jamk

Laadunvarmistussovellus sähkökeskuksien valmistuksessa

Mari Kolu-Lukkarinen

Opinnäytetyö, AMK
Syyskuu 2022
Tietojenkäsittely ja tietoliikenne
Insinööri (AMK), Tieto- ja viestintätekniikan tutkinto-ohjelma

jamk | Jyväskylän ammattikorkeakoulu
University of Applied Sciences

jamk

Kuvailulehti

Kolu-Lukkarinen, Mari

Laadunvarmistussovellus sähkökeskuksien valmistuksessa

Jyväskylä: Jyväskylän ammattikorkeakoulu. Syyskuu 2022, 34 sivua

Tietojenkäsittely ja tietoliikenne. Insinööri (AMK), Tieto- ja viestintätekniikan tutkinto-ohjelma. Opinnäytetyö AMK.

Julkaisun kieli: suomi

Julkaisulupa avoimessa verkossa: kyllä

Tiivistelmä

Opinnäytetyön toimeksiantajana toimi Pasram Oy, joka on jyväskyläläinen sähkö- ja automaatiosuunnitteluun erikoistunut yritys. Yrityksessä on kehitetty varastokirjanpito web-sovellus, johon toimeksiantaja halusi lisätä sähkökeskusten laadunvarmistuksen. Web-sovelluksen tarkoitus on luovuttaa asiakkaalle tasalaatuisia sähkökeskuksia, joihin liittyvät dokumentit ovat kaikkien saatavilla.

Opinnäytetyön toteutustavaksi valikoitui soveltava tutkimus. Tutkimuksessa oli tavoitteena tutkia, että saadaanko ennalta valituilla teknologioilla aikaan toimiva web-sovellus. Tutkimuksen lähteenä toimivat teknologioiden dokumentaatiot sekä muut kirjalliset lähteet.

Soveltavassa osassa tuli ottaa huomioon web-sovelluksen jatkokehitysmahdollisuudet ja suunnittelun tärkeys ennen ohjelmoinnin aloittamista. Web-sovellus toteutettiin suositulla Django web-kehyksellä, joka on kehitetty Python-ohjelmointikieltä käyttäen.

Opinnäytetyön tekemisen aikana todettiin, että huolellisella suunnittelulla voidaan estää turhan työn tekeminen kaikkein parhaiten. Tuloksina saatiin selville, että ennalta valitut teknologiat sopivat web-sovellusten kehittämiseen ja jatkokehittäminen on helppoa tulevaisuudessa.

Toimeksiantaja oli tyytyväinen web-sovelluksen lopputulokseen ja sovelluksen jatkokehittäminen on jo suunnitelmissa. Web-sovellus täytti melkein kaikki toimeksiantajan vaatimukset ja valmiiden sähkökeskusten dokumentit ovat halutulla tavalla tallessa kaikkien saatavilla.

Avainsanat (asiasanat)

Web-sovellus, Django, laadunvarmistus, Python

Muut tiedot (salassa pidettävät liitteet)

-

jamk

Description

Kolu-Lukkarinen, Mari

Quality assurance web application in electrical cabinet manufacturing

Jyväskylä: JAMK University of Applied Sciences, September 2022, 34 pages

Information and Communication Technologies. Bachelor's Degree Programme in Information and Communication Technology. Bachelor´s thesis.

Permission for open access publication: Yes

Language of publication: Finnish

Abstract

The thesis was assigned by Pasram Oy, a company specializing in electrical and automation design. Pasram has developed an inventory management web application and the company wanted to add quality assurance for electrical cabinets to the application. The purpose of the web application is to hand over electrical cabinets of equal quality to customer and have all related documents available to all parties.

Applied research was chosen as research method. The aim of the study was to investigate whether a working web application can be created with pre-selected technologies. Documentation of technologies and other written sources serve as the source of the research.

The applied part had to take into account the further development possibilities of the web application and the importance of planning before starting programming. The web application was implemented with the popular Django web framework, which was developed using Python programming language.

During the research it was found that careful planning is the best way to prevent unnecessary work. The results revealed that the pre-selected technologies are suitable for developing web applications and further development will be easy in the future.

The client was satisfied with developed application and further development is already planned. The web application met almost all of the client's requirements, and the documents of the finished electrical cabinets are saved as desired and are available to everyone.

Keywords/tags (subjects)

Web application, Django, quality assurance, Python

Miscellaneous (Confidential information)

-

jamk

KIITOKSET

Haluan kiittää kaikkia niitä henkilöitä, jotka ovat tukeneet, kannustaneet ja mahdollistaneet tämän opinnäytetyön valmistumisen. Kiitokset ystävilleni, naapureilleni ja sukulaisilleni kaikesta tuesta, kannustuksesta ja myötäelämisestä. Loitte minuun uskoa, kun vaivuin epätoivoon kaiken kiireen keskellä.

Isot kiitokset perheelleni, jotka joutuivat tekemään itse ruokansa ja pesemään pyykkinsä. Kiitos aviomieheni Heikki Lukkarinen, kun veit minut veneilemään, jotta saisin muuta ajateltavaa. Kiitos lapset, Wiivi, Veeti ja Venla, kun olette antaneet äidin kirjoittaa ja olla rauhassa, kun olen sitä tarvinnut ja höpöttäneet niitä näitä saadaksenne huomiota.

Suuret kiitokset työkavereilleni Pasram Oy:ssä. Kiitos erityisesti esihenkilöni Pasi ja Rami Karell, että annoitte minulle mahdollisuuden tehdä opinnäytetyön ja mahdollistitte valmistumiseni aikataulussa. Kiitos ohjaajani Antti Koivurova kaikista niistä rakentavista palautteista ja kannustuksista. Kiitos Tatu Honkanen, kun autoit mahdottomilta tuntuvissa ongelmakohdissa. Kiitokset vielä Juhamatti Leppänen ja Tuomas Eronen, että kannustitte puskemaan eteenpäin.

Puuppolassa 11.7.2022 Mari Kolu-Lukkarinen

Sisältö

Kuviot

Taulukot

1 Johdanto

1.1 Toimeksiantaja ja tausta

Opinnäytetyön toimeksiantajana toimi jyväskyläläinen sähkö- ja automaatioyritys Pasram Oy. Yrityksen ovat perustaneet veljekset Pasi ja Rami Karell syksyllä 2012. Pasram Oy työllistää noin 30 henkilöä, jotka jakaantuvat sähkö-, automaatio- ja ohjelmistosuunnittelijoihin, sähköasentajiin ja hallinnollisiin työntekijöihin.

Toimeksiantaja halusi sähkökeskuksien toiminnan ja laadunvarmistukseen helposti käytettävän ja nopean sovelluksen. Sovelluksen tarve ilmeni siten, että haluttiin tietää millainen ulkonäöltään ja tiedoiltaan jo asiakkaalle luovutettu sähkökeskus on luovutettu asiakkaalle. Haun tuli olla nopea, selkeä ja yhdenmukainen. Toimeksiantaja halusi, että sovellusta voidaan myöhemmässä vaiheessa kehittää lisää.

1.2 Tutkimuksen tavoite

Tämän opinnäytetyön tavoitteena oli tehdä sähköasentajille web-sovellus, jolla voidaan varmistaa sähkökeskuksen laatu ja toiminta, ennen kuin se luovutetaan asiakkaalle. Sovelluksella voidaan tarkistaa, kuka kytkennät on tehnyt, milloin ja miten. Henkilökohtaisia tavoitteita oli oppia enemmän web-sovellusten suunnittelusta ja toteutuksesta.

1.3 Tutkimusongelma ja -menetelmä

Usein kuulee kysyttävän, että onko siitä keskuksesta kuvia? Ja yhtä usein kuulee vastauksen, että on ne jossain, täytyy etsiä ne. Dokumentaatiota on paljon, mutta ne on ripoteltu moneen paikkaan, joihin kaikkiin ei sähköasentajilla ole pääsyä. Yhtenä tutkimusongelmana voidaan siis pitää dokumentaation pirstaleisuutta.

Toinen tutkimusongelma on laadun valvonta ja sen parantaminen, koska usein halutaan tietää mitkä ovat viimeisimmät sähkökuvat keskuksesta. Tällä hetkellä tiedot kyllä löytyvät, mutta ne haluttaisiin löytää nopeammin ja helpommin siten, että ne ovat nähtävillä jokaisella sähköasentajalla ja -suunnittelijalla.

Tutkimusmenetelmänä opinnäytetyössä käytettiin soveltavaa tutkimista, jossa suurin painoarvo on käytännön ongelman ratkaisussa ennalta valituilla teknologioilla. Tilastokeskuksen (N. d.) mukaan soveltavalla tutkimuksella tarkoitetaan toimintaa, joka on tehty uuden tiedon saamisesta ja sen päämääränä on tuon tiedon soveltaminen käytännössä.

Tutkimuskysymykset, joihin opinnäytetyössä pyrittiin vastaamaan, olivat

1. Saadaanko käytettävillä teknologioilla aikaan laadun parantuminen?
2. Miten jatkokehittäminen tulee ottaa huomioon web-sovellusta tehtäessä?
3. Kuinka tärkeää on suunnitella web-sovellus ennen varsinaisen työn aloittamista?
4. Kuinka rajata toteutus opinnäytetyöhön sopivaksi?

Tutkimustuloksina käytettiin käyttäjiltä saatua tietoa, joka koostui heidän käytännöistä ja kokemuksista. Tätä saatua tietoa tuki myös toimeksiantajalta saatu ammattitaito ja tieto yhdistettynä itse kerättyyn materiaaliin. Lähteistä saatavaan tietoon tuli suhtautua kriittisesti ja niitä täytyi pohtia systemaattisesti.

2 Valitut teknologiat

2.1 Valintaperusteet

Opinnäytetyön sovellus on osa suurempaa varastokirjanpitokokonaisuutta, johon oli valittu teknologiat ja työkaluksi valmiiksi. Tämän vuoksi teknologioiden ja työkalujen vertailua ei opinnäytetyötä varten tarvinnut tehdä.

Sovelluskehykseksi valikoitui Django Web Framework, koska se on nopea, skaalautuva ja siihen löytyy laaja dokumentaatio. Django Web Framework toimi hyvin yhteen muiden valittujen teknologioiden kanssa.

