
{% endblock sidenav %}
```

Kuva 33. Sivuvalikon linkkejä.

Linkkien lisätyylittelyä varten käytin Bootstrapin tarjoamia Glyphicon-kuvia, jotka ovat pieniä, yleisesti käytettyjä ikoneita. Aktiivisena olevan linkin oikealle puolelle ilmestyy oranssi viiva, joka helpottaa käyttökokemusta. Linkkien tekstit käännetään suomeksi Djangon avulla.

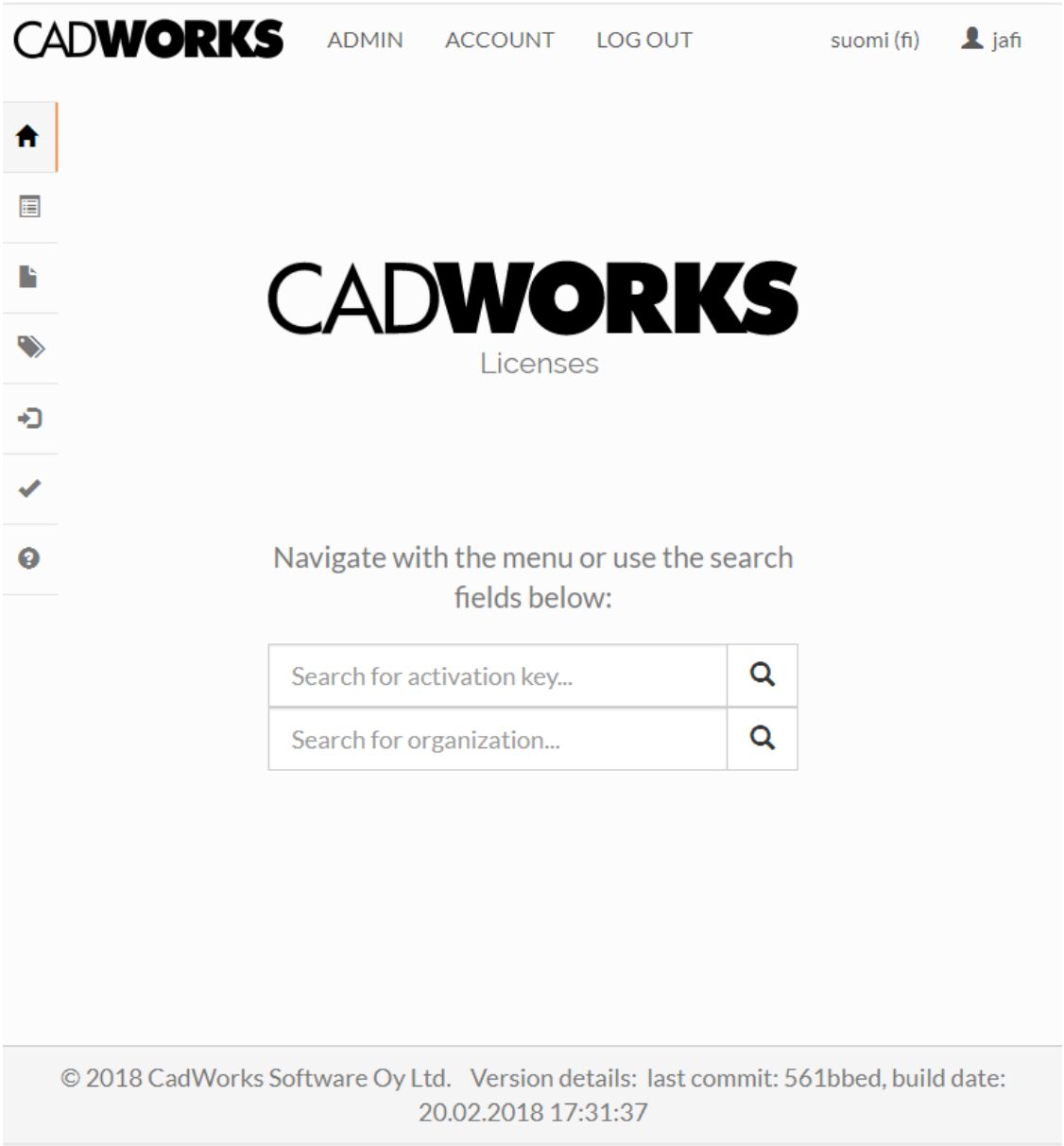
Rakensin mobiililaitteita silmällä pitäen sivuvalikkoon CSS:n avulla marginaalimuutoksia. Käyttäjä näkee valikon supistuvan koossa riippuen laitteen näytön koosta, mutta todellisuudessa teen vain marginaalimuutoksen perustuen sivun leveyteen.

```
nav.sidebar {
    width: 200px;
    height: 100%;
    float: left;
    margin-bottom: 0;
    margin-left: -160px;
}
```

Kuva 34. Sivuvalikon koon muuttaminen.

Sivuvalikon koon muuttamiseen kirjoitettu koodi kuvassa 34 on sijoitettu käytännössä CSS:n mediakyselysyntaksiin, jonka avulla voidaan tehdä erilaisia sääntöjä perustuen sivun leveyteen tai pituuteen, helpottaen websivun sovittamista mobiililaitteille. (W3Schools 2018.) Mediakysely muodostetaan seuraavan esimerkin mukaisesti

```
@media not|only mediatyyppi and (expressiot) {
      ...
      Kuvan 30 koodi
      ...
}
```



Kuva 35. Sivuston näkymä tabletilla.

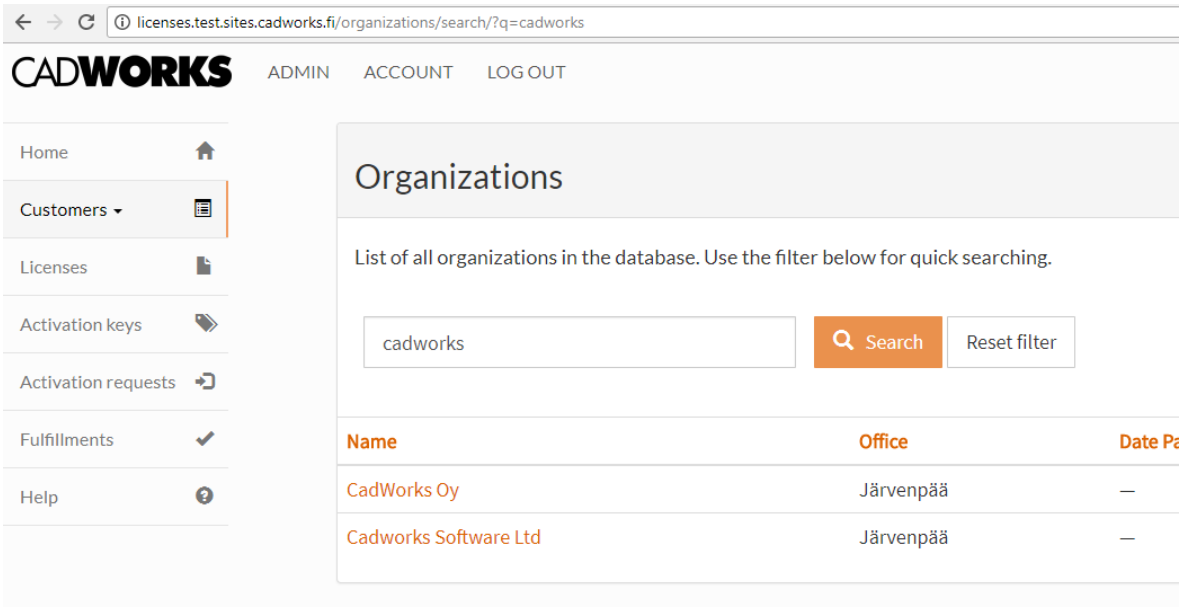
Sivuston skaalatessa alaspäin resoluution mukaan, esimerkiksi käyttämällä selainta tabletilla kuvassa 35, sivuvalikko siirtyy niin, että vain ikonit jäävät näkymään.

Pohjasivun osio `{% block content %}` sitoo kaiken etusivun pääsisällön. Etusivu ei ole keskeisessä asemassa. Jos esimerkiksi käyttäjä kuuluu vain yhteen organisaatioon, uudelleenohjataan käyttäjä suoraan oman organisaationsa sivulle sisäänkirjautumisen jälkeen. Etusivu on näin ollen tehty vain sen pakollisuuden vuoksi. Kuva (Kuva 36.) sisältää etusivun koodin sisältöosion.

```
{% block content %}
    <div class="licenses-index">
        <div class="page-header">
            <h1 class="h1">{% svg '2008_Cadworks_logo_LICENSES' %}</h1>
            <br>
        </div>
        {% if request.user.is_authenticated %}
            {% if request.user.is_staff %}
                <p class="lead text-muted">
                    {% trans 'Navigate with the menu or use the search fields below:' %}
                </p>
                {% trans 'Search for activation key...' as search_key %}
                {% trans 'Search for organization...' as search_org %}
                <div class="row">
                    <div class="col-lg-6">
                        <form action="{% url 'all-keys-search' %}">
                            <div class="input-group">
                                <input id="searchbox" class="form-control input-lg" name="q"
                                       placeholder="{{ search_key }}">
                                <span class="input-group-btn">
                                        <button class="btn btn-default btn-lg">
                                            <span class="glyphicon glyphicon-search"></span>
                                        </button>
                                    </span>
                            </div><!-- /input-group -->
                        </form>
                    </div><!-- /.col-lg-6 -->
                    <div class="col-lg-6">
                        <form action="{% url 'licenses-organizations-search' %}">
                            <div class="input-group">
                                <input id="searchbox" class="form-control input-lg" name="q"
                                       placeholder="{{ search_org }}">
                                <span class="input-group-btn">
                                <button class="btn btn-default btn-lg">
                                    <span class="glyphicon glyphicon-search"></span>
                                </button>
                            </span>
                            </div><!-- /input-group -->
                        </form>
                    </div><!-- /.col-lg-6 -->
                </div><!-- /.row -->
            {% else %}
                <p class="index-p-desktop lead text-muted">
                    {% trans 'Look for your organizations from the menu on the left to get started.' %}
                </p>
                <p class="index-p-mobile lead text-muted">
                    {% trans 'Look for your organizations from the toggle menu above to get started.' %}
                </p>
            {% endif %}
        {% else %}
            <p class="lead">
                {% trans 'Please log in or register to access this page.' %}
            </p>
        {% endif %}
    </div>
{% endblock %}
```

Kuva 36. Etusivun sisältö.

Itse etusivun sisältö on yksinkertainen ja sisältää peruskäyttäjille tekstin, joka ohjaa käyttämään edelleen valikon linkkejä. Järjestelmänvalvojille on aktivointiavaimen- ja lisenssinhakukentät, jotka tekevät tekstisyöttöä vastaan kyselyn tietokannasta. Nappia painamalla tehdään lomakkeen avulla POST-pyyntö aktivointiavaimen tai lisenssin hakutauluun. Haussa käytetty merkkijono näkyy selaimen osoitekentässä, ja samalla merkkijonolla suoritetaan haku.



Kuva 37. Organisaatiohaku.

Kuvassa 37 on suoritettu haku etusivun kautta merkkijonolla ”cadworks”. Seuraava näkymä on taulunäkymä, jossa on sama hakukenttä sekä suoritetun haun tulokset. Edellinen haku muistetaan hakukentässä.

Merkkijonon käsittely vaatii hakukentän tekstin tallentamista muuttujaan ja sen suodatusta Djangon puolella `views.py` -näkymäkoodissa kuvassa 38. Koodin `get_queryset()`, jolla tehdään näkymän tauluun kyselyn ehto, muodostuu merkkijonolistassa olevista merkeistä, jotka saadaan sivun `request`-objektista. Funktio `get_context_data()` auttaa viimeisen merkkijonon tallentamisessa. Aktivointikoodin etsiminen käyttää identtistä logiikkaa.

```
115  class OrganizationSearchListView(OrganizationsView):
116      """
117      Display OrganizationAdminListView filtered by the search query.
118      """
119
120      def get_queryset(self):
121          result = super(OrganizationSearchListView, self).get_queryset()
122          query = self.request.GET.get('q')
123          if query:
124              query_list = query.split()
125              result = result.filter(
126                  reduce(operator.and_,
127                         (Q(name__icontains=q) for q in query_list))
128              )
129          return result
130
131      def get_context_data(self, **kwargs):
132          context = super(OrganizationSearchListView, self).get_context_data(**kwargs)
133          query = self.request.GET.get('q')
134          if query:
135              query_list = query.split()
136              context['q'] = [q for q in query_list]
137          return context
138
```

Kuva 38. Hakukentän logiikka.

Pohjasivun viimeisenä osiona on sivun alaosassa sijaitsevan copyright- ja versiotiedon komponentti, footer, joka ladataan erillisenä mallineena pohjasivulle.

```
290         {% block footer %}
291             {% include 'licenses/footer.html' %}
292         {% endblock %}
293     </div>
294     </body>
295     </html>
296
```

Kuva 39. Pohjasivun footer-osio ja määrittelyn loppuosa.

Pohjasivun määrittelyn lopussa olevassa osiossa `footer` kuvassa 39 käytetään `{% include %}` -tagia, joka sisällyttää mallinteen (Kuva 40.) pohjasivun alaosaan.

```
1   {% load version %}
2   {% load i18n %}
3
4   <footer class="footer navbar-fixed-bottom">
5       <p class=" text-muted text-center">© 2018 CadWorks Software Oy Ltd.
6           {% if request.user.is_authenticated %}
7               {% trans 'Version details:' %} {% display_version %}
8           {% endif %}
9       </p>
10  </footer>
```

Kuva 40. footer.html.

Alatunniste näyttää yrityksen nimen sekä koontiversion viimeisimmän Gitin puolella tehdyn kommitin sekä sen päivämäärän.

Pohjasivun määrittely on todella kattava, ja syystäkin, sillä sitä periytetään lähes kaikissa muissa mallineissa. Tavoitteena oli luoda perusteellinen pohja muille sivuille, jotta sivujen tuottaminen olisi nopeaa.

Valmistuneen pohjasivun näkymä on erittäin yksinkertainen, koska se ei sisällä mitään tietokantakyselyitä tai request-objektin manipulointia. Ainoastaan malli on saatava käytettäväksi johtuen navigoinnissa käytetyistä käyttäjän organisaatioista. Kuvan 41 koodi sisältää pohjasivun näkymäkoodin kokonaisuudessaan.

```
49  class IndexView(SiteAuthRequiredMixin, TemplateView):
50      template_name = 'licenses/base.html'
51      model = Organization
52
```

Kuva 41. Pohjasivun näkymäkoodi.

Lopuksi luodaan etusivulle URL-kaavain kuvassa 42, ja muodostetaan linkki etusivulle.

```
27      url(r'^home/$', IndexView.as_view(), name='licenses-index'),
```

Kuva 42. Etusivun URL-kaavain.

6.4.2 Käyttäjätilin luominen

Rekisteröinnissä käytin kirjastoa nimeltä *django-registration-redux*, joka sisältää valmiiksi kaiken tarvittavan käyttäjätilin luomiseksi. Rekisteröinnin käyttöönotto ei vaadi kuin rekisteröintilomakkeen muokkausta oman näköiseksi.

Kirjaston asentamisen jälkeen se aktivoidaan lisäämällä se Djangon asetusten INSTALLED_APPS -listaan kuvassa 43, kuitenkin ennen `django.contrib.auth` -kirjaston lisäämistä.

```
68      'bootstrap3',
69      'pipeline',
70      'admin_gitlog',
71      'registration',
72      'django.contrib.admin',
73      'django_tables2',
74      'rest_framework',
```

Kuva 43. Registration -paketin lisääminen projektiin.

Logiikka muodostaa yksinkertaisen prosessin, jolla käyttäjä rekisteröi ja aktivoi käyttäjätilinsä sähköpostin avulla. Kun uusi käyttäjä luodaan, rekisteröinnin logiikka luo tietoja vastaan merkinnän tietokantaan. Käyttäjätili on käytössä vasta, kun käyttäjä on vahvistanut tilin luomisen seuraamalla vahvistuslinkkiä sähköpostiviestissä. Kirjastossa on sisäänrakennettu `RegistrationForm` -lomake, jota hyödynnän mallineessa. Kirjasto sisältää kaksi näkymää, `RegistrationView` ja `ActivationView`, jotka muodostavat rekisteröinti/aktivointi -logiikan, joita voin hyödyntää niitä muokkaamatta.

Kirjasto vaatii URL-kaavaimen muodostamisen, jonka kautta kirjaston logiikka etsii mallinteet ja osaa käyttää näkymiä. Kaavain muodostetaan yhdellä rivillä käyttämällä `include()` -funktiota (Kuva 44).

```
11      url(r'^listing/', include('registration.backends.default.urls')),
12
```

Kuva 44. Registration-kirjaston URL-kaavain.

Alustamisen jälkeen voin tehdä perusmallinteiden päälle uusia mallineita käyttämällä samoja nimiä, kuin kirjaston omat mallinteet. Mallinteet tulee luoda kansiorakenteeseen `templates/registration/`, jotta kirjaston logiikka osaa muodostaa näkymät oikein.

```
{% extends 'accounts/base.html' %}
{% load static %}
{% load svg %}
{% load i18n %}

{% block content %}
    <div id="registration">
        <div id="registration-header">
            <h1 class="h1">{% trans 'Registration to CadWorks Services' %}</h1>
        </div>
        <div id="registration-main">
            <form role="form" action="" method="post">
                {% csrf_token %}
                <p class="required">
                    <label class="required" for="id_username">{% trans 'Username*:' %}</label>

                    <input class="form-control input-lg" type="text" name="username" autofocus required id="id_username"
                           maxlength="150"/>
                    <ul style="...">
                        <li>
                            <small class="helptext text-muted">
                                {% trans 'Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.' %}
                            </small>
                        </li>
                    </ul>

                </p>
                <p class="required">
                    <label class="required" for="id_email">{% trans 'E-mail*:' %}</label>
                    <input class="form-control input-lg" type="email" name="email" required id="id_email"/>
                </p>
                <p class="required"><label class="required" for="id_password1">{% trans 'Password*:' %}</label>
                    <input
                            class="form-control input-lg"
                            type="password"
                            name="password1"
                            required
                            id="id_password1"/>
                    <small class="helptext text-muted">
                        <ul style="...">
                            <li>{% trans 'Your password can&#39;t be too similar to your other personal information.' %}</li>
                            <li>{% trans 'Your password must contain at least 8 characters.' %}</li>
                            <li>{% trans 'Your password can&#39;t be a commonly used password.' %}</li>
                            <li>{% trans 'Your password can&#39;t be entirely numeric.' %}</li>
                        </ul>
                    </small>
                </p>
                <p class="required"><label class="required" for="id_password2">{% trans 'Password (again)*:' %}</label>
                    <input class="form-control input-lg" type="password" name="password2" required id="id_password2"/>
                    <ul style="...">
                        <li>
                            <small class="helptext text-muted">
                        {% trans 'Enter the same password as before, for verification.' %}
                    </small>
                        </li>
                    </ul>
```

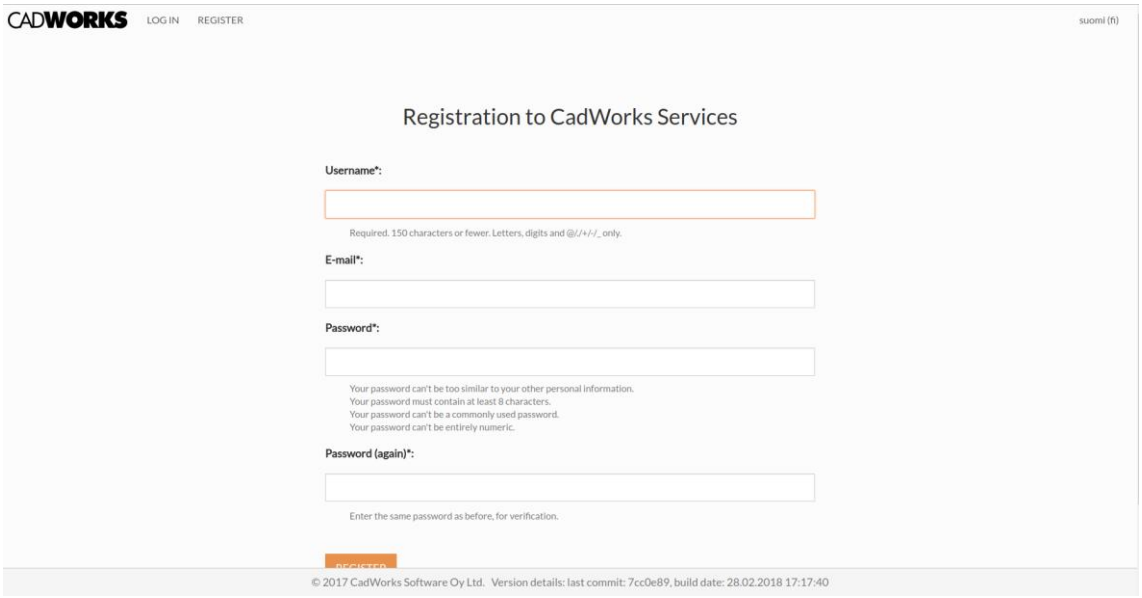
Kuva 45. Rekisteröintilomakkeen mallinne.

```
                </p>
                {% trans 'REGISTER' as register %}
                <input class="btn btn-lg cadworks-button" type="submit" value="{{ register }}"/>
            </form>
        </div>
    </div>
{% endblock content %}
```

Kuva 46. Rekisteröintilomakkeen mallinteen loppuosa.

Kuvat 45 ja 46 sisältävät rekisteröintilomakkeen mallinteen kokonaisuudessaan. Käyttäjätilin hallinnointiin teimme erillisen ”accounts” -nimisen projektin, jonka pohjasivua tämä mallinne hyödyntää. Pohjasivu on lähes sama kuin lisenssisivun pohja. Tähän mallineeseen ladataan käyttöön merkkijonojen käännökset muodostava i18n -kirjasto. Osioon `content` tein siistin lomakkeen. Lomake näyttää pakolliset kentät ja ilmoittaa käyttäjälle mahdollisista virheistä.

Valmiin lomakesivun kuvassa 47 tyylittelyssä on käytetty Bootstrapin hyödyllisiä tekstikenttäluokkia sekä lomakeluokkia, joilla voi tarkistaa käyttäjän syötettä. Lomake on keskitetty sivulle, jolloin myös mobiiliversio näyttää siistiltä.

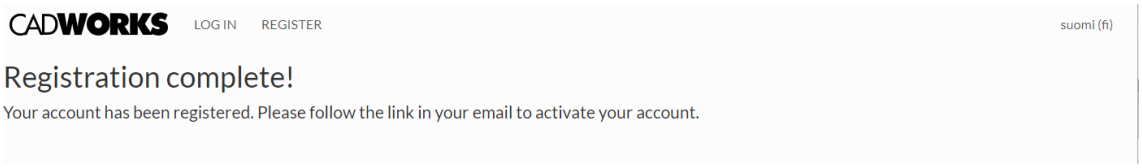


Kuva 47. Rekisteröintilomake.

Painamalla lomakkeen lähetysnappia käyttäjä ohjataan sivulle, jossa käyttäjää pyydetään vahvistamaan rekisteröinti. Tein yksinkertaisen sivun, jonka ainoa tarkoitus on välittää tieto käyttäjälle. Kuvat 48 ja 49 esittävät tämän mallinteen muodostamisen.

```
{% extends 'accounts/base.html' %}

{% block content %}
    <div class="container">
        <h1 class="page-header">Registration complete!</h1>
        <p class="lead">Your account has been registered. Please follow the link in your email to activate your account.</p>
    </div>
{% endblock content %}
```

Kuva 48. Rekisteröinnin hyväksymisen mallinne.



Kuva 49. Rekisteröinnin hyväksyminen.

Rekisteröintikirjasto huolehtii sähköpostiviestin lähettämisen käyttäjälle. Kirjasto sallii sähköpostiviestin muokkaamisen HTML-muodossa. Tein kuvan 50 mukaisen viestin, joka virallistaa viestin ja josta käyttäjä näkee, että viesti on tullut yritykseltämme.

```
<!doctype html>
<html lang="en">

<head>
    {% block title %}
        <title>Cadworks registration</title>
    {% endblock %}
</head>

<body>
{% block content %}
    <p>

        You (or someone pretending to be you) have asked to register an account at
        Cadworks Services.  If this wasn't you, please ignore this email
        and your address will be removed from our records.

    </p>
    <p>

        To activate this account, please click the following link within the next
        7 days:

    </p>

    <p class="lead">
        <a href="http://{{ request.site }}{% url 'registration_activate' activation_key %}">
            http://{{ request.site }}{% url 'registration_activate' activation_key %}
        </a>
    </p>
    <p>

        -Cadworks Management

    </p>
{% endblock %}
</body>

</html>

```

Kuva 50. Sähköpostiviestin HTML-tiedosto.

Kun käyttäjä klikkaa sähköpostiviestissä olevaa linkkiä, jonka jälkeen tili on käytössä. Käyttäjä ohjataan onnistuneen aktivoinnin sivulle 51 tapaan.

Kuva 51. Käyttäjätilin aktivoinnin onnistuminen.

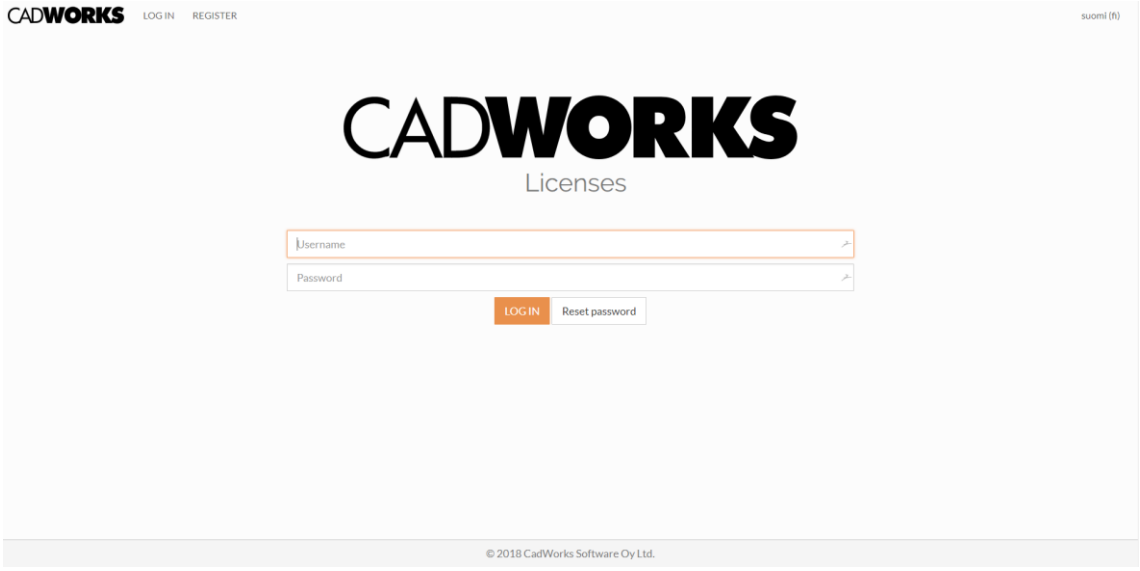
6.4.3 Sisäänkirjautuminen

Näkymän sisäänkirjautumiseen tein lisensiointiprojektin sisällä eikä ”accounts”-projektissa, sillä tämä näkymä on ensimmäinen näkymä, jonka peruskäyttäjä näkee. Kirjautumissivu sisältää lomakkeen, jossa pyydetään käyttäjän käyttäjänimeä ja salasanaa, ja jos siinä ilmenee virheitä, käyttäjä näkee kyseiset virheet. Lomake sisältää myös painikkeet rekisteröintiin ja salasanan palauttamiselle. Mallinne perii pohjasivun ominaisuudet. Kuva 52 edustaa tätä mallinetta.

```
{% extends 'licenses/base.html' %}
{% load static %}
{% load svg %}
{% load i18n %}

{% block title %}
    {% trans 'Login' %}
{% endblock %}

{% block content %}
    <div class="login-container">
        <div class="login">
            <div class="page-header">
                <h1 class="logo"><a href="{% url 'licenses-index' %}">{% svg '2008_Cadworks_logo_LICENSES' %}</a></h1>
            </div>
            <br>
            <div class="login-main">
                {% if form.non_field_errors %}
                    <ul class='form-errors'>
                        {% for error in form.non_field_errors %}
                            <small class="text-danger">{{ error }}</small>
                        {% endfor %}
                    </ul>
                {% endif %}
                <form method="post" action="" class="login-form">
                    {% csrf_token %}
                    <div class="form-inputs">
                        <p>
                            {% trans 'Username' as username %}
                            {% trans 'Password' as password %}
                            <input placeholder="{{ username }}" name="username" autofocus required id="id_username"
                                   class="form-control input-lg" maxlength="254"/>
                        </p>
                        <p>
                            <input placeholder="{{ password }}" type="password" name="password" required
                                   id="id_password"
                                   class="form-control input-lg"/>
                        </p>
                    </div>
                    <div class="row">
                        <span class="input-group-lg">
                            <button class="btn btn-lg cadworks-button">
                                {% trans 'LOG IN' %}
                            </button>
                        </span>
                        <span class="input-group-lg">
                            <a href="{% url 'password_reset_recover' %}" class="btn btn-default btn-lg">
                                {% trans 'Reset password' %}
                            </a>
                        </span>
                    </div>
                </form>
            </div>
        </div>
    </div>
{% endblock %}
```

Kuva 52. Sisäänkirjautumissivun mallinne.

Mallineessa tuli jonkin verran supistaa kirjautumattomille käyttäjille näkyvää tietoa. Tästä syystä esimerkiksi alatunnistetekstissä ei näy version tietoja. Sivulta kuvassa 53 pääsee helposti rekisteröintisivulle sivun ylävalikosta.

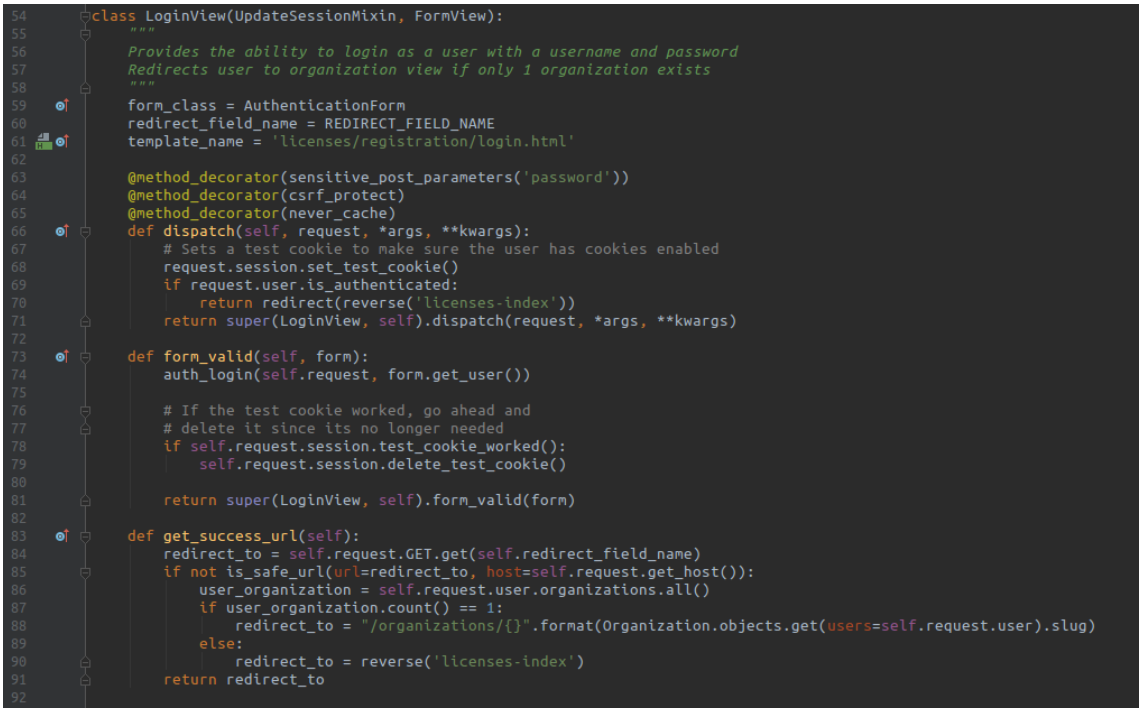


Kuva 53. Kirjautumissivu.

Sivun tyylittely sisältää kuvan 54 mukaiset määrittelyt. Lomakkeen koko muuttuu riippuen käyttäjän laitteesta. Lomakkeen syöttökentät ja painikkeet käyttävät myös Bootstrapin omia luokkia.