Tietokannaksi valikoitui PostgreSQL, koska se pystyy käsittelemään isoa määrää tietoa ja se on helppokäyttöinen ja nopea. Valintaan vaikutti myös sen suosio, laaja dokumentaatio ja aktiivinen kehittäminen.

Tailwind CSS Framework valittiin tähän projektiin siitä syystä, ettei haluttu tehdä samanlaista toteutusta kuin muilla tekemällä Bootstrapilla. Myös Tailwind CSS:stä löytyy myös laaja dokumentaatio ja iso määrä esimerkkejä internetistä.

2.2 Django Web Framework

Django Web Frameworkin (myöhemmin Django) ovat kehittäneet yhdysvaltalaisen sanomalehden, Lawrance Journal-Worldin, web-ohjelmoijat Adrian Holovaty ja Simon Willison, myöhemmin mukaan liittyi myös Jacob Kaplan-Moss. Kehitystyö on aloitettu vuoden 2003 syksyllä ja avoin lähdekoodi julkaistiin heinäkuussa vuonna 2005. Django on nimetty Jazz-kitaristi Django Reinhardtin mukaan. (Behrens 2012.)

Djangon hyviä puolia ovat sen nopeus, ilmainen avoin lähdekoodi, dokumentaatio, tietoturvallisuus ja skaalautuvuus. Django huolehtii monista asioista niin hyvin, ettei ohjelmoijan tarvitse niihin puuttua, kuten autentikointiin tai tietoturvallisuuteen. Tällöin ohjelmoija saa käyttää kaiken ajan suunnitteluun, koodin kirjoittamiseen ja dokumentoimiseen. (Big 2022, Lesson 1.)

Djangosta on tehty paljon opiskelumateriaalia, esimerkiksi Youtube-videoita, ilmaiskursseja tai suurempia oppimiskokonaisuuksia. Djangon kehittämisessä on alusta asti otettu huomioon se, että se on helppo oppia ja melkein kaikesta löytyy dokumentaatiota ja ohjeita (Mts. Lesson 1).

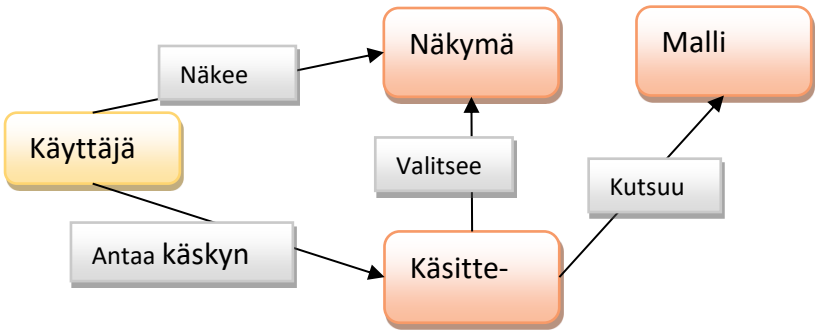
Djangon kehitystyö on aktiivista, esimerkiksi noin kolmen vuoden välein julkaistaan Long Term Support (LTS) versio eli versio, joka on vakaa ja sillä on pitkä tuki. Django 3.2 on tuettu ainakin vuoden 2024 puoliväliin saakka. Tällä hetkellä uusin versio 4.0 on julkaistu joulukuussa 2021. (Mts. Lesson 1.)

Djangoa käytävät monet isot organisaatiot kuten Washington Post, Pinterest, Instagram, Spotify, YouTube, DropBox ja Mozilla. Syyksi moni näistä organisaatioista ilmoittaa Djangon nopeuden ja skaalautuvuuden heidän tarpeisiinsa sopivaksi. Täytyy muistaa, että Django on alun perin kehitetty sanomalehteä varten, joten esimerkiksi artikkeleiden tekeminen on otettu erityisen hyvin huomioon. (Korsun 2016)

Djangossa on hienosti jaettu selain- ja palvelinpuoli, jolloin käyttöliittymää voi tehdä isossakin tiimissä helposti. Web-suunnittelijan, joka suunnittelee html-sivuja ei välttämättä tarvitse tietää mitään Djangosta ja toisinpäin käännettynä ohjelmoijan ei tarvitse osata sivujen ulkoasun luomista tai muokkausta. (Mts. Lesson 1.)

Django käyttää ohjelmointikielenä Pythonia, joka on helppo oppia, sopii monelle laitteelle, kolmanneksi suosituin kieli ja sen paketit tarjoavat kaiken mahdollisen mitä vain tarvitsee Web-sovelluksen kehittämisessä (mts. Lesson 1). Python ohjelmointikielenä on selkeä ja helppolukuinen, ilmainen avoin lähdekoodi ja se on helposti laajennettavissa erilaisilla paketeilla (BeginnersGuide/Overview 2019).

Kuviossa 1 on näkyvillä yleinen käsite MVC-arkkitehtuurista. Malli (Model) sisältää käyttöliittymän tietokantakuvauksen eli tiedon tallentamisen, ylläpidon ja käsittelyn. Näkymä (View) määrittelee käyttöliittymän ulkoasun, mitä käyttäjälle näytetään ja mitä tietoja häneltä kerätään. Käsittelijä (Controller) ottaa vastaan käyttäjän käskyjä ja sen perusteella toimii tiedonvälittäjänä mallin ja näkymän välillä.



Kuvio 1. Yleinen käsitys MVC-arkkitehtuurista

Djangon MVC-arkkitehtuuri poikkeaa hieman yleisestä käsityksestä. Siinä käytetään nimityksiä Model (Malli), Template (Sivupohja) ja View (Näkymä). Djangossa käytetään URL-määritystä, jonka avulla ohjataan oikea näkymä oikeaan sivupohjaan. (Mts. Lesson 2.)

Malleja luodessa kehittäjän ei tarvitse tietää tietokannan rakenteesta juurikaan mitään, koska Django osaa hoitaa tämän puolen erittäin hyvin. Django luo tietokannan, yhteydet tietokantataulujen välille ja varsinaisia SQL-komentoja ei tarvitse käyttää. Kehittäjän tarvitsee vain valita käytettävä tietokanta ja mahdolliset avainparit (Primary Key). Django tukee seuraavia tietokantoja

PostgreSQL, MySQL, MariaDB, SQLite ja Oracle. PostgreSQL on näistä suosituin ja SQLiteä Django käyttää automaattisesti, mutta sitä ei kannata koskaan käyttää valmiissa sovelluksessa. Djangoon voi halutessaan yhdistää myös muita tietokantoja, mutta ne vaativat jo hieman enemmän asentamista ja yhteensopivuuden varmistamista. (Mts. Lesson 2.)

Sivupohjat (Templates) ovat yleensä HTML-tiedostoja, mutta ne voivat olla muistakin tekstitiedostoja. Sivupohjan tulisi erottaa suunnittelu ja ohjelmalogiikka toisistaan, estää toistaminen ja olla tietoturvallinen ja ei sisällä koodin suorittamista.

Djangossa tehdään peruspohja sovelluksen sivulle, johon muut sivut pohjautuvat. Näin varmistetaan sivujen yhteneväinen ulkoasu ja asioiden toistaminen. Peruspohjaa käytetään siten, että muutetaan vain tarvittavaa osiota sivusta (kts. Kuvio 2). Peruspohjassa olevia osioita voi tarpeen mukaan pilkkoa vielä pienemmiksi. (Mts. Lesson 2.)

Kuvio 2. Esimerkki Django-sovelluksen sivun peruspohjasta.

2.3 PostgeSQL-tietokanta

PostgeSQL on yli 20 vuoden kehitystyön tulos ja kehitys jatkuu yhä. Ja se on nykyään yksi edistyneimmästä avoimen lähdekoodin tietokannasta. PostgreSQL on alun perin lähtöisin Kalifornian yliopistossa Berkleyssä kirjoitetusta POSTGRES-projektista, joka on aloitettu vuonna 1986 professori Michael Stonebreakerin johdolla. Vuonna 1994 POSTGRES-projektiin lisäsivät Andrew Yu ja Jolly Chen SQL-kielen tulkin ja vuonna 1995 nimeksi tuli Postgres95. Vuonna 1996 julkaistiin PostgreSQL 6.0, ja tällä hetkellä uusin julkaisu on PostgreSQL 14. (PostgreSQL 14.4 Documentation. N. d.)

DeBarros (2018) listaa kirjassaan PostgreSQL:n hyviä puolia seuraavasti:

- ilmainen, avoin lähdekoodi
- saatavilla Windows-, macOS- ja Linux-käyttöjärjestelmille
- SQL-toteutus noudattaa tarkasti ANSI-standardeja
- laajasti käytössä, eli avun löytäminen internetistä on helppoa
- saatava useissa versioissa, jotka keskittyvät suurien tietomäärien käsittelyyn, esimerkiksi Amazon Redshift ja Greenplum
- yleinen valinta websovelluksille, esimerkiksi Django- ja Ruby on Rails-framework

PostgreSQL:n asentaminen ja käyttöönotto on tehty helpoksi ja se on selostettu tarkemmin luvussa Tietokanta 3.1.1. PostgreSQL voi käyttää suoraan komentoriviltä tai asentamalla ilmaisen, graafisen käyttöliittymän, pgAdmin, joka on käyttäjäystävällisempi.

Opinnäytetyön sovelluksessa käytettiin Djangoa, joka automaattisesti luo tarvittavat taulut ja niiden väliset yhteydet. Tämä helpotti tietokannan käyttöä, koska tietokannan luomista lukuun ottamatta yhtään SQL-kieltä ei tarvinnut käyttää.

2.4 TailwindCSS framework

TailwindCSS:n (myöhemmin Tailwind) ovat kehitelleet Adam Wathan ja Steve Schoger, jotka julkaisivat Tailwindin marraskuussa vuonna 2017. Tailwindin menestys perustuu siihen, että se on enemmän CSS-apukirjasto eikä täydellinen käyttöliittymäpaketti kuin Bootstrap. Tailwind on pyrkinyt muuttamaan ohjelmistokehittäjien ajattelutapaa verkkosovellusten tyylistä toiminnallisemmaksi. (Wathan 2018.)