```
/* --- LOGIN --- */

.login-container {
    text-align: center;
    margin: 0 auto;
    width: 80%;
    padding-bottom: 5%;
}

.login-header {
    padding-bottom: 2%;
    border-bottom: 0;
}

.login form {
    text-align: center;
    display: inline;
}

.login form .form-errors {
    font-size: 1.1rem;
}

@media only screen and (min-width: 768px) {
    .login-container {
        text-align: center;
        margin: 0 auto;
        width: 50%;
    }

    .login-main {
        padding-bottom: 10%;
    }

    .login form {
        text-align: center;
    }
}

.login .row {
    margin: 0 auto;
}

.login .row .cadworks-button {
    margin: 0 auto;
}

@media only screen and (orientation: landscape) {
    .login .row .cadworks-button {
        margin: 0 auto;
    }

```

Kuva 54. Kirjautumissivun tyylimäärittelyt.



```
54  class LoginView(UpdateSessionMixin, FormView):
55      """
56      Provides the ability to login as a user with a username and password
57      Redirects user to organization view if only 1 organization exists
58      """
59      form_class = AuthenticationForm
60      redirect_field_name = REDIRECT_FIELD_NAME
61      template_name = 'licenses/registration/login.html'
62
63      @method_decorator(sensitive_post_parameters('password'))
64      @method_decorator(csrf_protect)
65      @method_decorator(never_cache)
66      def dispatch(self, request, *args, **kwargs):
67          # Sets a test cookie to make sure the user has cookies enabled
68          request.session.set_test_cookie()
69          if request.user.is_authenticated:
70              return redirect(reverse('licenses-index'))
71          return super(LoginView, self).dispatch(request, *args, **kwargs)
72
73      def form_valid(self, form):
74          auth_login(self.request, form.get_user())
75
76          # If the test cookie worked, go ahead and
77          # delete it since its no longer needed
78          if self.request.session.test_cookie_worked():
79              self.request.session.delete_test_cookie()
80
81          return super(LoginView, self).form_valid(form)
82
83      def get_success_url(self):
84          redirect_to = self.request.GET.get(self.redirect_field_name)
85          if not is_safe_url(url=redirect_to, host=self.request.get_host()):
86              user_organization = self.request.user.organizations.all()
87              if user_organization.count() == 1:
88                  redirect_to = "/organizations/{}".format(Organization.objects.get(users=self.request.user).slug)
89              else:
90                  redirect_to = reverse('licenses-index')
91          return redirect_to
92
```

Kuva 55. Kirjautumissivun näkymäkoodi.

Kirjautumissivun näkymäkoodi kuvassa 55 sisältää Djangon sisäänrakennetun kirjautumiskoodin, jota muokkasin niin, että jos käyttäjä kuuluu vain yhteen organisaatioon, käyttäjä ohjataan onnistuneen kirjautumisen jälkeen oman organisaationsa sivulle. Näkymä tarkistaa myös, jos käyttäjä on jo kirjautunut, ja uudelleenohjaa käyttäjän takaisin etusivulle.

Näkymä sisältää myös joitain Djangon omia dekoraattoreita. Rivin 63 `sensitive_post_paratemers()` piilottaa argumentteja vastaan olevat sivun elementit POST-pyynnön rakenteesta (Django Software Foundation 2018), jotta salasanatietoa ei voi väärinkäyttää. Dekoraattori `csrf_protect` mahdollistaa koko lomakkeiden salaamisen (Django Software Foundation 2018) ja `never_cache` ei sisällytä sivua selaimen välimuistiin (Django Software Foundation 2018). Nämä ovat erittäin tärkeitä käyttäjän tietojen väärinkäyttöä estettäessä.

Kirjautumissivulle tein vielä URL-kaavaimen kuvassa 56 kirjautumissivun käyttöönottoa varten.

```
55      url(r'^accounts/login/', LoginView.as_view(), name='licenses-login'),
```

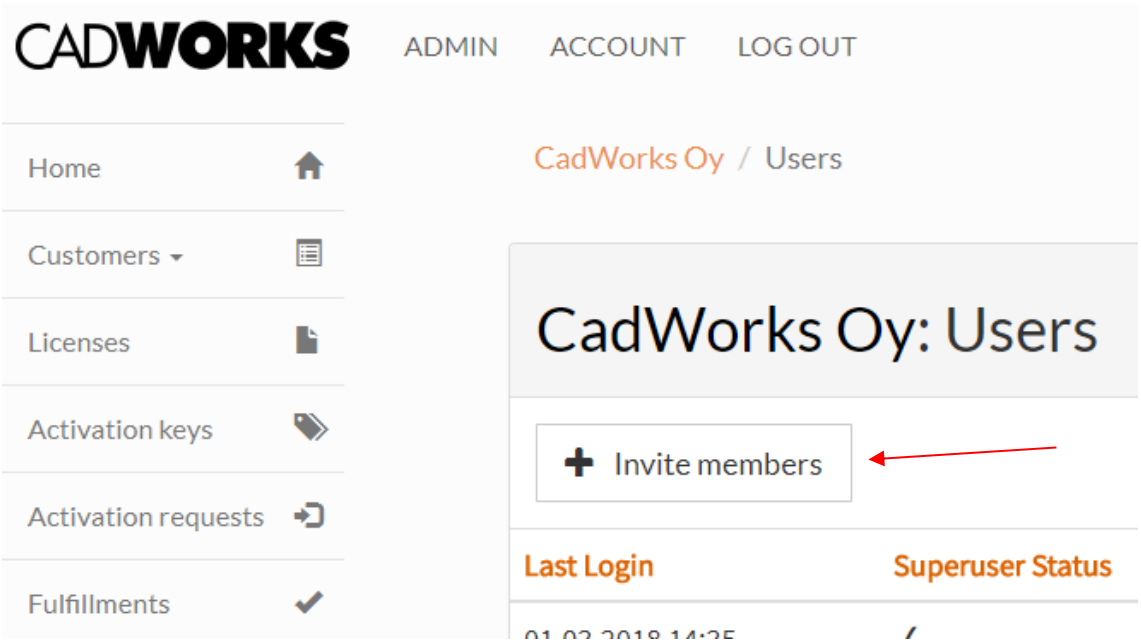
Kuva 56. Kirjautumissivun URL-kaavain.

6.4.4 Kutsut

Kutsujen avulla käyttäjä voi lähettää sähköpostiviestin haluamalleen samaan organisaatioon kuuluvalle henkilölle. Henkilölle luodaan lennosta uusi käyttäjätili tilapäisellä salasanalla, ja henkilöä ilmoitetaan kutsusta sähköpostiviestillä, joka

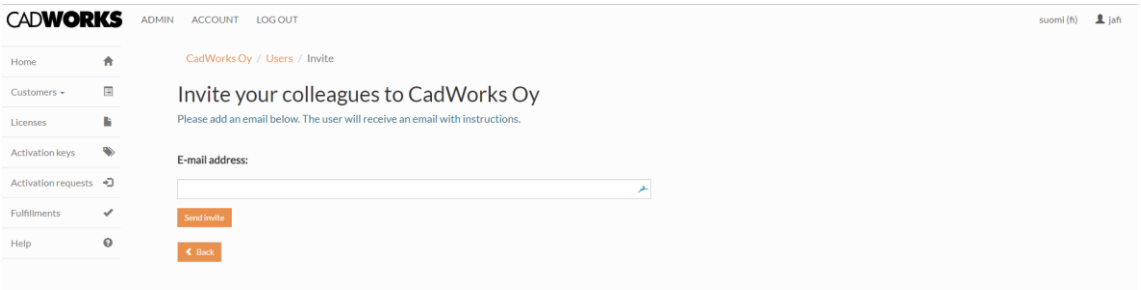
sisältää linkin, jolla käyttäjä ohjataan kirjautumissivun kautta salasanan vaihtamiseen tarkoitettuun näkymään. Uusi käyttäjä voi täällä asettaa itselleen uuden salasanan ja aloittaa tilin käyttämisen. Hyödynsin käyttäjäkutsujen tekemiseen aikaisemmin *django-invitations* -nimistä kirjastoa kutsujen logiikan muodostamiseen. Kirjastossa oli kuitenkin liian paljon ominaisuuksia, jotka eivät tulisi käyttöön. Otin kirjaston lähdekoodista mallia, ja tein sivuston käyttöön oman, supistetun version kutsuista.

Sivulle päästään, kun käyttäjä painaa organisaation käyttäjälistaussivulla (Kuva 57).



Kuva 57. Painike kutsun lähettämissivulle.

Painikkeen takaa aukeava kutsusivu kuvassa 58 sisältää lomakkeen, jossa annetaan sähköpostiosoite, jonne kutsu lähetetään.



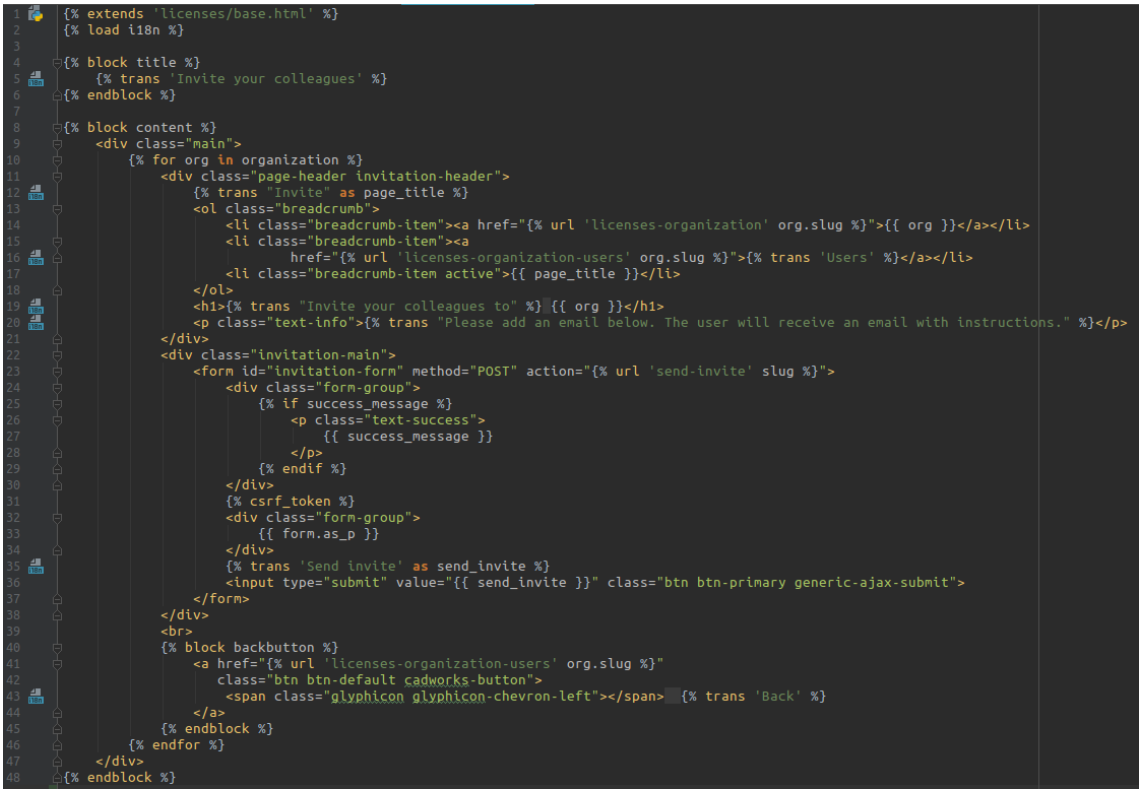
Kuva 58. Kutsusivu.

Kutsusivun tyylimäärittely kuvassa 59 sisältää muutoksia nappien väreille sekä lomakkeen kokomuutoksia.

```
.invitation {
    margin: 0 auto 10%;
}

.invitation-main {
    position: relative;
}

.invitation-main .messages {
    margin: 0 auto;
}

.invitation-main form p > * {
    margin: 10px 0;
}

.invitation-main form {
    width: 80%;
}

.invitation-main form > .btn {
    transition: background-color 100ms ease-out;
    background-color: rgba(225, 99, 2, 0.7);
    color: #fff;
    border: none;
    text-align: center;
}

.invitation-main form > .btn:hover {
    background-color: #fefefe;
    border-color: rgba(0, 0, 0, 0.2);
    color: #000;
}

.invitation-main form > .btn:focus {
    background-color: #fefefe;
    border-color: rgba(0, 0, 0, 0.2);
    color: #000;
    outline: 0;
}

.invitation-main form > .btn:focus {
    background-color: #5cb85c;
    border-color: #4cae4c;
    color: #fff;
    outline: 0;
}

```

Kuva 59. Kutsusivun tyylimäärittely.

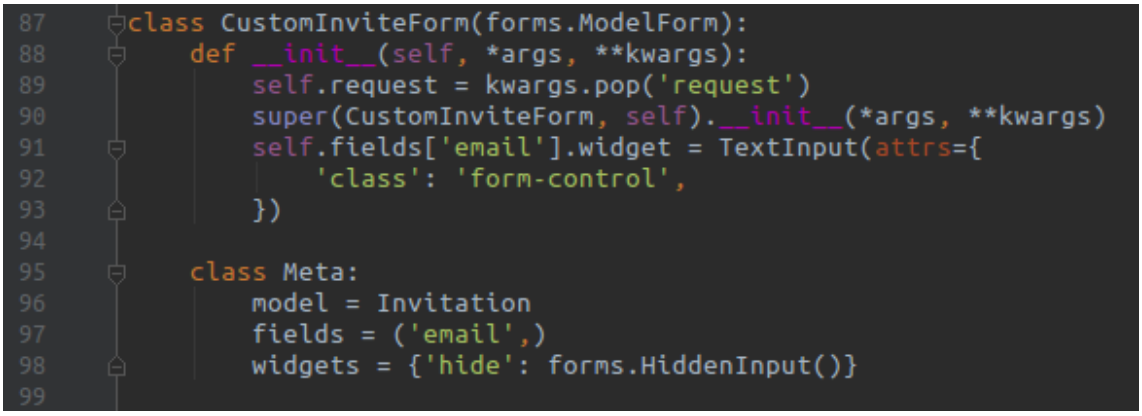
Kutsujen mallinne kuvassa 60 on rakennettu lähes samaan tapaan kuin rekisteröinnin ja sisäänkirjautumisen mallinteet. Sivu sisältää navigointia auttavan elementin, josta käyttäjä näkee puurakenteen, josta ilmenee sivujen looginen järjestys.