Tailwind on oman dokumentaationsa (2022) mukaan nopea, joustava ja luotettava. Tailwindissä ei ole ennalta määriteltyjä komponentteja, kuten esimerkiksi painikkeita tai ilmoituspalkkeja, vaan kaikki luodaan itse, kuten painikkeet, otsikot tai pudotusvalikot. Daniel Katungin (2021) mielestä voi olla etua tai haittaa siitä, että kaikki on luotava itse. Toisaalta sivuston tai sovelluksen ulkonäöstä saadaan hyvinkin yksilöllinen ja erottuva, kun kaikki luodaan itse. Valmiina Talwindissä on joitakin perus-, komponentti- ja apuohjelmamäärittelyjä, kuten väriluokkia tai skaalautuvuus eri näytöille. Käytännössä Tailwindiä käytettäessä HTML-tiedostossa tyylittely kirjoitetaan luokkaan. (Chu 2021.)

Otsikon muotoilua varten otsikko teksti laitetaan div-osion sisälle (kts. Kuvio 3). Otsikon muotoilu tapahtuu kirjottamalla div-osion luokkaan halutut tyylittelyt, kuten isompi fontti, keskitetty teksti ja marginaalit.

```
<div class="text-xl text-center font-semibold pb-2 m-4">
    <h1>{{cabinet_name}} </h1>
</div>
```

Kuvio 3. Otsikon muotoileminen Tailwindin luokkia käyttäen

Tailwindin lokaalisti tallennettavaan konfigurointitiedostoon (styles.css) voidaan määritellä itse luotu luokka (Kts. Kuvio 4.). Ensin annetaan luokalle nimi, jonka jälkeen määritellään halutut tyylit esimerkiksi painikkeelle. Esimerkiksi kaikkien painikkeiden taustavärin vaihtaminen on nopeaa tällä tavalla.

```
.btn{
    @apply font-semibold bg-primary min-w-fit text-black py-2 px-4 hover:bg-opacity-80;
}
```

Kuvio 4. Painikkeen muotoilu konfiguraatiotiedostossa

Tailwindin voi ottaa käyttöön monella eri tavalla. Taulukossa 1 on esitelty niistä kolme ja niiden eroavaisuuksia (Katungi 2021; Wathan 2022). Opinnäytetyön web-sovelluksen teossa on käytetty kolmatta tapaa, koska ei haluttu olla riippuvaisia internet-yhteydestä ja projektissa ei tarvita NodeJS:ä missään muualla.

Taulukko 1. Tailwindin käyttötavat

Tapa	Hyvät puolet	Huonot puolet
CDN, sisällönjakoverkko (Content Delivery Network)	Yleisin Tarvitaan vain linkki html-sivuun, jossa on tarvittava tyylitiedosto Nopean siirron toteuttaa sisällön tallentuminen välimuistiin	Ei voi käyttää kolmannen osapuolen tarjoamia laajennuksia. Ei voi poistaa tarpeettomia tiedostoja. Joidenkin komponenttien kohdalla ei ole muokkausmahdollisuutta.
npm-käyttö	Yleinen JavaScript Framework käyttää samantapaista lähestymistapaa.	Vaatii npm-pakettien asennuksen eli NodeJS:n.
CLI Standalone	Ei tarvitse ylimääräisiä npm-paketteja. Suoritetaan itsenäisesti omassa tiedostossa.	Ei voi käyttää kaikkia ominaisuuksia.

3 Sovelluksen toteutus

3.1 Toteutuksen suunnittelu

Sovelluksen toteuttaminen aloitettiin huolellisella suunnittelulla, joka sisälsi palavereja käyttäjien ja toimeksiantajan kanssa. Palavereiden ansiosta saatiin hyvä kokonaiskuva siitä mitä sovellukselta haluttiin. Ennen varsinaista ohjelmointia käytettiin paljon aikaa suunnitteluun, kuten vaatimusmäärittelyyn, tietokantaan, sovelluksen ulkoasuun ja prototyyppiin.

Sovelluksen tehtävienhallintaan käytettiin JIRA-tehtävienhallintaohjelmistoa (myöhemmin Jira). Jira on yleisesti Pasram Oy:ssä käytössä ja entuudestaan tuttu ohjelmisto. Jiraan luotiin projekti nimeltään Keskusten laadunvarmistus, jonka tehtävälistalle (backlog) tehtiin tehtäviä. Tehtäviä luotiin sen mukaan, kun uusia asioita tuli vastaan. Jiraa käyttämällä saavutettiin sovelluksen johdonmukainen valmistuminen, koska tehtävät voitiin asettaa tärkeysjärjestykseen.

3.2 Vaatimusmäärittely

Matti Vuori (2009) kuvaa artikkelissaan vaatimusmäärittelyn huonoja käytänteitä. Hän painottaa vaatimusmäärittelyn tärkeyttä ja tarpeellisuutta. Vaatimusmäärittely on prosessi, jonka tuotoksena on yhteinen ymmärrys, oppiminen ja sitoutuminen projektiin. Vaatimusmäärittely koetaan pakollisena paheena, joka liian usein tehdään huolimattomasti, koska se vie liikaa aikaa. Tässä

kuitenkin lopputuloksen kannalta ollaan väärässä, koska monta asiaa jää huomiotta, jos vaatimuksia ei dokumentoida. (Vuori 2009.)

Yleinen sananlasku on, että asiakas on aina oikeassa. Sananlasku pitäisi pitää mielessä, kun kehitetään ohjelmistoja tai sovelluksia, koska silloin vaatimusmäärittelyn tekeminen on helpompaa ja lopputulos vastaa asiakkaan toiveita. Yli kahdenkymmenen vuoden asiakaspalvelu, keittiö- ja ravintolan työ ovat osoittaneet, että asiakasta voi ohjailla niin, että yhteysymmärrys saadaan aikaan helposti.

Virallista vaatimusmäärittelyä ei sovelluksesta vaadittu, mutta vaatimukset määriteltiin pitämällä palavereja sähköasentajien ja toimeksiantajan kanssa. Sähköasentajille esiteltiin sovelluksen prototyyppi ja palautteen perusteella sitä paranneltiin, jotta sovelluksesta tuli käyttäjäystävällisempi. Toisin sanottuna käyttäjiltä eli sähköasentajilta saatiin toiminnalliset vaatimukset. Toimeksiantaja puolestaan antoi reunaehtoja ja rajoituksia sovelluksen käyttöön eli pääosin ei-toiminnallisia vaatimuksia.

Alfamen vaatimusmäärittelyoppaan (2021) mukaan yksi vaatimusmäärittelyn onnistumiseen vaikuttavista asioista on se, että tutustuu ja sitoutuu ihmisiin, joille sovellusta tekee. Yhtä tärkeää on oppaan mukaan myös antaa tilaa uusille ideoille ja jatkaa iterointia projektin aikana. Sovelluksen tekemisen aikana hieman vieraaksi jääneet sähköasentajat tulivat tutuiksi. Toimeksiantajalla oli oma näkemyksensä siitä mitä sovelluksessa tuli olla ja sähkösuunnittelijat tulevat tarvitsemaan sovellusta hieman eri näkökulmasta kuin sähköasentajat. Näiden kaikkien tarpeista ja toiveista koostettiin käyttäjätarinoita (kts. Taulukko 2.), joiden pohjalta oli helppo tehdä toiminnallisia vaatimuksia.

Taulukko 2. Käyttäjätarinoita toiminnallisista vaatimuksista.

Käyttäjätarina	**Käyttäjä**
Haluan ”tsekkilistan”, josta voi tarkistaa, että kaikki tarvittavat valokuvat on tehty.	Sähköasentaja
Haluan tietää, jos jokin dokumentti tarvitsee allekirjoitustani.	Sähkösuunnittelija ja sähköasentaja
Haluan ottaa puhelimen kameralla kuvia.	Sähköasentaja
Haluan tallentaa kuvia tietylle projektille.	Sähköasentaja
Haluan tallentaa monta kuvaa kerralla.	Sähköasentaja
Haluan tallentaa noin 200 sivuisen sähkösuunnitelman.	Sähkösuunnittelija
Haluan täyttää kaavakkeen ja tallentaa sen.	Sähköasentaja
Haluan katsella, kuinka asiakkaan edellinen keskus on rakennettu	Sähkösuunnittelija ja Sähköasentaja
Haluan tulostaa kirjalliset dokumentit pdf-muodossa.	Toimeksiantaja, Sähkösuunnittelija ja Sähköasentaja

Alfamen vaatimusmäärittelyoppaan (2021) mukaan toiminnalliset vaatimukset kuvaavat mitä sovelluksen pitää pystyä tekemään.

Taulukossa 3 on kuvattu toiminnallisia vaatimuksia, jotka kerättiin käyttäjien vaatimuksista ja toiveista sekä toimeksiantajan havainnoista. Käyttäjätarinoista on pyritty listaamaan tärkeimmät vaatimukset sovellukselle.

Taulukko 3. Toiminnalliset vaatimukset

Vaatimus	Kuvaus
Tietojen tallentaminen	Sovelluksella pystytään tallentamaan valokuvia ja kirjallisia dokumentteja.
Tietojen selaus	Sovelluksella voidaan tarkastella asiakkaalle jo luovutettuja keskuksia.
Valokuvien ottaminen	Valokuvia voidaan ottaa puhelimen kameralla.
”Tsekkauslista”	Lista, josta voi varmistaa, että kaikki tarvittavat dokumentit ja valokuvat on otettu
Omien kuvien ja dokumenttien tarkastelu	Käyttäjällä on oma sivu, jossa voi muokata dokumentteja tai valokuvia.
Lähetteen tulostus	Sovelluksella voi tulostaa lähetteen.

Ei-toiminnalliset vaatimukset liittyvät ei-näkyviin ominaisuuksiin eli sovelluksen ja järjestelmän sisäisiin ja teknisiin ominaisuuksiin. Ei-toiminnallisilla vaatimuksilla on erittäin tärkeä rooli nimenomaan tietojärjestelmän luotettavuuden, laadun ja kustannustehokkuuden takaamisessa koko sovelluksen elinkaaren aikana. Ei-toiminnalliset vaatimukset eivät välttämättä näy käyttäjälle missään vaiheessa, mutta sitten kun ne näkyvät se on iso ongelma, ja siinä vaiheessa niitä on vaikea korjata. Ei-toiminnallisten vaatimusten kanssa on tärkeää tasapainotella, koska esimerkiksi parantamalla tietoturvaa saattaa huonontaa suorituskykyä. (Leppänen 2021.)

Opinnäytetyön ei-toiminnallisia vaatimuksia on kuvattu taulukossa 4. Nämä vaatimukset mukailivat varastokirjanpitosovelluksen vaatimuksia.

Taulukko 4. Ei-toiminnalliset vaatimukset

Vaatimus	Kuvaus
Saatavuus	Sovellus tulee olla käytettävissä 95% ajasta.
Skaalautuvuus	Sovelluksen käyttäjiä saattaa olla kerralla satoja. Sovellusta käytetään pääasiassa puhelimella, mutta myös tabletilla tai tietokoneella.
Siirrettävyys	Sovelluksen tulee toimia kaikilla alustoilla (Android, IPhone)
Tietoturva	Sovellus vaatii kirjautumisen. Käyttäjän tekemisistä jää merkintä loki-tiedostoon.
Käytettävyys	Sovellusta käytetään pääasiassa puhelimella, mutta myös tabletilla tai tietokoneella.
Ylläpidettävyys	Päivittäminen on helppoa.
Integroitavuus	Voidaan hakea tietoa toisesta tietokannasta.
Suorituskyky	Sovelluksen vasteaika on 8 sekunnin sisällä.

Sovellus toteutettiin Django Frameworkilla, joka käyttää Python-ohjelmointikieltä. Ulkoasu rakennettiin Tailwindillä. Tietokantana käytettiin PostgreSQL:ää. Opinnäytetyön web-sovellus on osa yrityksen Varastokirjanpito-sovellusta, jonka voi tarvittaessa poistaa käytöstä. Sovelluksen laajentuminen tulevaisuudessa oli oltava mahdollista.

3.3 Alkuvalmistelut

Opinnäytetyön web-sovellusta kehitettiin virtuaalikoneella, joka luotiin VMware Workstation-ohjelmalla. Virtuaalikoneelle asennettiin Debian 10-käyttöjärjestelmä, joka on avoimen lähdekoodin Linux-pohjainen järjestelmä. Visual Studio Code-ohjelma (myöh. VS Code) ohjelmointia varten. Django, jolla luotiin projekti, sovellukset ja tarvittavat Python-paketit, esimerkiksi virtuaalinen ympäristö. PostgreSQL-tietokannan tallennusta varten. Vain sovelluksen oma tietokanta luotiin tämän opinnäytetyön aikana, muut oli tehty ennen tämän opinnäytetyön aloittamista.

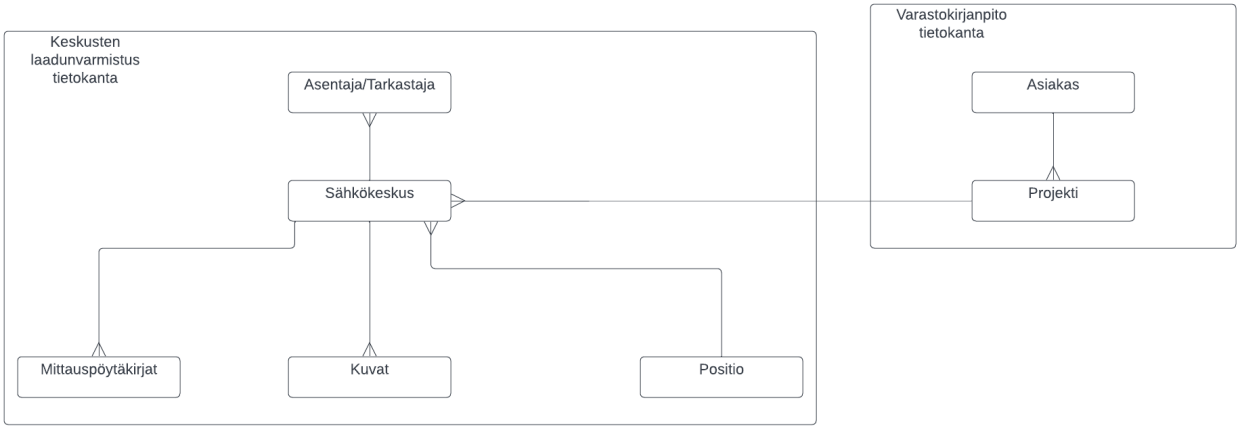
Opinnäytetyön web-sovellus liittyy kiinteänä osana aikaisemmin kehiteltyyn Varastokirjanpito-sovellukseen, mutta myöhempää mahdollista markkinointia varten haluttiin, että Keskusten laadunvarmistus-sovellus voitaisiin poistaa myytävästä paketista. Tämän teki mahdolliseksi se, että web-sovelluksesta luotiin projektiin oma sovellus.

3.4 Suunnittelu

3.4.1 Tietokanta

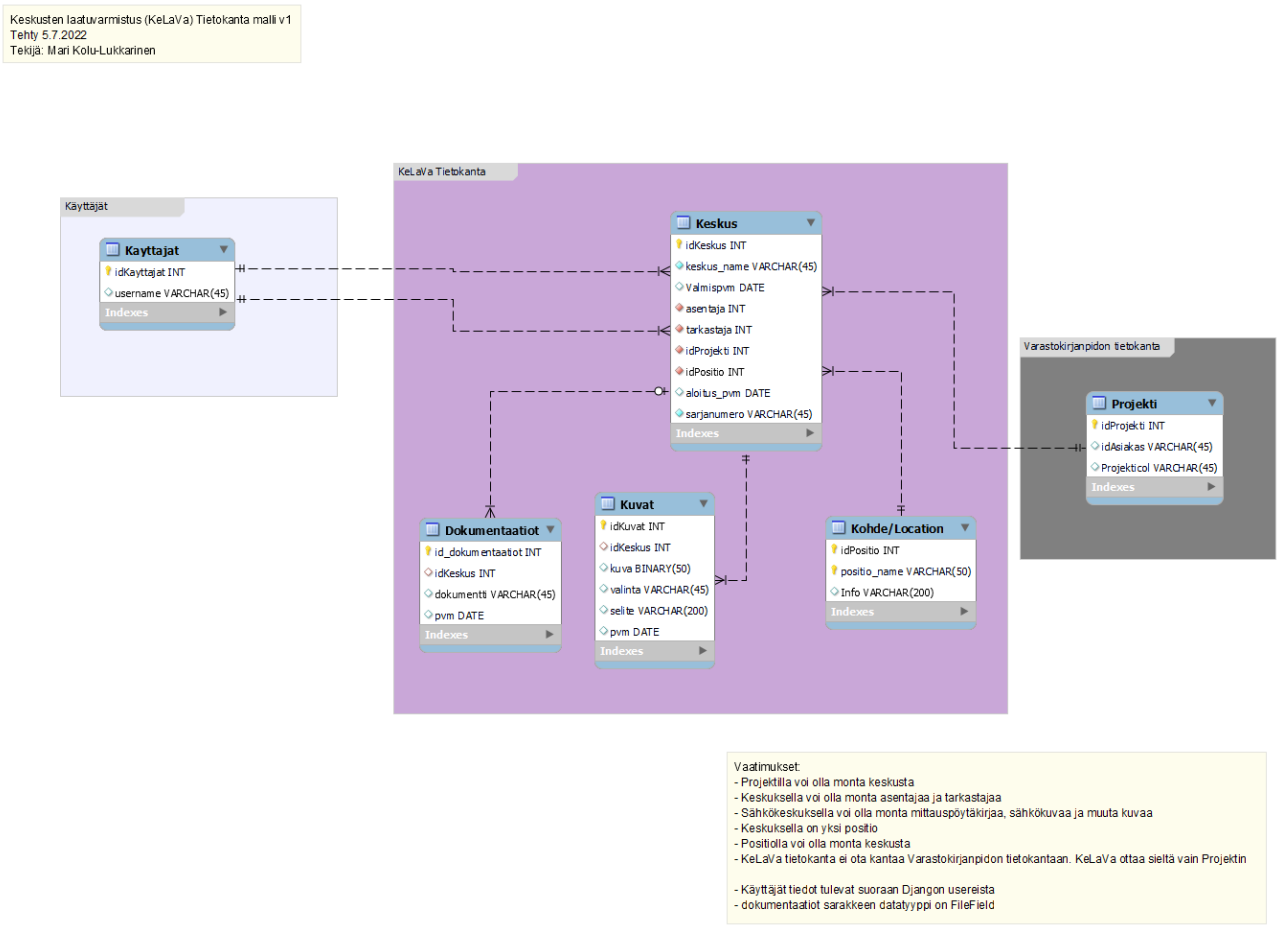
Tietokannan suunnittelu aloitettiin muodostamalla vaatimusten perusteella käsitemalli (kts. Kuvio 5.). Se on pelkistetty malli tietokannasta, joka muodostuu käsitteistä ja niiden välisistä riippuvuuksista. Käsitemallista saadaan selville seuraavat asiat:

- Sähkökeskuksella voi olla yksi tai useampi asentaja, tarkastaja, mittauspöytäkirja tai kuva.
- Projektilla voi olla yksi tai useampi sähkökeskus
- Positiolla eli tietyllä paikalla voi olla yksi tai useampi sähkökeskus
- Projekti on määritelty Varastokirjanpidon tietokannassa



Kuvio 5. Käsitemalli keskusten laadunvarmistustietokannasta ja yhteydestä varastokirjanpidon tietokantaan

Käsitemallin jälkeen suunniteltiin looginen malli (kts. Kuvio 6.), joka on edellistä tarkempi kuvaus tietokannasta ja siinä on kuvattu käsitteet ja niiden väliset yhteydet. Loogisessa mallissa tietokannat on jaoteltu kolmeen osaan, Djangon luomaan käyttäjä tietokantaan (Kayttajat), Varastokirjanpidon tietokantaan ja opinnäytetyössä rakennettuun Keskusten Laadunvarmistus (KeLaVa) tietokantaan. Loogisessa mallissa on otettu huomioon perusavaimet (Primary Key) ja viiteavaimet (Foreign Key).



Kuvio 6. Looginen malli keskusten laadunvarmistus tietokannasta

3.4.2 Prototyyppi

Vaatimusmäärittelyn perusteella tehtiin ensin paperille karkea luonnos sovelluksen ulkoasusta eli Wireframe. Tällaisen luonnoksen muuttaminen on halpaa ja nopeaa ja se vastaa kysymyksiin: Mitä, missä ja miten? (Nelimarkka 2019). Sähköasentajilta ja toimeksiantajalta tulleiden palautteiden takia luonnosta muokattiin pariin otteeseen ennekuin siirryttiin seuraavaan suunnitteluvaiheeseen.