```
{% extends 'licenses/base.html' %}
{% load i18n %}

{% block title %}
    {% trans 'Invite your colleagues' %}
{% endblock %}

{% block content %}
    <div class="main">
        {% for org in organization %}
            <div class="page-header invitation-header">
                {% trans "Invite" as page_title %}
                <ol class="breadcrumb">
                    <li class="breadcrumb-item"><a href="{% url 'licenses-organization' org.slug %}">{{ org }}</a></li>
                    <li class="breadcrumb-item"><a
                            href="{% url 'licenses-organization-users' org.slug %}">{% trans 'Users' %}</a></li>
                    <li class="breadcrumb-item active">{{ page_title }}</li>
                </ol>
                <h1>{% trans "Invite your colleagues to" %} {{ org }}</h1>
                <p class="text-info">{% trans "Please add an email below. The user will receive an email with instructions." %}</p>
            </div>
            <div class="invitation-main">
                <form id="invitation-form" method="POST" action="{% url 'send-invite' slug %}">
                    <div class="form-group">
                        {% if success_message %}
                            <p class="text-success">
                                {{ success_message }}
                            </p>
                        {% endif %}
                    </div>
                    {% csrf_token %}
                    <div class="form-group">
                        {{ form.as_p }}
                    </div>
                    {% trans 'Send invite' as send_invite %}
                    <input type="submit" value="{{ send_invite }}" class="btn btn-primary generic-ajax-submit">
                </form>
            </div>
            <br>
            {% block backbutton %}
                <a href="{% url 'licenses-organization-users' org.slug %}"
                   class="btn btn-default cadworks-button">
                    <span class="glyphicon glyphicon-chevron-left"></span> {% trans 'Back' %}
                </a>
            {% endblock %}
        {% endfor %}
    </div>
{% endblock %}
```

Kuva 60. Kutsujen mallinne.

Mallineessa tuodaan lomake käyttöön `{{ form }}` -tagin avulla. Django sallii lomakkeiden tekemisen `forms.py` -tiedostoon erillisiksi luokkamäärittelyiksi, jolloin mallinteen määrittely helpottuu. Lomakkeessa määritellään ainoastaan merkkijonokenttä, joka sisältää sähköpostiosoitteen. Kuvassa 61 on esitetty kutsujen käyttämä lomake. Tein tietokantaan Invitation-luokan, jonka instanssi muodostuu jokaisesta lähetetystä kutsusta. Siitä ilmenee kutsun lähettäjä, vastaanottaja sekä muuta hyödyllistä tietoa. Mallin sähköpostiosoitekenttää käytetään lomakkeessa, jotta käyttäjätili voidaan luoda oikein. Mallilla on myös kutsun lähettämiseen tarvittavat funktiot, joita kutsutaan, kun lomake on lähetetty.



```
class CustomInviteForm(forms.ModelForm):
    def __init__(self, *args, **kwargs):
        self.request = kwargs.pop('request')
        super(CustomInviteForm, self).__init__(*args, **kwargs)
        self.fields['email'].widget = TextInput(attrs={
            'class': 'form-control',
        })

    class Meta:
        model = Invitation
        fields = ('email',)
        widgets = {'hide': forms.HiddenInput()}
```

Kuva 61. Lomake kutsuja varten.

Näkymäkoodin muodostaminen oli hieman hankalampaa. Koska käyttäjätilin käyttäjänimi muodostuu sähköpostiosoitteesta, kutsujen näkymäkoodin tulee

tarkistaa olemassa olevista käyttäjätileistä annettua sähköpostiosoitetta vastaava tili ja jos sellainen löytyy, vaihdetaan vain kyseisen tilin organisaatio. Jos tiliä ei löydy entuudestaan, luodaan uusi käyttäjätili.

```
class ProcessInviteeView(UserPassesTestMixin, FormView):
    """
    A view to process new invitees to either
     - send an invite if the user is new or
     - send a notification about a new organization.

    User will receive an email with instructions.
    """
    login_url = '/not-found/'
    redirect_field_name = None
    template_name = 'invitations/forms/_invite.html'
    form_class = CustomInviteForm
    model = Organization

    def test_func(self):
        if self.request.user.is_staff:
            return True
        return self.request.user.organizations.filter(slug=self.kwargs['slug'])

    @method_decorator(login_required)
    def dispatch(self, request, *args, **kwargs):
        return super(ProcessInviteeView, self).dispatch(request, *args, **kwargs)

    def get_form_kwargs(self):
        kwargs = super(ProcessInviteeView, self).get_form_kwargs()
        kwargs['request'] = self.request
        return kwargs

    def get_context_data(self, **kwargs):
        ctx = super(ProcessInviteeView, self).get_context_data(**kwargs)
        ctx['slug'] = self.kwargs['slug']
        ctx['organization'] = Organization.objects.filter(slug=self.kwargs['slug'])
        return ctx
```

Kuva 62. Kutsunäkymän muodostaminen.

Näkymä muodostetaan aikaisempien näkymien mukaisesti. Näkymässä tulee ottaa huomioon käyttäjän organisaatio. Näkymä sisältää myös testin, joka tarkistaa, kuuluuko näkymää käyttävä käyttäjä sivun organisaatioon. Funktion `dispatch()` edessä oleva dekoraattori `login_required` tarkistaa, onko käyttäjä kirjautunut, ennen kuin lomake lähetetään eteenpäin. Funktiolla `get_form_kwargs()` voidaan lähettää lomakkeen luokalle avainsana-argumentteja, ja `get_context_data()` sisällyttää mallineeseen lisämuuttujia.

Näkymäkoodin funktio `form_valid()` käsittelee lopullisen kutsun muodostamisen. Koodilla tarkistetaan, löytyykö annetulla sähköpostilla olemassa olevaa käyttäjää. Jos käyttäjä löytyy, tarkistetaan, kuuluuko se sivua vastaavaan organisaatioon. Jos ei kuulu, sen organisaatioksi asetetaan kutsujan organisaatio, jonka jälkeen luodaan Invitation-luokan uusi instanssi kutsusta. Kutsu on olemassa olevan käyttäjän tapauksessa ”ilmoitus” organisaation vaihtumisesta.

Jos olemassa olevaa käyttäjää ei löydy, luodaan uusi käyttäjä. Käyttäjätilin luonti vaatii salasanan asettamisen, jonka ratkaisin luomalla mielivaltaisen kirjaimia ja numeroita sisältävän kahdeksanmerkkisen merkkijonon ja asetin sen uuden käyttäjän salasanaksi, kunnes se vaihdetaan. Tilapäinen salasana annetaan `send_invitation()` -funktion argumentiksi, joka myöhemmin sisällyttää sen

sähköpostiviestissä. Uusi käyttäjä saa linkin sivustolle, jonne käyttäjä kirjautuu sähköpostin ja tilapäisen salasanan avulla, ja vaihtaa salasanan uuteen.

Muuttuja `invite` on Invitation-luokan uusi instanssi, joka lähettää tietoja vastaan sähköpostiviestin funktioiden `send_notification()` ja `send_invitation()` avulla. Kutsun linkki, jonka uusi käyttäjä saa, sisältää mielivaltaisen, monimerkkisen merkkijonon, jotta linkki olisi mahdollisimman uniikki. Tätä varten kutsutaan Pythonin funktiota `get_random_string()`, joka luo merkkijonon. Kuva **Error! Reference source not found.** esittää valmista kutsun muodostamisen logiikkaa.

```
133     def form_valid(self, form):
134         from django.utils.translation import ugettext as _
135 
136         email = form.cleaned_data["email"]
137         organization = Organization.objects.get(slug=self.kwargs['slug'])
138         success_message = _(' has been invited!')
139 
140         try:  # find existing user
141             invitee = User.objects.get(email=email, username=str(email))
142             if not invitee.organizations.filter(slug=self.kwargs['slug']):  # check if existing user has organization
143                 organization.users.add(invitee)  # if not, add user to organization
144                 organization.save()
145             try:
146                 invite = form.save()
147                 invite.inviter = self.request.user
148                 invite.organization = organization
149                 invite.key = get_random_string(64).lower()
150                 invite.save()
151                 invite.send_notification(self.request)  # create invitation
152                 return self.render_to_response(
153                     self.get_context_data(
154                         success_message="{} {}".format(email, success_message)))
155             except:
156                 return self.form_invalid(form)
157 
158         except User.DoesNotExist:  # no existing user found
159             invitee = User(email=email, username=str(email))  # create a new user
160             password = ''.join(random.choice(string.ascii_uppercase + string.digits) for _ in range(8))
161             kwargs = {
162                 'temp_pass': password
163             }
164             invitee.set_password(password)
165             invitee.save()
166             organization.users.add(invitee)  # add new user to an organization
167             organization.save()
168             try:
169                 invite = form.save()
170                 invite.inviter = self.request.user
171                 invite.organization = organization
172                 invite.key = get_random_string(64).lower()
173                 invite.save()
174                 invite.send_invitation(self.request, **kwargs)  # create invitation
175                 return self.render_to_response(
176                     self.get_context_data(
177                         success_message="{} {}".format(email, success_message)))
178             except:
179                 return self.form_invalid(form)
180 
181     def form_invalid(self, form):
182         return self.render_to_response(self.get_context_data(form=form))
183 
```

Kuva 63. Kutsun muodostamisen logiikka.

Näkymälle annetaan lopuksi kuvan 64 mukainen URL-kaavain. Kaavain sisällytetään muiden organisaationäkymien kaavainten joukkoon.

```
48          url(r'^send-invite/$', ProcessInviteeView.as_view(),
49              name='send-invite'),
```

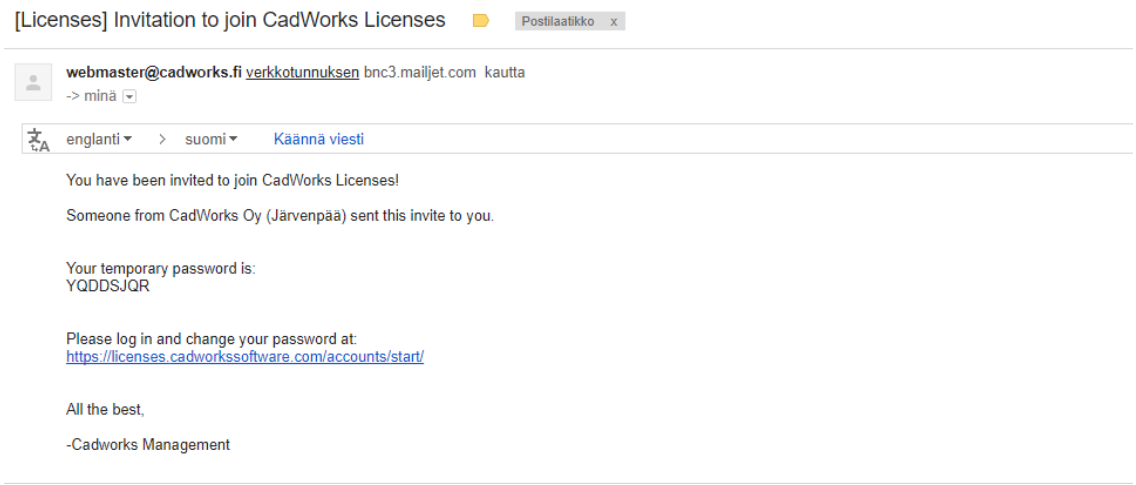
Kuva 64. Kutsunäkymän URL-kaavain.

Tein myös erilliset sähköpostiviestit riippuen siitä, onko viestin tyyppi ilmoitus vai uuden käyttäjän luonti. Viestit ovat tekstitiedostoja, joita palvelinpuolen logiikka käyttää. Esimerkiksi kuvan 65 viesti uudelle käyttäjälle, johon sisällytetään kutsun lähettäjän organisaatio, tilapäinen salasana sekä linkki sivulle.

```
{% load i18n %}
{% autoescape off %}
{% blocktrans with site_name as sitename %}

You have been invited to join CadWorks {{ sitename }}!

Someone from {{ organization }} sent this invite to you.