Näköismalli eli MockUp on hyvin tarkasti ulkoasultaan suunniteltu malli. Sen tekeminen on hieman kalliimpaa kuin karkean luonnoksen, mutta sen avulla sovelluksen tai ohjelman myyminen asiakkaalle on helpompaa. (Mts. 2019.)

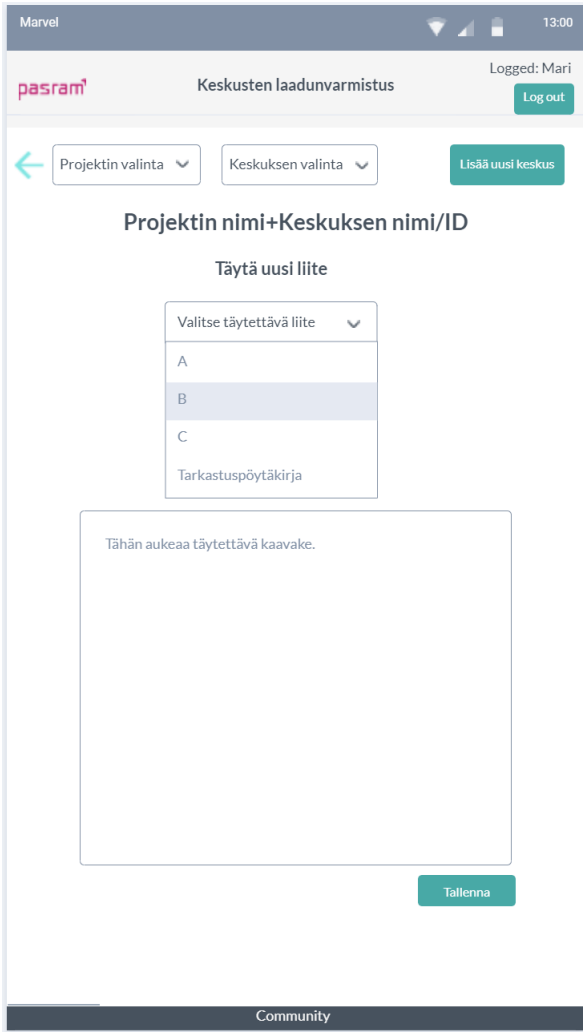
Prototyyppi on edellisten välimuoto. Prototyyppi on visuaalinen, interaktiivinen ja sillä voidaan testata kuinka käyttäjä käyttää sovellusta. Prototyypillä voidaan aikaisessa vaiheessa testata kuinka käyttäjä käyttää sovellusta ja tarpeen mukaan muuttaa suunnitelmia. Prototyypissä ulkoasu on melkein loppuun asti hiottu ja siinä on melkein kaikki toiminnallisuudet, esimerkiksi mitä tapahtuu, kun painaa painikkeesta. (Mts. 2019.)

Opinnäytetyön web-sovelluksen prototyypin tekemiseen käytettiin ilmaisohjelmaa nimeltään MarvelApp. Sillä pystyttiin luomaan yksi projekti, jonka alle tehtiin prototyyppi sovelluksesta sivu kerrallaan. Sivun luonnin jälkeen voitiin rakentaa painikkeiden väliset linkitykset ja mahdolliset muut interaktiiviset toiminnot. Tämän jälkeen MarvelAppilla voitiin luoda käyttäjätesti, jonka avulla näki miten käyttäjä esimerkiksi liikuttaa hiirtä näytöllä tai mistä painikkeista hän painelee. Prototyypin ulkoasu ei ole ihan tarkka, koska siihen ei ole esimerkiksi oikeata värimaailmaa. Prototyyppi tehtiin ajatellen käyttäjiä, jotka käyttävät sovellusta puhelimella. Myöhemmässä vaiheessa on tarkoitus rakentaa oikeanlainen näkymä myös isompia näyttöjä käyttäville.

Kuviossa 7 on näkyvillä prototyypin etusivu, johon päästään Varastokirjanpidon valikon kautta.

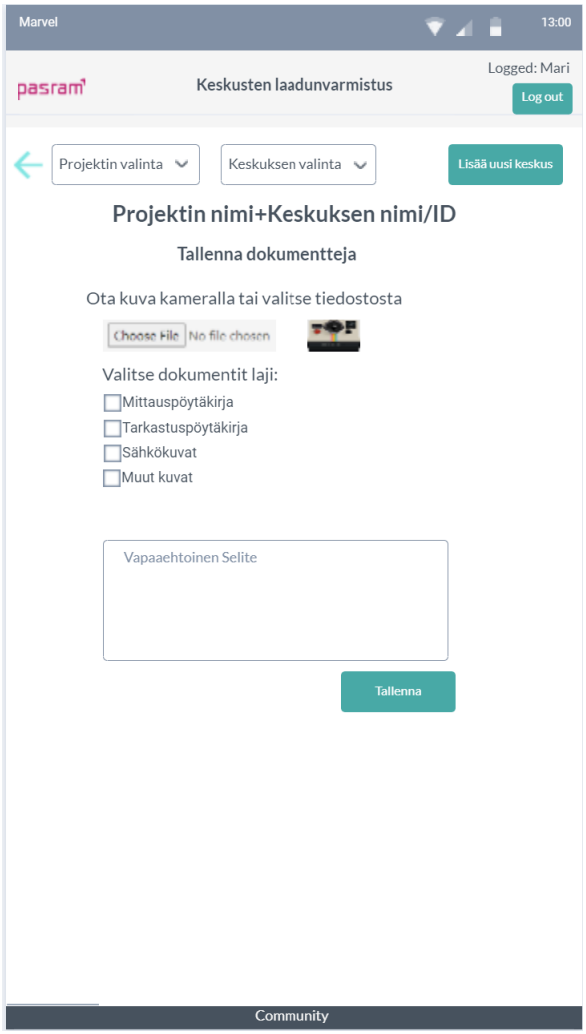
Kuvio 7. Sovelluksen prototyypin etusivunäkymä

Kuviossa 8 on näkyvillä web-sovelluksen Lisää liite -näkymä, johon päästään klikkaamalla etusivulla Täytä liite -painiketta.



Kuvio 8. Sovelluksen prototyypin liitesivunäkymä

Kuvia pääsee lisäämään klikkaamalla etusivulla lisää dokumenttipainiketta (kts. Kuvio 9). Sivulla on mahdollisuus lisätä kuva joko ottamalla käytettävän laitteen kameralla kuva tai valitsemalla tiedosto.



Kuvio 9. Sovelluksen prototyypin dokumentin tallennussivunäkymä

3.5 Toteutuksen ohjelmointi

3.5.1 Palvelinpuolen ohjelmointi

Tietokannan tekeminen vaati kirjautumisen PostgreSQL:ään ja yhden komennon SQL-kieltä. Kuvissa 10 on luotu keskuslaadunvalvonta niminen tietokanta ja tulostettu sen tiedot.

```
postgres=# CREATE DATABASE keskuslaadunvarmistus;
CREATE DATABASE
postgres=# \l+
postgres=# \l+
                                                                    List of databases
        Name          |  Owner   | Encoding |   Collate   |    Ctype    |   Access privileges   |  Size   | Tablespace |                Description
----------------------+----------+----------+-------------+-------------+-----------------------+---------+------------+--------------------------------------------
 keskuslaadunvarmistus | postgres | UTF8     | fi_FI.UTF-8 | fi_FI.UTF-8 |                       | 8553 kB | pg_default |
```

Kuvio 10. Keskuslaadunvalvonta-tietokannan luominen komentorivillä

Seuraavaksi luotiin olemassa olevaan Varastokirjanpito-projektiin uusi sovellus, nimeltään quality (kts. kuvio 11.). Komento luo tarvittavat tiedostot projektiin automaattisesti.

Kuvio 11. Sovelluksen luominen projektiin

Taulukossa 5 on kuvattu edellä luotujen tiedostojen sisältö. Itse lisättyjä tiedostoja olivat urls.py ja forms.py. Selainpuolen ohjelmointi on pääosin templates-kansiossa, tietokannan käyttöä varten on migrations-kansio ja models.py-tiedosto ja kaikki muut tiedostot ovat palvelinpuolen ohjelmointia varten.

Taulukko 5. Sovelluksen kansiot ja tiedostot

Tiedosto tai kansio	**Kuvaus**
migrations-kansio	Django tekee automaattisesti tänne tiedostoja, kun muutoksia tehdään models.py-tiedostoon ja migrate komento tehdään.
templates-kansio	Kansio sisältää kaikki HTML-tiedostot. Kansion sisällä tiedostot voidaan tarvittaessa jakaa mm. include- tai partial-kansioihin
__init__.py	Tiedosto kertoo Pythonille, että tiedosto on paketti ja kuuluu projektiin. Tarvittaessa täällä määritellään tarvittavat alimoduulit. Opinnäytetyön sovelluksessa tämä tiedosto on tyhjä.
admin.py	Tiedostossa voidaan muokata hallintapaneelissa näytettävää näkymää, joka näkyy vain pääkäyttäjälle, admin. Opinnäytetyön sovelluksessa ei ollut tarvetta muuttaa tätä näkymää.
apps.py	Tiedostossa voidaan määritellä käsitteitä. Opinnäytetyön sovelluksessa ei ollut tarvetta määrityksille.
forms.py	Tiedostossa luodaan kaavakkeisiin/lomakkeisiin tarvittavat tiedot, kuten näytettävä ja täytettävä teksti.
models.py	Tiedostossa rakennetaan tietokannan taulut ja niiden väliset yhteydet eli määritellään tietokannan rakenne. Tiedoston muutoksien käyttöönotto vaatii migrate-komennon suorittamisen.
tests.py	Tiedostoon voidaan luoda suoritettavia testejä. Opinnäytetyön sovelluksessa ei tehty testejä.
urls.py	Tiedostossa luodaan linkit URL:ien ja näkymien välille.
views.py	Tiedostoon tehdään funktioita ja/tai luokkia, jotka näkyvät käyttöliittymässä.

Tietokannan luomiseen tarvittavat luokat on tehty models.py-tiedostoon (kts. Kuvio 12.) Cabinet-luokka sisältää sähkökeskuksen sarakkeet, Document-luokassa dokumenttien ja Photo-luokassa kuvien sarakkeet. Jokaiseen luokkaan haluttiin selvyyden vuoksi id, joka luodaan automaattisesti ja on samalla perusavain (Primarykey), jota käytetään yhteyden luomiseksi toisiin luokkiin viiteavaimena (Foreignkey).

```python
class Cabinet(models.Model):

    id = models.AutoField(primary_key=True)
    name = models.CharField(max_length=45)
    date_of_manufacture = models.DateField(blank=True, null=True)
    serial_number = models.CharField(max_length=45,blank=True, null=True)
    mechanic = models.CharField(max_length=45)
    inspector = models.CharField(max_length=45,blank=True, null=True)
    location = models.CharField(max_length=50)
    project = models.CharField(max_length=45)
    date_to_start = models.DateField(blank=True, null=True, default=datetime.date.today)

    def __str__(self):
        return (f'{self.name}')


class Document(models.Model):
    id = models.AutoField(primary_key=True)
    name = models.CharField(max_length=45)
    document = models.FileField(upload_to='quality_temp/doc/%Y/%m/%d/', blank=True)
    date = models.DateField(blank=True, null=True, default=datetime.date.today)
    cabinet = models.ForeignKey(Cabinet, on_delete=PROTECT)

    def __str__(self):
        return (f'{self.name}')


class Photo(models.Model):

    category_list = (
        ("mittauspöytäkirja", "Mittauspöytäkirja"),
        ("sähkökuva", "Sähkökuva"),
        ("valokuva", "Valokuva"),
        ("liite", "Liite"),
    )

    id = models.AutoField(primary_key=True)
    name = models.CharField(max_length=45)
    photo = models.FileField(upload_to='quality_temp/photo/%Y/%m/%d/', blank=True)
    category = models.CharField(max_length=30, choices=category_list, default=category_list[0][0])
    description = models.CharField(max_length=200, blank=True, null=True)
    date = models.DateField(blank=True, null=True, default=datetime.date.today)

    cabinet = models.ForeignKey(Cabinet, on_delete=PROTECT)

    def __str__(self):
        return (f'{self.name}')

class StatusList(models.Model):

    id = models.AutoField(primary_key=True)
    minutes = models.BooleanField(default=False)
    electricpic = models.BooleanField(default=False)
    photopic = models.BooleanField(default=False)
    checked = models.BooleanField(default=False)

    cabinet = models.OneToOneField(Cabinet, on_delete=PROTECT)

    def __str__(self):
        return (f'{self.id}')

```

Kuvio 12. Keskusten laadunvarmistus tietokannan taulujen luominen

Tietokanta ei ole ihan samanlainen kuin oli suunniteltu, koska Djangolla ei tällä hetkellä pysty tekemään kahden tietokanannan välistä yhteyttä perus- ja viiteavaimilla. Varastokirjanpidon tietokannan tieto projektista ja käyttäjätiedot tuotiin selainpuolen avulla. Toinen muutos oli, että

Kohde/Lokaatio-taulua ei tehty erikseen, koska se on jokaisella sähkökeskuksella erilainen. Sovellukseen haluttiin lisätä palkki, josta näkisi työn etenemisen ja tämä toteutettiin lisäämällä Status-List-taulu tietokantaan.

Käyttäjälle näkyvät tiedot määriteltiin funktioilla ja luokilla views.py-tiedostossa. Etusivulla on kaksi pudotusvalikkoa, jotka ovat toisistaan riippuvaisia eli ensin valitaan projekti ja sitten sen perusteella voidaan valita haluttu sähkökeskus. Kuviossa 13 on näkyvillä funktio nimeltään quality_view, joka on pudotusvalikoiden, sähkökeskuksen yksityiskohtien ja tallennettujen tiedostojen tietojen välittäjä.

```
@login_required(login_url='/login')
def quality_view(request, pk=None):
    '''
    Frontpage view
    Dropdowns view
    '''
    context= {
        "projects" : Project.objects.all(),
        "cabinet_details" : Cabinet.objects.all().filter(id = pk)
            }
    if pk:
        context= {
            "projects" : Project.objects.all(),
            "cabinet_details" : Cabinet.objects.all().filter(id = pk),
            "photos" : Photo.objects.all().filter(cabinet = pk),
        }
    elif(request.method == 'GET'):
        if request.GET.keys():
            project = None
            if 'project':
                cabinet = None
                cabinet_details = None
                pk = None
                pk = request.GET.get('cabinet')
                project = request.GET.get('project')
                cabinet = Cabinet.objects.all().filter(project=project)
                cabinet_details =  Cabinet.objects.all().filter(id = pk)

            context= {
                'project' : project,
                "projects" : Project.objects.all(),
                "photos" : Photo.objects.all().filter(cabinet = pk),
                "cabinet_details": cabinet_details,
                "cabinets": cabinet
                }

    return render(request, 'quality/quality.html', context)
```

Kuvio 13. Käyttäjälle näkyvän tiedon määrittely funktio

Uuden sähkökeskuksen luominen tapahtuu kuviossa 14 näkyvällä funktiolla. Jos käyttäjä on täyttänyt oikein kaavakkeen niin tallennetaan tiedot tietokantaan ja palataan takaisin etusivulle ja näytetään viesti tallennuksen onnistumisesta. Virheen sattuessa näytetään virheviesti ja käyttäjä voi täyttää kaavakkeen uudelleen.

```python
@login_required(login_url='/login')
def add_new_cabinet(request):
    '''
    Add new Cabinet view
    '''

    if request.method == "POST":
        form = CabinetModelForm(request.POST)
        if form.is_valid():
            cd = form.cleaned_data
            pc = Cabinet(
                name = cd['name'],
                date_of_manufacture = cd['date_of_manufacture'],
                serial_number = cd['serial_number'],
                mechanic = cd['mechanic'],
                inspector = cd['inspector'],
                location = cd['location'],
                project = cd['project'],
                date_to_start = cd['date_to_start'],
            )
            pc.save()
            #print(pc.id)
            return redirect('success')
    else:
        form = CabinetModelForm()
    return render(request, 'quality/form_add_new.html', {'form': form})

```

Kuvio 14. Uuden sähkökeskuksen lisääminen

Sähkökeskuksen päivittämisen funktio on hyvin samankaltainen kuin edellä esitetty uuden luominen (kts. Kuvio 15.). Erona on se, että tietokannasta haetaan ensin instanssi, että saadaan kaikki olemassa olevat tiedot käyttäjälle näkyviin. Onnistuneen päivittämisen jälkeen käyttäjä ohjautuu etusivulle takaisin, jossa näkyy päivitetyt tiedot.

```python
@login_required(login_url='/login')
def update_cabinet(request, pk):
    cabinet = Cabinet.objects.get(id = pk)
    form = CabinetUpdate(instance = cabinet)
    if request.method == "POST":
        form = CabinetUpdate(request.POST, instance = cabinet)
        if form.is_valid():
            form.save()
            print(cabinet.id)
            return redirect('details', pk=cabinet.id)


    return render(request, 'quality/cabinet_update.html',  {'form': form})
```

Kuvio 15. Sähkökeskuksen tietojen päivittäminen

Tiedoston tallentamisessa käytetään samankaltaista funktiota kuin uuden sähkökeskuksen luomisessa (kts. Kuvio 16.). Erona on, että käyttäjä on jo valinnut sähkökeskuksen, johon tallennettava tiedosto liittyy. Onnistuneen tallentamisen jälkeen käyttäjä ohjautuu sovelluksen etusivulle.

```python
@login_required(login_url='/login')
def upload_file(request, pk):
    '''
    Upload files view
    '''
    obj = Cabinet.objects.get(id=pk)
    print(f"OBJ == {obj}")
    if request.method == 'POST':
        form = PhotoForm(request.POST, request.FILES)
        if form.is_valid():
            cd = form.cleaned_data
            pc = Photo(
                name = cd['name'],
                photo = cd['photo'],
                category = cd['category'],
                description = cd['description'],
                cabinet = obj,
                date = cd['date'],
            )
            pc.save()
            return redirect('details', pk=pk)
    else:
        form = PhotoForm()
    context = {
        'form' : form,
        'pk' :pk,
    }
    return render(request, 'quality/photo_form.html', context)
```

Kuvio 16. Tiedoston tallentaminen

Yllä mainitut näkymät ohjataan oikeaan URL-osoitteeseen urls.py-tiedostossa (kts. Kuvio 17). Tiedostossa määritellään ensin käyttäjälle näkyvä osoitepolku, näkymä ja annetaan osoitteelle nimi, jota voidaan käyttää selainpuolella, kun halutaan ohjata käyttäjä tietylle sivulle.

```python
urlpatterns = [

    # Project
    path('', views.quality_view, name='quality'),
    path('cabinet/<str:pk>/', views.quality_view, name='details'),
    path('fill_form/<str:pk>', views.fill_form, name='fill_form'),
    path('add_document/<str:pk>/', views.upload_file, name='add_document'),
    path('add_new/', views.add_new_cabinet, name='add_new_cabinet'),
    path('success/', views.success, name='success'),
    path('update_model/<str:pk>/', views.update_cabinet, name='update_model'),
    path('add_document/<str>/', views.upload_file, name='add_document'),
]
```

Kuvio 17. Projektin näkymien ohjaaminen oikeaan URL-osoitteeseen

Kaavakkeiden määrittely on tehty kahdella eri tavalla. Kuviossa 18 on näkyvillä uuden sähkökeskuksen ja dokumentin tallentaminen. Ne on tehty Form-luokkaa käyttäen, jolloin on mahdollista määrittää itse mitä tietoja halutaan käyttäjälle näyttää ja asettaa tiettyjä rajoituksia tai

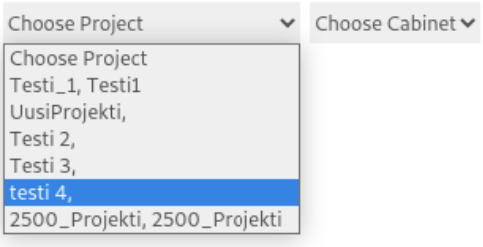
oletusarvoja. Alimmaisella luokalla päivitetään sähkökeskusta ja siinä on käytetty ModelForm-luokkaa, joka luo suoraan mallista kaavakkeen. Tässä kaavakkeessa haluttiin päästä muuttamaan vain sähkökeskuksen nimeä, kuvausta ja valmistuksen aloittamispäivämäärää.

```
class DateInput(forms.DateInput):
    input_type = 'date'

class CabinetModelForm(forms.Form):
    name = forms.CharField()
    description = forms.CharField()
    project = forms.ModelChoiceField(queryset=Project.objects.all())
    manufacture_started = forms.DateField(widget=DateInput())

class PhotoForm(forms.Form):
    name = forms.CharField()
    photo = forms.FileField()
    category = forms.ChoiceField(choices=Photo._meta.get_field('category').choices)
    description = forms.CharField()
    requires_attention = forms.BooleanField(required=False, label="The file needs a signature")

class CabinetUpdate(forms.ModelForm):
    class Meta:
        model = Cabinet
        fields = ('name', 'description', 'manufacture_started')
        widgets = {'manufacture_started': DateInput()}
```

Kuvio 18. Kaavakkeiden luominen

3.5.2 Selainpuolen ohjelmointi

Sovelluksen sivuja varten luotiin sivupohja, jota käytetään kaikissa muissa sivuissa. Sivupohjassa käyttäjä valitsee ensin projektin (kts. Kuvio 19) ja sitten tarkasteltavan sähkökeskuksen.