Your temporary password is:
{{ temp_pass }}


Please log in and change your password at:
{{ invite_url }}


All the best,

-Cadworks Management

{% endblocktrans %}
{% endautoescape %}

```

Kuva 65. Kutsuviesti.

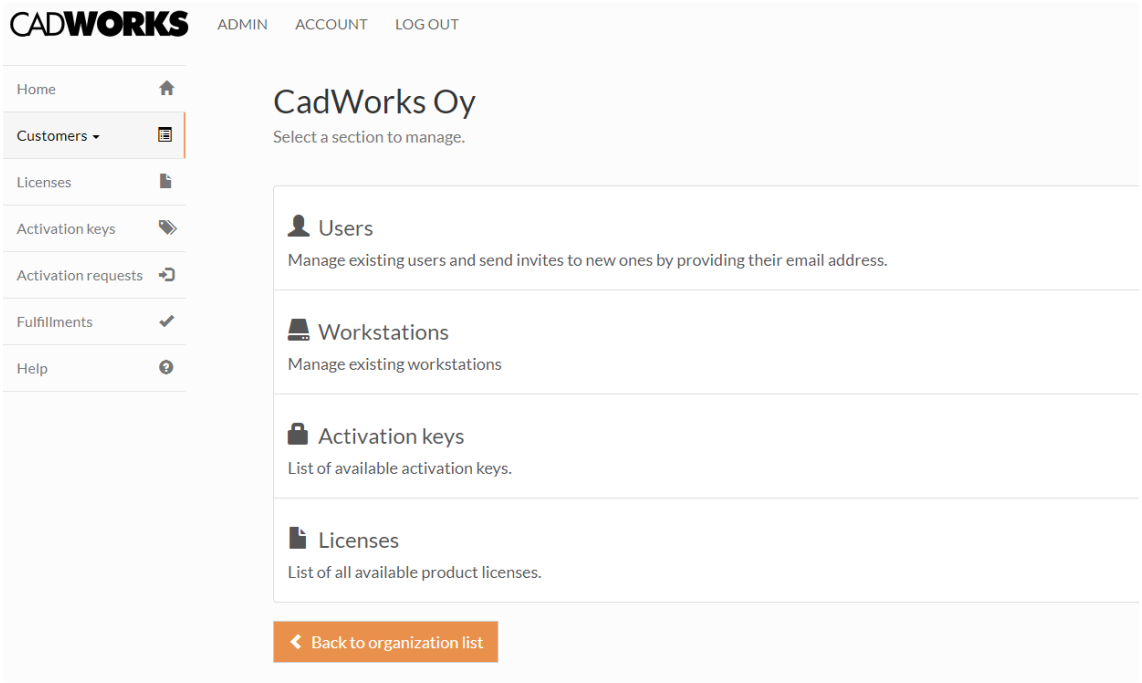
Viesti näkyy käyttäjälle kuvan 66 mukaisesti. Sähköpostiviestit lähetämme Mailjet -palvelun kautta.



Kuva 66. Käyttäjän saama kutsu.

6.4.5 Organisaation etusivu

Jokaiselle organisaatiolle on kuvan 67 mukainen etusivu, jota käyttämällä navigoidaan eri listanäkymiin. Sivu koostuu listasta ja sen elementeistä, jotka ovat painikkeita ja joissa on linkki seuraavalle sivulle. Listaelementeillä on jo valmiiksi hyvä tyylimäärittely, joka mukailee sivun ulkoasua, joten lisää tyylimäärittelyjä ei tarvita.



Kuva 67. Organisaation etusivu.

Painikkeet ovat `list-group` -elementin alkioita, jotka määritellään luokalla `list-group-item`. Luokan `list-group-item-action` avulla tehdään listan alkioista ”toiminnallisia”, jotta ne myötäilevät painikkeita. Luokka `flex-column` asettaa lapsielementtien suunnan ja `align-items-start` määrittää niiden kohdistamisen alkupaikan, tässä tapauksessa elementin vasen ylälaita

Toteutin painikkeiden tekstin asettelun käyttämällä Bootstrapin Flexbox-menetelmää, jonka määrittelyn aloitin jo listaelementtien määrittelyn aikana. Se korvaa vanhemman Grid-asettelumenetelmän, sillä Flexboxin määrittäminen on kehittäjille yksinkertaisempaa ja nopeampaa. Luokalla `d-flex` aktivoidaan elementin Flexbox-ominaisuudet, luokka `w-100` tekee elementistä leveydeltään 100% suhteessa emoelementin leveyteen, ja `justify-content-between` -luokan avulla määritellään lapsielementtien kohdistaminen pystysarakkeen y-suunnassa.

Kuva 68 edustaa valmista organisaation etusivua, jossa nämä listaelementit ovat käytössä. Etusivu sisältää painikkeen listaryhmän lopussa, joka vie käyttäjän takaisinpäin organisaatioiden listaukseen.

```
{% block content %}
    <div class="main">
        <div class="page-header">
            <h1>{{ organization.name }}</h1>
            <h4 class="text-muted">
                {% trans 'Select a section to manage.' %}
            </h4>
            <br>
        </div>
        <div class="list-group">
            <a href="{% url 'licenses-organization-users' organization.slug %}"
               class="list-group-item list-group-item-action flex-column align-items-start ">
                <div class="d-flex w-100 justify-content-between">
                    <h3 class="mb-1"><span
                            class="glyphicon glyphicon-user"></span>  {% trans 'Users' %}</h3>
                    <p class="mb-1">{% trans 'Manage existing users and send invites to new ones by providing their email address.' %}</p>
                </div>

            </a>
            <a href="{% url 'licenses-organization-workstations' organization.slug %}"
               class="list-group-item list-group-item-action flex-column align-items-start ">
                <div class="d-flex w-100 justify-content-between">
                    <h3 class="mb-1"><span
                            class="glyphicon glyphicon-hdd"></span>  {% trans 'Workstations' %}</h3>
                    <p class="mb-1">{% trans 'Manage existing workstations' %}</p>
                </div>

            </a>
            <a href="{% url 'licenses-organization-activation-keys' organization.slug %}"
               class="list-group-item list-group-item-action flex-column align-items-start ">
                <div class="d-flex w-100 justify-content-between">
                    <h3 class="mb-1"><span
                            class="glyphicon glyphicon-lock"></span>  {% trans 'Activation keys' %}
                    </h3>
                    <p class="mb-1">{% trans 'List of available activation keys.' %}</p>
                </div>

            </a>
            <a href="{% url 'licenses-organization-licenses' organization.slug %}"
               class="list-group-item list-group-item-action flex-column align-items-start ">
                <div class="d-flex w-100 justify-content-between">
                    <h3 class="mb-1"><span
                            class="glyphicon glyphicon-file"></span>  {% trans 'Licenses' %}</h3>
                    <p class="mb-1">{% trans 'List of all available product licenses.' %}</p>
                </div>
            </a>
        </div>
        {% block backbutton %}
            <a href="{% url 'licenses-organizations' %}"
               class="btn btn-default btn-lg cadworks-button">
                <span class="glyphicon glyphicon-chevron-left"></span>  {% trans 'Back to organization list' %}
            </a>
        {% endblock %}
    </div>
{% endblock %}
```

Kuva 68. Organisaation etusivun mallinne.

Kuvan 69 organisaation etusivun näkymäkoodi on erittäin yksinkertainen, koska mitään lisäominaisuuksia ei tarvita. Näkymän luokkaan `OrganizationDetailView` peritään useamman emoluokan ominaisuuksia. `SiteAuthRequiredMixin` sekä `LoginRequiredMixin` sisältävät testejä käyttäjälle, jolla tarkistetaan, onko käyttäjällä pääsyä koko lisensiointisivulle ja onko käyttäjä kirjautunut sivulle. `OrganizationMixin` on tekemäni luokka, joka sisältää kaikki tarpeelliset kyselyt tietokantaan, joilla haetaan käyttäjän organisaatiotietoja. Viimeinen, `TemplateView`, joka on Djangon sisäänrakennettu luokkapohjaisten näkymien luokka, jolla määritetään näkymä lataamaan mallinne (Django Softwaare Foundation 2018).

```
class OrganizationDetailView(SiteAuthRequiredMixin, LoginRequiredMixin,
                             OrganizationMixin, TemplateView):
    """
    Returns details of single organization.
    """
    login_url = '/not-found/'
    redirect_field_name = None
    template_name = 'licenses/organization.html'
```

Kuva 69. Organisaation etusivun näkymäkoodi.

Tämän sivun URL-kaavain kuvassa 70 sisältää organisaation tunnisteen `slug`, joka muodostuu organisaation nimestä. Se sisältää vain pieniä kirjaimia ja numeroita, ja sanojen välit on korvattu väliviivalla.

```
51      url(r'^organizations/(?P<slug>[-\w]+)/', OrganizationDetailView.as_view(), name='licenses-organization'),
52
```

Kuva 70. Organisaation etusivun URL-kaavain.

Yksittäiselle organisaatiolle liittyy useita eri sivuja, josta jokainen vaatii URL-kaavaimen. URL-kaavaimen voi määritellä siten, että yhteen kaavaimeen sisällytetään kaikki alikaavaimet, jotka tässä tapauksessa vaativat organisaation tunnisteen. Toteutin sen käyttämällä Djangon django.conf.urls -URL-määrittelykirjastosta löytyvällä `include()` -funktiota, jolla muodostetaan tällaisia kaavainten ketjutuksia.

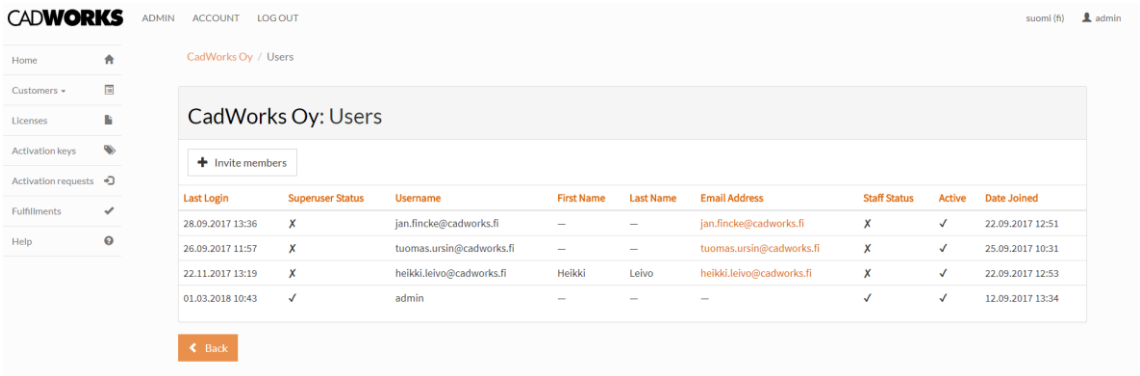
```
31      url(r'^organizations/(?P<slug>[-\w]+)/', include([
32          url(r'^search/$', OrganizationLicenseSearchView.as_view(),
33              name='licenses-organization-search'),
34          url(r'^users/$', OrganizationUsersView.as_view(), name='licenses-organization-users'),
35          url(r'^workstations/$', OrganizationWorkstationsView.as_view(),
36              name='licenses-organization-workstations'),
37          url(r'^workstations/(?P<pk>\d+)/$', OrganizationWorkstationDetailView.as_view(),
38              name='licenses-organization-workstations-detail'),
39          url(r'^activation_keys/$', OrganizationActivationKeysView.as_view(),
40              name='licenses-organization-activation-keys'),
41          url(r'^activation_keys/search/$', OrganizationActivationKeySearchListView.as_view(),
42              name='licenses-organization-activation-keys-search'),
43          url(r'^licenses/$', OrganizationLicensesView.as_view(), name='licenses-organization-licenses'),
44          url(r'^licenses/(?P<license_pk>\d+)/$', LicenseDetailView.as_view(), name='organization-license-details'),
45          url(r'^activation_keys/(?P<key_slug>[-\w]+)$',
46              OrganizationActivationKeyDetailView.as_view(),
47              name='licenses-organization-activation-keys-key'),
48          url(r'^send-invite/$', ProcessInviteeView.as_view(),
49              name='send-invite'),
50      ])),
```

Kuva 71. Organisaationäkymien URL-kaavaimet.

Kuvassa 71 on määritetty kaikki organisaation linkkien kaavaimet, jolloin koodi yksinkertaistuu ja toisto vähenee. Organisaatioon liittyy neljä eri listanäkymää, ja niiden kaavaimet ovat sisällytetty tähän ketjuun.

6.4.6 Käyttäjälistanäkymä

Käyttäjälistanäkymä kuvassa 72 listaa kaikki organisaatiolle kuuluvat käyttäjät ja niiden tietoja. Näkymä käyttää Djangon puolelle rakennettua taulua. Tein taulut käyttämällä lisäkirjastoa *django-tables2*, jonka avulla taulut luodaan Djangon puolella Pythonin avulla, ja renderöidään lopuksi näkymän kautta sivulle. Sivun kautta voidaan myös kutsua lisää käyttäjiä organisaatioon.



Kuva 72. Organisaation käyttäjälista.

Mallinne kuvassa 73 on yksinkertainen. Sivun pääelementit on sidottu Bootstrapin `panel` -luokan sisälle, jolla muodostetaan taulun paneeliulkoasu. Mallineessa on uusi tagi, `{% render-table %}`, jonka avulla tuodaan Djangon puolella luotu taulu näkymään.

```
{% extends 'licenses/organization.html' %}
{% load i18n %}

{% block title %}
    {{ organization.name }}
{% endblock %}

{% block content %}
    <div class="main">
        <div class="page-header">
            {% trans "Users" as page_title %}
            <ol class="breadcrumb">
                <li class="breadcrumb-item"><a
                        href="{% url 'licenses-organization' organization.slug %}">{{ organization.name }}</a></li>
                <li class="breadcrumb-item active">{{ page_title }}</li>
            </ol>
        </div>
        {% if table %}
            <div class="panel panel-default">
                <div class="panel-heading">
                    <h1><a href="{% url 'licenses-organization' organization.slug %}"
                           class="organization-header-link">{{ organization.name }}</a>: {{ page_title }}</h1>
                </div>
                <div class="panel-body">
                    <a href="{% url 'send-invite' organization.slug %}" class="btn btn-default btn-lg">
                        <span class="glyphicon glyphicon-plus"></span>
                        {% trans 'Invite members' %}
                    </a>
                </div>
                <div class="table-responsive">
                    {% load render_table from django_tables2 %}
                    {% render_table table %}
                </div>
            </div>
        {% else %}
            <p class="text-info">{% trans 'Error loading table data. Please contact CadWorks administration.' %}</p>
        {% endif %}
        {% block backbutton %}
            <a href="{% url 'licenses-organization' organization.slug %}"
               class="btn btn-lg cadworks-button">
                <span class="glyphicon glyphicon-chevron-left"></span>
                {% trans 'Back' %}
            </a>
        {% endblock %}

    </div>
{% endblock %}
```

Kuva 73. Käyttäjälistan mallinne.

```
class OrganizationUsersTable(tables.Table):
    class Meta:
        model = User
        attrs = {
            'class': 'table table-hover license-table paleblue',
        }
        exclude = ('password', 'id',)
```

Kuva 74. Käyttäjälistan määrittely.

Taulumäärittelyssä kuvassa 74 voidaan määritellä erikseen luokan ominaisuuksiksi mallin kentät, mutta peruskentät ja -ominaisuudet ovat tässä tapauksessa riittäviä. Määritin Meta-luokkaan malliksi User-luokan, joka hakee tietokannan vastaavan mallin kentät tauluun. Sanaston `attrs` avulla tauluun voidaan määrittää HTML-luokkia etukäteen, ja `exclude` on monikko, johon määritetään ne kentät, joita ei haluta taulun näyttävän näkymässä. Salasana- ja perusavainkentät halutaan luonnollisesti piilottaa käyttäjiltä.