Kuvio 19. Projektin valinta pudotusvalikko

Kuviossa 20 on näkyvissä, kuinka pudotusvalikot, joiden perusteella projektin ja keskuksen valinnat tehtiin. Painikkeille on annettu id ja määritelty välitettävä tieto htmx-kirjastoa ja JavaScriptiä käyttäen.

```
<select name="select_project" id="select_project" hx-get="{% url 'quality' %}" hx-target="#main"
    hx-vars="project:get_selected_value('select_project')" hx-trigger="click">
    <option value="">Projektin valinta</option>
    {% for project in projects %}
    <option value="{{ project }}" id="select_project">{{ project }}</option>
    {% endfor %}
</select>

<select id="Select" hx-get="{%url 'quality' %}" hx-target="#main" hx-push-url="true"
    hx-vars="cabinet:get_selected_value('Select')" hx-trigger="click">
    <option value="">Keskuksen valinta</option>
    {% for cabinet in cabinets %}
    <option value="{{ cabinet.pk }}" id="cabinet_value">{{ cabinet }}</option>
    {% endfor %}
```

Kuvio 20. Projektin ja sähkökeskuksen valinta

Muuttujan siirtämistä varten luotiin JavaScript-funktio (kts. Kuvio 21.). Funktiota käytetään molemmissa pudotusvalikoissa siten, että haetaan muuttuja id:n perusteella, jolloin se saadaan pyyntöön (request) mukaan, jota käsitellään views.py-tiedostossa.

```
<script>
    function get_selected_value(selectorName) {
        console.log(document.getElementById(selectorName).value)
        return document.getElementById(selectorName).value;
    }

</script>
```

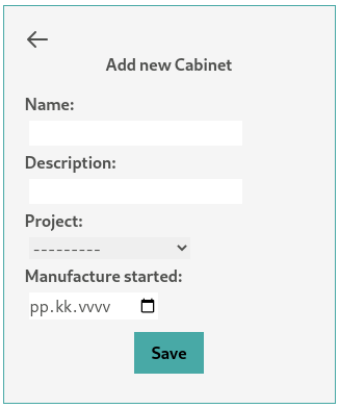
Kuvio 21. Muuttujan siirtäminen JavaScriptilla

Sovelluksessa kaavakkeiden HTML-tiedostot ovat samankaltaisia. Kuviossa 22 on näkyvillä uuden sähkökeskuksen lisääminen tietokantaan.

```
<h1 class="font-bold mb-3.5 text-center">Add new Cabinet</h1>
<form method="post" action="" hx-post="">
    {% csrf_token %}
    {{ form.non_field_errors }}
    <div class="pr-1">
        {% for field in form.visible_fields %}
        <div class="font-bold grid grid-cols-1 text-left">
            {{ field.errors }}
            <div>
                {{ field.label_tag }}
            </div>
            <div id="field">
                {{ field }}
            </div>
        </div>
        {% endfor %}
    </div>
    <div class="flex flex-col justify-center items-center">
        <input type="submit" class="btn m-2" value="Save">
    </div>
</form>
</div>
```

Kuvio 22. Uuden sähkökeskuksen lisääminen

Uuden sähkökeskuksen luominen on tehty käyttäjälle mahdollisimman helpoksi. Kuviossa 23 on näkyvillä täytettävä kaavake.



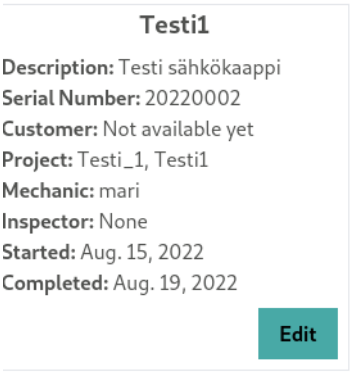
Kuvio 23. Kaavake, jolla tehdään uusi sähkökeskus

Sähkökeskuksen valinnan tai luomisen jälkeen etusivulle latautuu sen tiedot. Tiedot haluttiin eri järjestykseen kuin mitä ne olisi kaavakkeen kautta tulleet, joten niille tehtiin oma listausnäkymä (kts. Kuvio 24).

```
36      {% for cabinet in cabinet_details %}
37      <div class="border">
38
39          <div class="text-xl text-center font-semibold pb-2 m-4">
40              <h1>{{cabinet}} </h1>
41          </div>
42          <div id="info" class="text-left m-4">
43              <p><span class="font-semibold">Positio: </span>{{cabinet.location}}</p>
44              <p><span class="font-semibold">Sarjanumero: </span>{{cabinet.serial_number}}</p>
45              <p><span class="font-semibold">Asiakas: </span>tulossa oleva ominaisuus</p>
46              <p><span class="font-semibold">Projekti: </span>{{cabinet.project}}</p>
47              <p><span class="font-semibold">Asentajat: </span>{{cabinet.mechanic}}</p>
48              <p><span class="font-semibold">Tarkastaja: </span>{{cabinet.inspector}}</p>
49              <p><span class="font-semibold">Valmistuspvä: </span>{{cabinet.date_of_manufacture}}</p>
50
51
52              <div class="grid content-end justify-end">
53                  <button class="btn h-min m-2">
54                      <a href="{% url 'update_model' cabinet.pk %}">Muokkaa</a>
55                  </button>
56              </div>
57
58          </div>
59      </div>
60      {% endfor %}
```

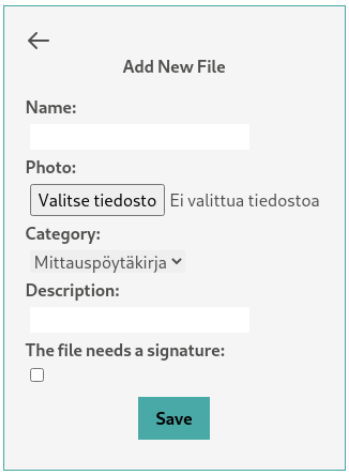
Kuvio 24. Sähkökeskuksen tiedot listattuna

Edellä kuvatut tiedot näkyvät käyttäjälle kuviossa 25 kuvatulla tavalla. Muokkaa-painiketta painamalla avautuu muokkaussivu, joka on samankaltainen kuin uuden sähkökeskuksen lisääminen esitäytetyillä tiedoilla.



Kuvio 25. Käyttäjänäkymä: Sähkökeskuksen tiedot

Uuden tiedoston lisääminen on HTML-tiedostoltaan samanlainen kuin uuden sähkökeskuksen luomisen. Käyttäjälle näkymä on erilainen johtuen täytettävistä kentistä (kts. Kuvio 26.). Alkuperäisestä suunnitelmasta poiketen käyttäjän laitteella otettava kuva ei onnistu vielä. Ohjelmoinnin edetessä haluttiin, että tallennettava tiedosto voi vaatia allekirjoituksen. Tämä toteutettiin lisäämällä kaavakkeeseen mahdollisuus laittaa ruksi, jolloin tämä tieto saadaan siirrettyä tietokantaan.



Kuvio 26. Käyttäjänäkymä: Uuden tiedoston lisääminen

Etusivulle haluttiin listaus lisätyistä dokumenteista (kts. Kuvio 27.). Taulukko on näkyvissä, jos sähkökeskukseen on lisätty tiedostoja. Tiedostoista haluttiin näkyviin nimi, kategoria ja tiedoston latauspäivä. Viimeisessä sarakeessa näkyy tallennetut kuvatiedostot pikkukuvakkeina, jota klikkaamalla se avautuu isompana taulukon yläpuolelle, PDF-tiedoston voi ladata omalle laitteelleen.

```
{% if photos %}
<div class="text-xl font-semibold pb-2">
    <h1>Saved Documents</h1>
</div>
<div id="modal01" class="" onclick="this.style.display='none'">
    <span class="right-0 cursor-pointer">&times;</span>
    <img id="img01" class="object-fill">
</div>
<table class="table-auto border-collapse border border-slate-400 whitespace-nowrap">
    <thead class="">
        <tr id="table_photo">
            <th id="table_photo">Name</th>
            <th id="table_photo">Category</th>
            <th id="table_photo">Date</th>
            <th id="table_photo">Photo</th>
        </tr>
    </thead>
    <tbody class="">
        {% for photo in photos %}
        <tr id="table_photo">
            <p>--- {{requires_attention}}</p>
            {% if requires_attention %}
            <td id="table_photo" class="bg-red">{{ requires_attention }}</td>
            {% else %}
            <td id="table_photo">{{ photo.name }}</td>
            {% endif %}
            <td id="table_photo">{{ photo.category }}</td>
            <td id="table_photo">{{ photo.date }}</td>
            <td id="table_photo">
                <div class="w-10">
                    {% if 'pdf' in photo.photo.url %}
                    <a href="{{photo.photo.url}}" type="application/pdf" download><img class=" h-full"
                            src="{% static 'images/download-pdf.png' %}" alt="Download pdf"></a>
                    {% elif 'png' in photo.photo.url %}
                    <img class="object-cover cursor-pointer" onclick="onClick(this)"
                        src="{{photo.photo.url}}" />
                    {% endif %}

                </div>
            </td>
        </tr>
        {% endfor %}
    </tbody>
</table>
</div>
</div>
{% endif %}
```

Kuvio 27. Tallennetujen tiedostojen taulukkonäkymä

Kuviossa 28 on näkyvissä, kuinka tallennetut tiedostot taulukko näkyvät käyttäjälle. PDF-tiedoston voi ladata omalle laitteelle punaista nuolta klikkaamalla.

Name	Category	Date	Photo
Testi1	Liite	Aug. 19, 2022	
Testi123	Mittauspöytäkirja	Aug. 19, 2022	
Liite A	Liite	Aug. 19, 2022	
Cabiinetti123456	Mittauspöytäkirja	Aug. 19, 2022	
Testi1	Mittauspöytäkirja	Aug. 19, 2022	

Kuvio 28. Käyttäjänäkymä: Tallennetut tiedostot taulukko

3.6 Jatkokehitys

Jatkokehitys ideoita tuli ilmi jo ensimmäisessä palaverissa, jossa opinnäytetyön aihetta käsiteltiin. Jo silloin tuli ilmi tarve siitä, että halutaan sovelluksen tallentavan tiedostot samaan paikkaan kuin muukin projektin dokumentaatio, jolloin ne ovat helposti löydettävissä. Tämän ominaisuuden

toteutuminen kuitenkin vaatii pilvipalvelun käytön, joka ei ollut opinnäytetyötä tehtäessä vielä ajankohtainen asia.

Lähetyslistan luominen on tarpeellinen siinä vaiheessa, kun sähkökeskus rakennetaan yrityksen toimitiloissa ja lähetetään asiakkaalle. Tämän ominaisuuden rakentaminen vaatii tutustumista laajemmin esimerkiksi lähetyslistan sisältöön ja muihin vaadittaviin toimintoihin.

Pidetyissä palavereissa kerrottiin kuinka kätevää olisi, jos voisi sähkökeskuksen perusteella hakea Varastokirjanpidosta siihen tarvittavat osat ja ottaa ne käyttöön. Tämän voisi toteuttaa muuttamalla Varastokirjanpidon tietokantaa niin, että tuotteet voisi kohdentaa tietylle sähkökeskukselle projektin sijaan.

Puhelimen kameran käyttö kuvien tallentamisessa helpottaisi ja nopeuttaisi sovelluksen käyttöä. Ominaisuus voidaan myöhemmin toteuttaa selainpuolella esimerkiksi JavaScriptiä käyttämällä.

Monen dokumentin tai kuvan tallentaminen kerralla nopeuttaisi sovelluksen käyttöä ja siihen tarvitaan välimuistin (session) käyttöä. Tämän ominaisuuden tekeminen opinnäytetyön aikana olisi vaatinut paljon aikaa, joten se jätettiin pois. Tämän ominaisuuden kehittäminen on tulevaisuudessa ensimmäisenä listalla.

4 Tulokset ja pohdinta

Opinnäytetyön tavoitteena oli suunnitella ja toteuttaa toimiva web-sovellus sähkökeskusten toiminnan ja laadun tarkkailuun. Henkilökohtaisena tavoitteena oli oppia lisää web-sovellusten kehittämisestä ja tarkastella kriittisesti löydettyjä materiaaleja ja muita tietoja.

Käytettävät teknologiat olivat valmiiksi valittuja ja ne olivat sopivat sovelluksen tekemiseen muutamia poikkeuksia lukuun ottamatta. Esimerkiksi kahden tietokannan välinen yhteys ei toiminut niin kuin suunnitteluvaiheessa oli kuviteltu, joten sen asian kanssa piti olla luova ja ratkaisu oli oikein onnistunut, mutta vaatii tarkastelua tulevaisuudessa, kun asiat kehittyvät.

Opinnäytetyön tekemisen aikana sai tutustua sähköasentajien ja -suunnittelijoiden työhön, joka oli mielenkiintoista. Heti alussa kävi selväksi, ettei liitteiden tekeminen sovelluksessa onnistuisi, koska ne vaativat allekirjoituksen. Sähköinen allekirjoitus olisi tähän tarkoitukseen hyvä, mutta opinnäytetyön tekemisen aika oli rajallinen ja asian tutkiminen olisi vaatinut paljon aikaa.

Opinnäytetyö rajattiin tutkimuskysymyksillä

1. Saadaanko käytettävillä teknologioilla aikaan laadun parantuminen?
2. Miten jatkokehittäminen tulee ottaa huomioon web-sovellusta tehtäessä?
3. Kuinka tärkeää on suunnitella web-sovellus ennen varsinaisen työn aloittamista?
4. Kuinka rajata toteutus opinnäytetyöhön sopivaksi?

Näihin kysymyksiin saatiin vastattua. Sovellusta tehdessä huomattiin, että Djangolla saa tehtyä paljon erilaisia asioita ja sen avulla on kätevä tehdä web-sovellusta. Tailwindin käyttö ulkoasun tyylittelyssä takaa sen, ettei samanlaista web-sovellusta tule vastaan. Tailwindin hallitseminen vaatii aikaa, mutta palkitsee lopussa. PostgreSQL-tietokantana palvelee myös suuria tietomääriä ja on helposti opittavissa. Laadun parantuminen näkyi siinä, että nyt kaikilla on pääsy samoihin kuviin ja pöytäkirjoihin ja ne ovat tallennettuina yhdenmukaisesti.

Jatkokehittäminen Djangolla on helppo ja nopeaa, koska sen kehittäminen on johdonmukaista ja jatkuvaa. Tulevaisuudessa sovelluksen muokkaaminen on helppoa, koska Django jakaa tiedostot järkevästi ja johdonmukaisesti. Tämän omaksuminen vaati ymmärrystä ja oman mielipiteen muuttamista pois totutusta.

Jo ennen opinnäytetyön aloittamista oli huomattu, että suunnittelu on yksi tärkeimmistä asioista ohjelmistokehityksessä. Opinnäytetyön tekeminen aloitettiin tarkalla suunnittelulla, mutta lopussa huomattiin, ettei kaikkea kuitenkaan osattu suunnitella loppuun saakka. Suunnittelu on suurimmalta osin miettimistä ja yksin se voi olla kapeakatseista, toisin sanottuna tiimityöskentelystä on suunnittelussa paljon hyötyä.

Opinnäytetyön rajaaminen oli haaste, koska toimeksiantajalla oli alusta alkaen jo paljon ideoita ja vaatimuksia. Suunnitteluvaiheessa piti tarkoin miettiä, että mitkä osa-alueet kannattaa toteuttaa opinnäytetyössä ja mitkä jätetään jatkokehitysvaiheeseen. Sovelluksen tekemisen aikana piti vetää raja mikä kuuluu opinnäytetyöhön ja mikä ei.

Web-sovelluksen toteuttaminen näin nopeassa ajassa oli haastavaa ja vaati paljon pitkäjänteisyyttä ja oma-aloitteisuutta. Usein ensimmäinen versio palvelin- tai selainpuolella tehtiin myöhemmin uudestaan, koska taitojen karttumisen myötä todettiin toisenlainen tapa paremmaksi ja suoraviivaisemmaksi. Yksin tällaisen web-sovelluksen tekeminen oli kuitenkin antoisaa ja mielenkiintoista, mutta myös vastuullista ja välillä haastavaakin.

Lähteet

BeginnersGuide/Overview. 2019. Python dokumentaatio. Viitattu 27.6.2022. https://wiki.python.org/moin/BeginnersGuide/Overview/.

Behrens, M. 2012. Django Book: Django's History. Viitattu 27.6.2022. https://django-book.readthedocs.io/en/latest/chapter01.html#django-s-history/.

Big, N. 2022. Django tutorial for beginners. Viitattu 27.6.2022. https://mastering-django.com/django-tutorial-for-beginners/.

Chu, J. 2021. Monetizing Open-Source: How Tailwind CSS has grown into a $2m+ business. Viitattu 13.7.2022. https://www.smalltechbusiness.com/monetizing-open-source-tailwind/.

DeBarros, A. 2018. Practical SQL: A Beginner's Guide to Storytelling with Data. Paikka: No Starch Press. 1. p. Viitattu 10.7.2022. https//janet.finna.fi, Skillsoft Books ITPro.

Get started with Tailwind CSS. 2022. Tailwind CSS dokumentaatio. Viitattu 11.7.2022. https://tailwindcss.com/docs/installation.

Katungi, D. 2021. Introduction to Tailwind CSS. Section Engineering Education artikkeli. Viitattu 13.7.2022. https://www.section.io/engineering-education/introduction-to-tailwind-css/.

Korsun, J. 2016. 10 popular websites built with Django. Viitattu 27.6.2022. https://djangostars.com/blog/10-popular-sites-made-on-django/.

Leppänen, T. 2021. Cheetah Consulting Blogi. Viitattu 8.7.2022. https://www.cheetah.fi/blog/mihin-tarvitaan-ei-toiminnallisia-it-vaatimuksia/.

Nelimarkka, P.2019. TTOS0100: Prototyypit. TTOS0100-kurssin Youtube-video. Viitattu 13.7.2022. https://www.youtube.com/watch?v=sLj2EJgD4gs

PostgreSQL 14.4 Documentation. N. d. PostgreSQL 14.4 dokumentaation verkkosivut. Viitattu 14.7.2022. https://www.postgresql.org/docs/current/history.html/.

Tutkimus- ja kehittämistoiminta. N. d. Tilastokeskuksen määritelmä 1. Viitattu 9.7.2022. https://www.stat.fi/meta/kas/t_ktoiminta.html.

Vaatimusmäärittelyopas. 2021. Alfamen vaatimusmäärittelyopas. Viitattu 14.7.2022. https://www.alfame.com/hubfs/files/Vaatimusma%CC%88a%CC%88rittely%20kettera%CC%88ssa%CC%88%20ohjelmistokehityksessa%CC%88%20-opas.pdf.

Vuori, M. 2009. Vaatimusmäärittelyn huonoimmat käytännöt. Systeemityö 2/2009. Viitattu 8.7.2022. https://www.mattivuori.net/julkaisuluettelo/liitteet/vaatimusmaarittelyn_huon_kayt_st.pdf/.

Wathan, A. 2018. Going Full-Time on Tailwind CSS. Blogi. Viitattu 10.7.2022. https://adamwathan.me/going-full-time-on-tailwind-css/.

Wathan, A. 2022. Standalone CLI: Use Tailwind CSS without Node.js. Blogi. Viitattu 27.6.2022 https://tailwindcss.com/blog/standalone-cli.