```
class OrganizationUsersView(SiteAuthRequiredMixin,
                            OrganizationMixin, SingleTableView):
    """
    Show list of organization users.
    """

    login_url = '/not-found/'
    redirect_field_name = None
    template_name = 'licenses/organization_users.html'
    table_class = OrganizationUsersTable
    table_pagination = {
        'per_page': 15
    }

    def get_queryset(self):
        return self.organization.users.all()

```

Kuva 75. Käyttäjälistan näkymäkoodi.

Sivun näkymäkoodin luokka `OrganizationUsersView` kuvassa 75 perii organisaation etusivun tapaan emoluokista myös käyttäjätestin sekä käyttäjän organisaatioon liittyvät tietokantakyselyt. Näkymä määrittelee `login_url` -muuttujan, jolla määritellään linkki, johon kirjautumaton käyttäjä ohjataan. Tälle käyttäjälle tulee luoda mielikuva, että kyseistä sivua ei ole olemassa. Näkymään lisätään aiemmin tekemäni taulun luokka sekä taulun sivutus monikolla `table_pagination`. Funktion `get_queryset()` voin varmistaa, että tauluun ilmestyy vain kyseisen organisaation käyttäjät.

Sivulle ilmestyneen taulun voi todentaa kuvassa 76 olevan HTML-määrittelyä tarkastelemalla selaimesta sivun lähdekoodia.

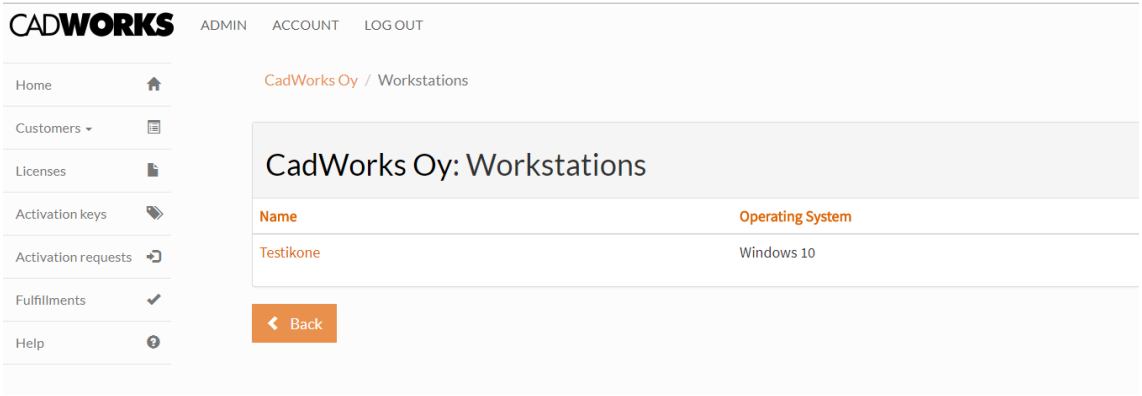
```
▼<div class="table-responsive">
  ▼<div class="table-container">
    ▼<table class="table table-hover license-table paleblue">
      ▼<thead>
        ▼<tr>
          ▶<th class="orderable username">…</th>
          ▶<th class="first_name orderable">…</th>
          ▶<th class="last_name orderable">…</th>
          ▶<th class="email orderable">…</th>
          ▶<th class="is_active orderable">…</th>
          ▶<th class="is_superuser orderable">…</th>
          ▶<th class="is_staff orderable">…</th>
          ▶<th class="last_login orderable">…</th>
          ▶<th class="date_joined orderable">…</th>
          </tr>
        </thead>
      ▶<tbody>…</tbody>
      </table>
    </div>
  </div>
```

Kuva 76. Käyttäjälistataulun HTML-lähdekoodi.

6.4.7 Työasemalistanäkymä

Kuvan 77 listanäkymästä voidaan selata organisaation kirjaamia työasemia. Listasta näkyy työaseman nimi ja sen käyttöjärjestelmän. Listan kautta pääsee myös

tarkastelemaan yksittäisen työaseman aktivointitapahtumia klikkaamalla työaseman nimestä.



Kuva 77. Organisaation työasemalista.

Mallineen toteutus kuvassa 78 on lähes identtinen käyttäjälistan mallineen kanssa.

```
{% extends 'licenses/organization.html' %}
{% load i18n %}

{% block title %}
    {{ organization.name }}
{% endblock %}

{% block content %}
    <div class="main">
        <div class="page-header">
            {% trans "Workstations" as page_title %}
            <ol class="breadcrumb">
                <li class="breadcrumb-item"><a
                        href="{% url 'licenses-organization' organization.slug %}">{{ organization.name }}</a></li>
                <li class="breadcrumb-item active">{{ page_title }}</li>
            </ol>
        </div>
        {% if table %}
            <div class="panel panel-default">
                <div class="panel-heading">
                    <h1><a href="{% url 'licenses-organization' organization.slug %}"
                           class="organization-header-link">{{ organization.name }}</a>: {{ page_title }}</h1>
                </div>
                <div class="table-responsive">
                    {% load render_table from django_tables2 %}
                    {% render_table table %}
                </div>
            </div>
        {% else %}
            <p class="text-info">{% trans 'Error loading table data. Please contact CadWorks administration.' %}</p>
        {% endif %}
        {% block backbutton %}
            <a href="{% url 'licenses-organization' organization.slug %}"
               class="btn btn-lg cadworks-button">
                <span class="glyphicon glyphicon-chevron-left"></span>
                {% trans 'Back' %}
            </a>
        {% endblock %}
    </div>
{% endblock %}
```

Kuva 78. Työasemanäkymän mallinne.

Työasemataulu sisältää uudenlaisen sarakkeen työaseman nimelle. Koska nimeä klikkaamalla halutaan siirtyä yksittäisen työaseman näkymään, nimen tulee olla linkki. Toteutin linkin käyttämällä taulukirjaston `LinkColumn` -luokkaa, joka määrittelee kustomoidun sarakkeen haluamillani tiedoilla, jossa jokainen sarakkeen alkio on linkki uuteen näkymään. Tauluun määritellään käytettävä malli, ja poissuljetaan mallin kenttiä, joita käyttäjien ei tarvitse tietää. Kuvassa 79 on valmis työasemataulu.

```
class OrganizationWorkstationsTable(tables.Table):
    name = tables.LinkColumn('licenses-organization-workstations-detail', args=[A('organization.slug'), A('id')],
                             verbose_name=_('Name'))

    class Meta:
        model = Workstation
        attrs = {
            'class': 'table table-hover license-table paleblue'
        }
        exclude = ('domain', 'organization', 'date_created', 'date_modified', 'id', 'hash',)
```

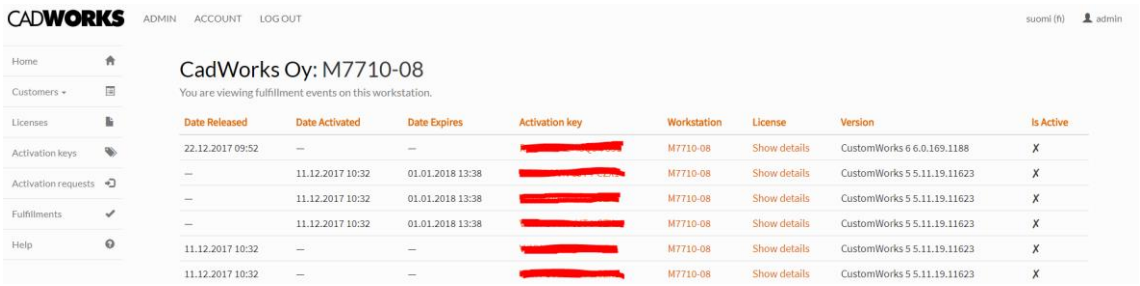
Kuva 79. Työasemalistan taulun määrittely.

Näkymäkoodi kuvassa 80 ei juurikaan eroa käyttäjälistanäkymästä. Funktio `get_queryset()` etsii tietokannasta kaikki organisaation työasemat.

```
class OrganizationWorkstationsView(SiteAuthRequiredMixin,
                                   OrganizationMixin, SingleTableView):
    """
    Render list of organization workstations.
    """

    login_url = '/not-found/'
    redirect_field_name = None
    template_name = 'licenses/organization_workstations.html'
    table_class = OrganizationWorkstationsTable
    table_pagination = {
        'per_page': 15
    }

    def get_queryset(self):
        return self.organization.workstations.all()
```

Kuva 80. Työasemalistan näkymäkoodi.

Yksittäisen työaseman näkymä kuvassa 81 sisältää taulun työaseman lisenssiaktivointitapahtumista. Taulusta selviää käyttäjälle, missä tilassa työasema on aktivointien kanssa. Näkymä auttaa myös järjestelmänvalvojia etsimään ongelmia aktivoinnissa.



Kuva 81. Yksittäisen työaseman näkymä.

Taulumäärittely kuvassa 82 sisältää kaikki Fulfillment-taulun rivit pois lukien perusavaimen ja työaseman nimen, koska taulu näyttää vain sen työaseman tiedot, jota tarkastellaan.

```
212 class WorkstationDetailTable(tables.Table):
213     class Meta:
214         model = Fulfillment
215         attrs = {
216             'class': 'table table-hover license-table paleblue'
217         }
218         exclude = ('id', 'workstation',)
219
```

Kuva 82. Yksittäisen työaseman taulumäärittely.

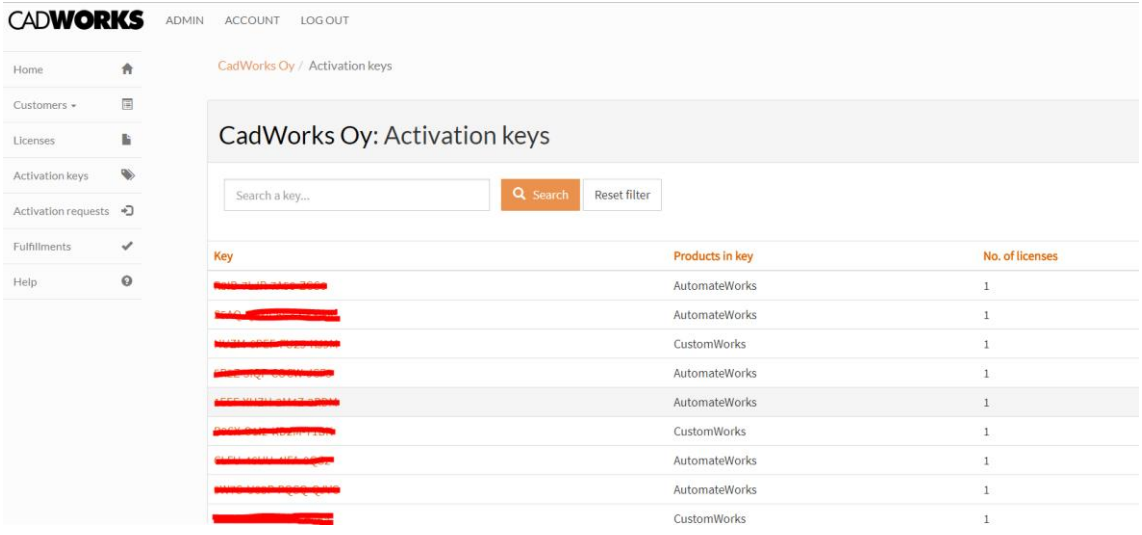
Näkymäkoodi kuvassa 83 on erittäin yksinkertainen tässä tapauksessa. Näkymän funktio `get_queryset()` asettaa ehdon kyselylle. Jos työasemalla on tapahtumia, palautetaan järjestetty listaus.

```
667 class WorkstationDetailView(SiteAuthRequiredMixin,
668                             WorkstationMixin, SingleTableView):
669     login_url = '/not-found/'
670     redirect_field_name = None
671     template_name = 'licenses/detailed_workstation.html'
672     table_class = FulfillmentTable
673
674     def get_queryset(self):
675         if self.workstation.fulfillments:
676             return self.workstation.fulfillments.order_by('-date_created')
677         else:
678             return self.workstation
679
```

Kuva 83. Yksittäisen työaseman näkymäkoodi.

6.4.8 Aktivointiavainnäkymä

Organisaatioiden aktivointiavainnäkymä kuvassa 84 listaa kaikki yksittäisen organisaation avaimet, niihin liitetyn tuotteen sekä avaimeen liitettyjen lisenssien lukumäärän. Avaimesta klikkaamalla käyttäjä voi tarkastella yksittäiseen avaimeen liittyvien lisenssin tietoja.



Kuva 84. Organisaation aktivointiavainlista.

Mallinne on rakenteeltaan yhtenäinen edellisten näkymien mallineiden kanssa, mutta sisältää hakukentän kuvassa 85, jolla voidaan etsiä merkkijonon avulla aktivointiavaimia. Hakukenttä on `panel` -elementin `panel-body` -alielementissä, ja sisältää syöttökentän sekä painikkeet haun suorittamiseen sekä syöttökentän nollaamiseen.

```
<div class="panel-body">
    <form action="{% url 'licenses-organization-activation-keys-search' organization.slug %}">
        <table class="table table-responsive borderless">
            <tbody>
                <tr>
                    <td>
                        {% trans 'Search a key...' as search_key %}
                        <input id="searchbox"
                               class="form-control input-lg"
                               name="q"
                               placeholder="{{ search_key }}"
                               value="{{ q | join:' ' }}"/>
                    </td>
                    <td>
                        <span class="input-group-lg">
                            <button class="btn btn-lg cadworks-button">
                                <span class="glyphicon glyphicon-search"></span>
                                {% trans 'Search' %}
                            </button>
                        </span>
                        <span class="input-group-lg">
                            <a href="{% url 'licenses-organization-activation-keys' organization.slug %}"
                               class="btn btn-default btn-lg">
                                {% trans 'Reset filter' %}
                            </a>
                        </span>
                    </td>
                </tr>
            </tbody>
        </table><!-- /.table -->
    </form>
</div>
```

Kuva 85. Aktivointiavainlistan mallineen hakukentän määrittely.

Hakukentän logiikan toteutin näkymäkoodissa, ja on identtinen sivuston etusivun hakukenttien kanssa. Hakukenttä aktivoi uuden näkymän `OrganizationActivationKeySearchListView`, joka lataa sivun ja näyttää taulun tiedot tehdyn haun perusteella.

```
class OrganizationActivationKeysView(SiteAuthRequiredMixin,
                                     OrganizationMixin, SingleTableView):
    """
    Show list of activation keys in organization.
    """
    login_url = '/not-found/'
    redirect_field_name = None
    template_name = 'licenses/organization_activation_keys.html'
    table_class = OrganizationActivationKeyTable
    table_pagination = {
        'per_page': 15,
    }

    def get_queryset(self):
        return ActivationKey.objects.by_organization(self.organization)


class OrganizationActivationKeySearchListView(OrganizationActivationKeysView):
    """
    Display ActivationKeyAdminListView filtered by the search query
    """

    def get_queryset(self):
        result = super(OrganizationActivationKeySearchListView, self).get_queryset()
        query = self.request.GET.get('q')
        if query:
            query_list = query.split()
            result = result.filter(
                reduce(operator.and_,
                       (Q(key__icontains=q) for q in query_list))
            )
        return result

    def get_context_data(self, **kwargs):
        context = super(OrganizationActivationKeySearchListView, self).get_context_data(**kwargs)
        query = self.request.GET.get('q')
        if query:
            query_list = query.split()
            context['q'] = [q for q in query_list]
        return context

```

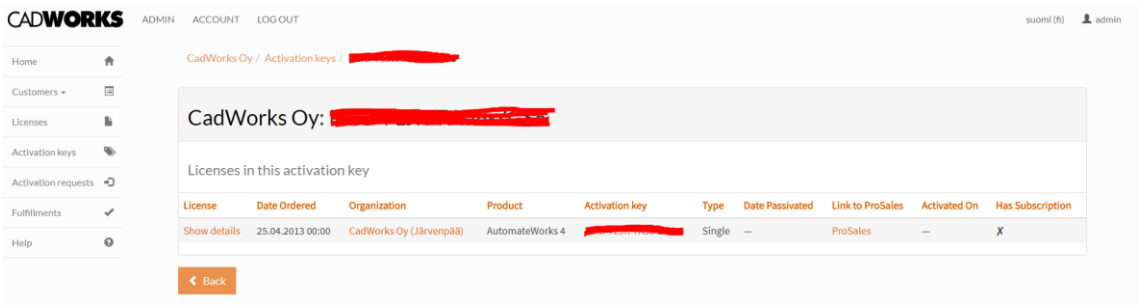
Kuva 86. Aktivointiavainlistan näkymäkoodi.

Taulun määrittely kuvassa 87 sisältää erikseen määritellyt sarakkeet avaimelle, avaimen tuotteelle ja lisenssien lukumäärälle, jolloin luokan metamäärittelyn mallia ei aseteta. Näissä käytetyt `product_names` ja `licenses_count` -ominaisuudet ovat luokan `ActivationKey` ominaisuuksia, joilla tehdään kyselyjä tietokantaan.

```
class OrganizationActivationKeyTable(tables.Table):
    key = tables.LinkColumn('licenses-organization-activation-keys-key',
                            args=[A('organization.slug'), A('slug')],
                            verbose_name=_('Key'))

    product_names = tables.TemplateColumn("{{ record.product_names }}",
                                          verbose_name="Products in key")

    count = tables.TemplateColumn("{{ record.licenses.count }}",
                                  verbose_name=_('No. of licenses'))

    class Meta:
        attrs = {
            'class': 'table table-hover license-table paleblue'
        }

```

Kuva 87. Aktivointiavainlistan taulumäärittely.

Aktivointiavaimen näkymä kuvassa 88 listaa avaimelle kuuluvat lisenssit ja sen tiedot. Käyttäjä voi halutessaan navigoida yksittäisen lisenssin tietoihin tai organisaation tietoihin. Järjestelmänvalvojille on linkki asiakashallintasovelluksen lisenssitietoriville.



Kuva 88. Yksittäisen aktivointiavaimen näkymä.

Aktivointiavaimen näkymäkoodissa kuvassa 89 haetaan erikseen aktivointiavaimen organisaatio ja aktivointiavaimen lisenssit. Taulun hakemisen funktio `get_table()` asettaa oikean taulun näkymään aktivointiavaimen tiedoilla.

```
494  class OrganizationActivationKeyDetailView(SiteAuthRequiredMixin,
495                                            ActivationKeyMixin, SingleTableView):
496      login_url = '/not-found/'
497      redirect_field_name = None
498      template_name = 'licenses/organization_detailed_activation_key.html'
499      table_class = ActivationKeyDetailTable
500      table_pagination = {
501          'per_page': 15,
502      }
503
504      @property
505      def organization(self):
506          return self.activation_key.organization
507
508      def get_queryset(self):
509          return self.activation_key.licenses.all()
510
511      def get_table(self):
512          return ActivationKeyDetailTable(self.get_queryset(), activation_key=self.activation_key)
513
514      def get_context_data(self, **kwargs):
515          context = super(OrganizationActivationKeyDetailView, self).get_context_data(**kwargs)
516          context['licenses'] = self.activation_key.licenses.all()
517          context['organization'] = self.organization
518          return context
519
```

Kuva 89. Yksittäisen aktivointiavaimen näkymäkoodi.

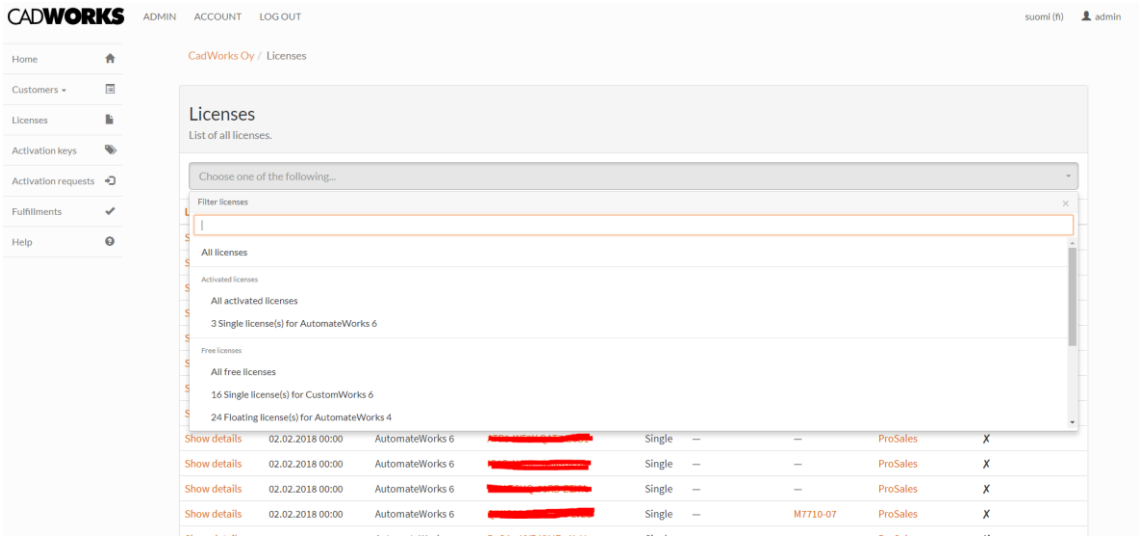
```python
class ActivationKeyDetailTable(tables.Table):
    id = tables.LinkColumn('license-details', args=[A('id')],
                           text=_("Show details"), verbose_name=_('License'))
    organization = tables.LinkColumn('licenses-organization', args=[A('organization.slug')],
                                     verbose_name=_('Organization'))

    activation_key = tables.LinkColumn('licenses-key', args=[A('activation_key.slug')],
                                       verbose_name=_('Activation key'), text=lambda lic: lic.activation_key.key)

    prosales_url = tables.TemplateColumn(
        """<a href='https://prosales.cadworks.fi/ps/order?entityId={{ record.prosales_order_id }}' target="_blank">ProSales</a>""",
        verbose_name=_('Link to ProSales'))

    def __init__(self, *args, **kwargs):
        self.__activation_key = kwargs.pop('activation_key')
        super(ActivationKeyDetailTable, self).__init__(*args, **kwargs)

    activated_on = tables.Column(accessor='activated_on_workstation')

    def render_activated_on(self, value):
        url = reverse('licenses-workstation', args=[value.pk])
        return format_html("""<a href="{}">{}</a>""".format(url, value.name))

    has_subscription = tables.BooleanColumn()

    def before_render(self, request):
        if not request.user.is_staff:
            self.columns.hide('prosales_url')
        else:
            self.columns.show('prosales_url')

    class Meta:
        model = License
        attrs = {
            'class': 'table table-hover license-table paleblue'
        }
        exclude = ('date_created',
                   'date_modified', 'serial_number', 'prosales_order_id', 'prosales_id', 'prosales_order_no',)
```

Kuva 90. Aktivointiavaimen taulumäärittely.

Taulumäärittely kuvassa 90 käyttää mallinaan License-luokkaa, ja sarakkeet toteutin lisenssinäkymän taulua mukaillen.

6.4.9 Lisenssinäkymä

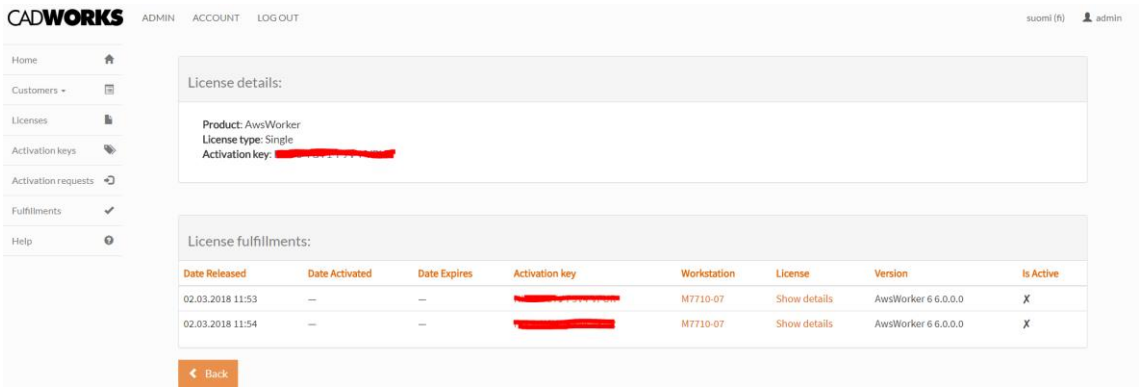
Lisenssien listaus oli näkymistä monimutkaisin. Se sisältää peruslistauksen lisäksi kuvassa 91 aktiivisena näkyvän suodattimen, joka auttaa käytössä olevien ja vapaiden lisenssien löytämisessä tuotteittain.



Kuva 91. Organisaation lisenssilista.

Lisenssin riviltä käyttäjä pääsee ensimmäisen sarakkeen linkkejä painamalla tarkastelemaan yksittäisen lisenssin toteutumistietoja kuvassa 92, eli onko lisenssi

aktiivinen vai ei. Toteutumistiedoissa näkyy myös kaikki edelliset aktivointitapahtumat. Jokaiselta riviltä pääsee myös lisenssin aktivoineen työaseman tietoihin. Järjestelmänvalvojille on erillinen sarake, joka sisältää linkin lisenssin tietoihin asiakashallintasovelluksessa. Muihin sarakkeisiin kuuluu tieto lisenssitilauksen aikaleimasta, passivoimisen, joka kuvaa lisenssin perumista, aikaleimasta ja lisenssin tyypistä.



Kuva 92. Yksittäisen lisenssin tapahtumatiedot.

Lisenssilistan näkymäkoodi kuvassa 93 on myös identtinen muiden organisaationäkymien kanssa. Näkymä perii `LicensesView` -luokan metodit, jossa toteutin sivulla näkyvän suodattimen.

```
253  class OrganizationLicensesView(OrganizationMixin, LicensesView):
254      """
255      Render list of licenses in an organization.
256      """
257
258      login_url = '/not-found/'
259      redirect_field_name = None
260      template_name = 'licenses/organization_licenses.html'
261      table_class = OrganizationLicensesTable
262
263      def get_initial_queryset(self):
264          return self.organization.licenses.all()
265
```

Kuva 93. Lisenssilistan näkymäkoodi.

Kuvan 94 lisenssinäkymän `LicensesView` funktiossa `get_context_data()` toteutetaan mallineessa käytettävät suodattimen muuttujat. Aktiivisista lisensseistä suodatetaan lisenssit tuotteittain ja ynnätään ne. Mallineen käytettäväksi annetaan aktiiviset ja vapaat lisenssit sekä edellä mainittu summa lisensseistä.

```
    def get_context_data(self):
        context = super(LicensesView, self).get_context_data()
        all_licenses = self.get_initial_queryset()
        active_licenses = [l[0] for l in Fulfillment.objects.activated().values_list('license__id')]
        q = Q(pk__in=active_licenses)

        licenses_by_product = all_licenses.values('product__name', 'product__major_version', 'type__name').annotate(
            count=Count('product')).order_by()

        context['active_licenses_by_product'] = all_licenses.filter(q).values(
            'product__name', 'product__major_version', 'type__name').annotate(count=Count('product')).order_by()

        context['free_licenses_by_product'] = all_licenses.filter(~q).values(
            'product__name', 'product__major_version', 'type__name').annotate(count=Count('product')).order_by()

        context['licenses_by_product'] = licenses_by_product
        return context
```

Kuva 94. Lisenssinäkymän suodattimen toteutus.

```
                <div class="panel-body">
                    <select class="selectpicker form-control input-lg btn-group-lg"
                            data-live-search="true" data-header="{% trans 'Filter licenses' %}"
                            data-selected-text-format="static"
                            data-size="10"
                            title="{% trans 'Choose one of the following...' %}"
                            onchange="window.location.href=this.value">
                        <option disabled selected style="..." value="Filter licenses"></option>
                        <option value="?status=all">{% trans 'All licenses' %}</option>
                        {% if active_licenses_by_product.count %}
                            <optgroup label="{% trans 'Activated licenses' %}">
                                <option value="?status=activated">{% trans 'All activated licenses' %}</option>
                                {% for l in active_licenses_by_product %}
                                    {% with count=l.count type=l.type__name product=l.product__name major=l.product__major_version %}
                                        <option value="?status=activated&type={{ type|escape }}&product={{ product|escape }}&major_version={{ major }}">
                                            {% blocktrans %}
                                                {{ count }} {{ type }} license(s) for {{ product }} {{ major }}
                                            {% endblocktrans %}
                                        </option>
                                    {% endwith %}
                                {% endfor %}
                            </optgroup>
                            <optgroup label="{% trans 'Free licenses' %}">
                                <option value="?status=free">{% trans 'All free licenses' %}</option>
                                {% for l in free_licenses_by_product %}
                                    {% with count=l.count type=l.type__name product=l.product__name major=l.product__major_version %}
                                        <option value="?status=free&type={{ type|escape }}&product={{ product|escape }}&major_version={{ major }}">
                                            {% blocktrans %}
                                                {{ count }} {{ type }} license(s) for {{ product }} {{ major }}
                                            {% endblocktrans %}
                                        </option>
                                    {% endwith %}
                                {% endfor %}
                            </optgroup>
                        {% else %}
                            <optgroup label="{% trans 'Free licenses' %}">
                                {% for l in licenses_by_product %}
                                    {% with count=l.count type=l.type__name product=l.product__name major=l.product__major_version %}
                                        <option value="?status=all&type={{ type|escape }}&product={{ product|escape }}&major_version={{ major }}">
                                            {% blocktrans %}
                                                {{ count }} {{ type }} license(s) for {{ product }} {{ major }}
                                            {% endblocktrans %}
                                        </option>
                                    {% endwith %}
                                {% endfor %}
                            </optgroup>
                        {% endif %}
                    </select>
                </div>
```

Kuva 95. Organisaation lisenssinäkymän suodattimen toteutus mallineessa.

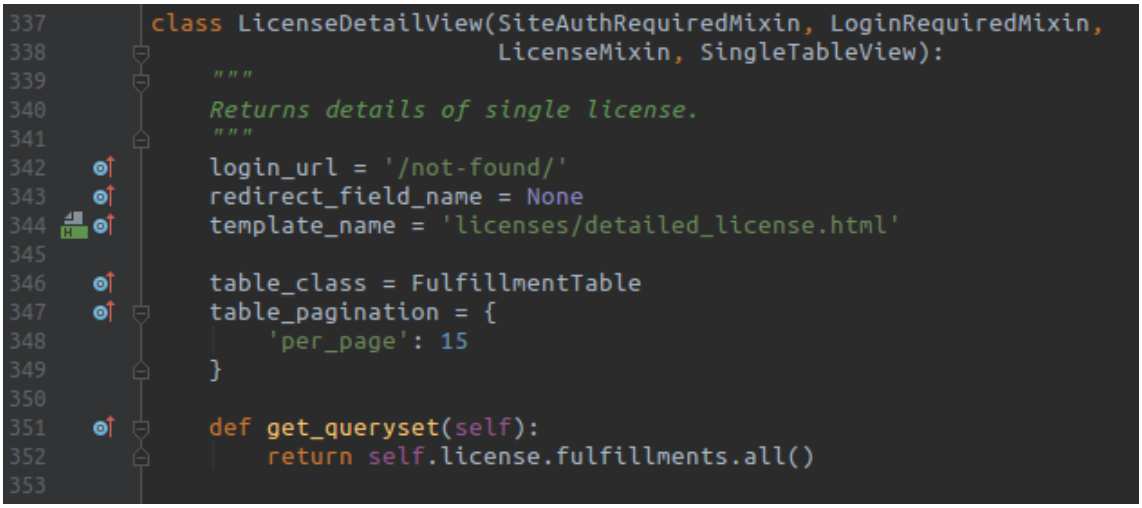
Käytin suodattimen toteutukseen kuvassa 95 avuksi Bootstrapilla luotua *selectpicker* -nimistä kirjastoa, joka tarjoaa `select`-tagille ulkoasun. Select -tagin valinnat muodostan kyselyiden tulosten perusteella. Jaan aktiiviset ja vapaat lisenssit erikseen tuotteittain. Kun käyttäjä tekee valinnan, sivu lataa tulokset uuteen listaan.

Lisenssilistan taulun määrittely kuvassa 96 koostuu useasta eri sarakkeesta. Tauluun kuuluu linkkisarakkeita sivustolle ja sen ulkopuolelle asiakashallintasovellukseen. Asiakashallintasovelluksen linkin piilottaminen toteutetaan myös taulun koodissa, koska käyttäjän oikeudet tulee tarkistaa ennen, kuin taulu renderöidään.

```
class OrganizationLicensesTable(tables.Table):
    id = tables.LinkColumn('organization-license-details', args=[A('organization.slug'), A('id')],
                           text=_("Show details"), verbose_name=_('License'))
    activated_on = tables.Column(accessor='activated_on_workstation')

    def render_activated_on(self, value):
        url = reverse('licenses-workstation', args=[value.pk])
        return format_html("""<a href="{}">{}</a>""".format(url, value.name))

    prosales_url = tables.TemplateColumn(
        """<a href='https://prosales.cadworks.fi/ps/order?entityId={{ record.prosales_order_id }}' target="_blank">ProSales</a>""",
        verbose_name=_('Link to ProSales'))

    has_subscription = tables.BooleanColumn()
    activation_key = tables.LinkColumn('licenses-organization-activation-keys-key',
                                       args=[A('organization.slug'), A('activation_key.slug')],
                                       text=lambda license: license.activation_key.key)

    def before_render(self, request):
        if not request.user.is_staff:
            self.columns.hide('prosales_url')
        else:
            self.columns.show('prosales_url')

    class Meta:
        model = License
        attrs = {'class': 'table table-hover license-table'}
        exclude = ('date_created', 'date_modified',
                   'serial_number', 'prosales_order_id', 'prosales_id',
                   'prosales_order_no', 'organization')
```

Kuva 96. Lisenssilistan taulun määrittely.

Yksittäisen lisenssin näkymäkoodi kuvassa 97 on lähes sama. Taulua sivutetaan niin, että jokaisella sivulla näkyy 15 riviä. Kyselyllä haetaan lisenssin toteutumistiedot.



```
class LicenseDetailView(SiteAuthRequiredMixin, LoginRequiredMixin,
                        LicenseMixin, SingleTableView):
    """
    Returns details of single license.
    """
    login_url = '/not-found/'
    redirect_field_name = None
    template_name = 'licenses/detailed_license.html'

    table_class = FulfillmentTable
    table_pagination = {
        'per_page': 15
    }

    def get_queryset(self):
        return self.license.fulfillments.all()
```

Kuva 97. Yksittäisen lisenssin näkymäkoodi.

Taulu kuvassa 98 määräytyy toteutumistaulusta Fulfillment. Taulun tarkoitus on näyttää lisenssin aktivointitapahtumat. Se sisältää tiedot lisenssistä, avaimesta, sekä työasemasta, jolla lisenssi on aktiivinen. Tauluun kuuluu myös totuusarvo `is_active`, jolla voidaan tarkastaa, onko lisenssi aktiivinen työasemalla.

```python
class FulfillmentTable(tables.Table):
    license = tables.LinkColumn('license-details', args=[A('license.id')],
                                text=_("Show details"), verbose_name=_('License'))

    activation_key = tables.LinkColumn('licenses-key', args=[A('activation_key.slug')],
                                       text=lambda license: license.activation_key.key,
                                       verbose_name=_('Activation key'))

    workstation = tables.LinkColumn('licenses-workstation', args=[A('workstation.pk')],
                                    text=lambda license: license.workstation.name, verbose_name=_('Workstation'))

    is_active = tables.BooleanColumn(accessor='is_activated')

    class Meta:
        model = Fulfillment
        attrs = {
            'class': 'table table-hover license-table paleblue'
        }
        exclude = ('id', 'date_created', 'date_modified', 'hostid')

```

Kuva 98. Yksittäisen lisenssin käyttämä tapahtumataulu.

Lisenssin mallinne kuvassa 99 käyttää `panel` -tagia. Otsikkoon listataan lisenssin tuote, tyyppi sekä lisenssiin yhdistetty aktivointiavain. Tapahtumataulu renderöidään alle. Valmis lisenssisivu on esitetty kuvassa 92. Lisenssin tiedot näkyvät erillisellä paneelilla, jolloin selaaminen on helppoa.

```
{% block title %}
    {{ organization.name }}
{% endblock %}
{% block content %}
    <div class="main">
        <div class="page-header">
            <h1>{{ workstation.name }}</h1>
            <div class="panel panel-default">
                <div class="panel-heading">
                    <h3 class="text-muted">
                        {% trans 'License details:' %}
                    </h3>
                </div>
                <br>
                <ul style="...">
                    <li class=""><b>{% trans 'Product' %}</b>: {{ license.product.name }}</li>
                    <li><b>{% trans 'License type' %}</b>: {{ license.type }}</li>
                    <li><b>{% trans 'Activation key' %}</b>:
                        <a href="{% url 'licenses-organization-activation-keys-key' license.organization.slug license.activation_key.slug %}">{{ license.activation
                    </li>
                </ul>
                <br>
            </div>
        </div>
        <br>
        {% if table %}
            <div class="table-responsive">
                {% csrf_token %}
                {% load render_table from django_tables2 %}
                <div class="panel panel-default">
                    <div class="panel-heading">
                        <h3 class="text-muted">{% trans 'License fulfillments:' %}</h3>
                    </div>
                    {% render_table table %}
                </div>
            </div>
            <div>
                {% block backbutton %}
                    {% if organization %}
                        <a class="btn btn-lg cadworks-button"
                           href="{% url 'licenses-organization-licenses' organization.slug %}">
                            <span class="glyphicon glyphicon-chevron-left"></span>
                            {% trans 'Back' %}
                        </a>
                    {% else %}
                        <a class="btn btn-lg cadworks-button" href="{% url 'all-licenses' %}">
                            <span class="glyphicon glyphicon-chevron-left"></span>
                            {% trans 'Back' %}
                        </a>
                    {% endif %}
                {% endblock %}
            </div>
        {% else %}
            <p class="text-info">{% trans 'Error loading table data. Please contact CadWorks administration.' %}</p>
        {% endif %}
    </div>
{% endblock %}
```

Kuva 99. Yksittäisen lisenssin mallinne.

7 YHTEENVETO

Molemmat Python ja Django olivat projektin alussa itselle uusia maailmoja. Projektin aikana opin erilaisiin ongelmiin nokkelia ja järkeviä ratkaisuja, jotka olisivat vaatineet toisenlaisella ohjelmointikielellä paljon enemmän aikaa ja pään raapimista. Sivuston rakentamiseen olin jo oppinut hyvät perusteet, mutta tähän projektiin niitä pääsi vasta todella soveltamaan. Opin myös todella paljon uusia ratkaisuja HTML:n ja CSS:n osalta, joita pääsin vapaasti kokeilemaan projektissa. Django-sovelluksen kehitystyössä ei ole mielestäni suoraan selkeää rajaa asiakas- ja palvelinohjelmoinnin välillä. Vaikka työni keskittyi enimmäkseen asiakaspuolen kehitykseen, työhön sulautui samalla hyvin paljon Djangon palvelinpuolta, mikä on mielestäni projektin kehityksen kannalta luonnollista. Ilman tietokantakyselyitä näkymillä ei ole mitään virkaa tässä sovelluksessa.

Palvelinpuolen ohjelmoinnin kehitystyö hidasti ajoittain asiakaspuolen webkehitystä, sillä ongelmat johtuivat usein palvelinpuolen loogisista virheistä. Sivustolla on tälläkin hetkellä vielä näkymäkohtaisia virheitä. Sovelluksen kehityksessä tulee ottaa erittäin tarkkaan huomioon peruskäyttäjän ja järjestelmänvalvojan erot, ja näkymät tulisi ohjelmoida niin, että näkymissä ei ole liikaa tai liian vähän tietoa, ja mahdolliset ongelmatilanteet käsiteltäisiin oikein. Sivuston ulkoasua ei myöskään suunniteltu alusta alkaen kunnolla, joten sain sen käytännön toteutukseen vapaat kädet. Tyyli muuttui hyvin paljon sen kehityksen aikana. Sivusto kaipaisi mielestäni käyttäjälle selkeää ja mielekästä ulkoasua, mikä sinänsä onnistui, mutta sivulla ei käytetä mitään monimutkaisia animointeja tai kuvia sivun elävöittämiseen. Sivusto voi olla jonkun mielestä tylsä, koska sisällössä keskitytään ainoastaan tietokannan tietoihin. Ulkoasua voidaan kehittää tietysti vielä jälkeenpäin.

Olen mielestäni kehittynyt todella paljon ongelmien ratkaisemisessa ja usean eri osa-alueen hallitsemisessa, johtuen juuri siitä, että olen joutunut miettimään sekä palvelin- että asiakaspuolen ongelmia. Olen erittäin tyytyväinen tällä hetkellä myös siihen, miten koodi on saatu pysymään siistinä ja turhilta toistoilta on vältytty. Olen oppinut kollegoiltani myös sen, että koodin tulee olla syntaksiltaan selkeää ja muiden tiimin jäsenten helposti luettavissa. Samaan kategoriaan kuuluu myös dokumentoinnin tärkeys. Sovellusta tullaan kehittämään ja parantamaan vuosia, joten koodissa ja sen ulkopuolella tulee olla selkeät ohjeet siihen, mitä jokin koodin osa tekee.

Tekemäni sovellus on noussut sisäiseen käyttöön hyvin nopeasti. Asiakkaiden käyttöön sovellusta ollaan testaamassa vaihtamalla kaikessa hiljaisuudessa käyttäjien huomaamatta vanhan lisenssien aktivoinnin logiikka Django-sovelluksen uuteen logiikkaan, ja sovelluksella on jo muutama koekäyttäjä.

Projektia kehittää minun lisäksi toinen jäsen, ja olemme saaneet todella paljon aikaiseksi melko lyhyessä ajassa. Kehitystyö jatkuu edelleen keskittyen loogisiin, lisensseihin liittyviin ongelmiin, ja näkymien parantelu on toissijainen tehtävä.

Projekti on ollut ns. hybridisovellus, jonka runko soveltuu erinomaisesti myös uuden tuotteemme kehitykseen.

LÄHTEET

Django Softwaare Foundation. (2018, 3 2). *Base views*. Haettu osoitteesta Django documentation: https://docs.djangoproject.com/en/2.0/ref/class-based-views/base/#django.views.generic.base.TemplateView

Django Software Foundation. (2017, 6 20). *Django overview*. Haettu osoitteesta Django: https://www.djangoproject.com/start/overview/

Django Software Foundation. (2017, 8 17). *FAQ: General*. Haettu osoitteesta Django documentation: https://docs.djangoproject.com/en/1.11/faq/general/#django-appears-to-be-a-mvc-framework-but-you-call-the-controller-the-view-and-the-view-the-template-how-come-you-don-t-use-the-standard-names

Django Software Foundation. (2018, 2 22). *Built-in template tags and filters*. Haettu osoitteesta https://docs.djangoproject.com/en/2.0/ref/templates/builtins/#include

Django Software Foundation. (2018, 3 1). *Cross Site Request Forgery protection*. Haettu osoitteesta Django documentation: https://docs.djangoproject.com/en/2.0/ref/csrf/#module-django.views.decorators.csrf

Django Software Foundation. (2018, 3 1). *Error reporting*. Haettu osoitteesta Django documentation: https://docs.djangoproject.com/en/2.0/howto/error-reporting/#filtering-sensitive-information

Django Software Foundation. (2018, 2 22). *Managing static files (e.g. images, JavaScript, CSS)* . Haettu osoitteesta https://docs.djangoproject.com/en/2.0/howto/static-files/

Django Software Foundation. (2018, 3 1). *View decorators*. Haettu osoitteesta Django documentation: https://docs.djangoproject.com/en/2.0/topics/http/decorators/

GitHub, Inc. (2018, 2 15). *Highlights from the past twelve months*. Haettu osoitteesta GitHub Octoverse 2017: https://octoverse.github.com/

Holovaty, A., & Kaplan-Moss, J. (2009). *The Definitive Guide to Django: Web Development Done Right, Second Edition.* New York: Springer-Verlag New York, Inc.

Hussain, A. (2018, 2 15). *The 9 Most In-Demand Programming Languages of 2017*. Haettu osoitteesta Linkedin: https://www.linkedin.com/pulse/9-most-in-demand-programming-languages-2017-abuzhar-hussain/

IEEE. (2018, 2 15). *The 2017 Top Programming Languages* . Haettu osoitteesta IEEE Spectrum: https://spectrum.ieee.org/computing/software/the-2017-top-programming-languages

Korpela, J. (2017, 7 13). *Gemena ja versaali ("pienet" ja "isot" kirjaimet)[Nykyajan kielenopas]*. Haettu osoitteesta Datatekniikka ja viestintä: http://www.cs.tut.fi/~jkorpela/kielenopas/6.2.html#karavaani

Lutz, M. (2011). *Programming Python, Fourth Edition.* Sebastopol: O'Reilly Media, Inc.

Lutz, M. (2013). *Learning Python, Fifth Edition.* Sebastopol: O'Reilly Media, Inc.

Python Software Foundation. (2017, 7 20). *5.4 Numeric Types - int, float, long, complex*. Haettu osoitteesta Python 2.7.13 documentation: https://docs.python.org/2/library/stdtypes.html#numeric-types-int-float-long-complex

Python Software Foundation. (2017, 7 18). *Built-in Types*. Haettu osoitteesta Python 2.7.13 documentation: https://docs.python.org/2/library/stdtypes.html#string-methods

Python Software Foundation. (2017, 7 27). *Data Structures*. Haettu osoitteesta Python 2.7.13 documentation: https://docs.python.org/2/tutorial/datastructures.html#dictionaries

Real Python. (2018, 2 16). *Asynchronous Tasks With Django and Celery*. Haettu osoitteesta Real Python: https://realpython.com/blog/python/asynchronous-tasks-with-django-and-celery/#what-is-celery

TIOBE Software BV. (2017, 7 13). *TIOBE Index*. Haettu osoitteesta TIOBE - The Software Quality Company: https://www.tiobe.com/tiobe-index/

TutorialsPoint. (2017, 7 31). *Python Functions*. Haettu osoitteesta https://www.tutorialspoint.com/python/python_functions.htm

TutorialsPoint. (2017, 7 18). *Python Variable Types*. Haettu osoitteesta TutorialsPoint: https://www.tutorialspoint.com/python/python_variable_types.htm

TutorialsPoint. (2018, 2 22). *Responsive Web Design Viewport*. Haettu osoitteesta https://www.w3schools.com/cSS/css_rwd_viewport.asp

W3Schools. (2018, 2 23). *CSS Media Queries*. Haettu osoitteesta https://www.w3schools.com/css/css3_mediaqueries.asp

Van Rossum, G. (2009, January). *The History of Python: Overview and Introduction*. Haettu osoitteesta Blogspot: http://python-history.blogspot.fi/2009/01/introduction-and-overview.html

Vihavainen, A., Paksula, M., Luukkainen, A., Mikkola, P., Laurinharju, J., & Pärtel, M. (2017, 7 15). *Muuttuja ja sijoitus*. Haettu osoitteesta Ohjelmoinnin perusteet Python-kielellä: https://www.cs.helsinki.fi/group/linkki/materiaali/python-perusteet/#5