Vuokranhallintaohjelmiston kehittäminen

Joona Laitila

Opinnäytetyö, AMK
Helmikuu 2023
Tieto- ja viestintätekniikka

jamk | Jyväskylän ammattikorkeakoulu
University of Applied Sciences

jamk

Kuvailulehti

Laitila, Joona

Vuokranhallintaohjelmiston kehittäminen

Jyväskylä: Jyväskylän ammattikorkeakoulu. Huhtikuu 2023, 40 sivua.

Tekniikan ala. Tieto- ja viestintätekniikan tutkinto-ohjelma. Opinnäytetyö AMK.

Julkaisun kieli: suomi

Julkaisulupa avoimessa verkossa: kyllä

Tiivistelmä

Opinnäytetyön tavoitteena oli kehittää toimeksiantajalle ohjelmisto, helpottamaan vuokranantajan työtaakkaa. Toimeksiantaja toimi vuokranantajana ja hallitsi useita vuokrakohteita, vuokrakohteiden määrän takia vuokra-asuntojen, sekä vuokrasopimuksien hallinta ja ylläpito oli haastavaa, sekä työlästä.

Ohjelmiston ideana oli automatisoida erilaisia vuokranhallintaan liittyviä asioita, kuten esimerkiksi vuokrankorotuksien suorittamista, vuokrasopimuksien luomista, vuokrattavien asuntojen, sekä vuokralaisten hallintaa.

Ohjelmistolla pystyisi hyödyntämään julkisia rajapintoja vuokrankorotuksia tehdessä, vuokrankorotuksiin käytettävän indeksin pystyy valitsemaan listasta ja valinnan tehtyä ohjelmisto palauttaa tämänhetkisen indeksin pisteluvun. Ohjelmistolla pystyy myös luomaan uusia vuokrakohteita, lisäämään niihin ominaisuuksia, tarkastelemaan vuokralaisten tietoja, luomaan vuokrasopimuksia, sekä lataamaan valmiin ja täytetyn vuokrasopimuksen tulostettavassa muodossa. Ohjelmiston toiminnot toimisivat rinnakkain ja jo vuokrasopimuksen luomisvaiheessa, sopimuksen tyhjät kentät täyttyvät automatisoidusti tiettyjen ehtojen täyttyessä. Ohjelmistoon sisältyy myös käyttäjien, sekä käyttöoikeuksien hallinta.

Ohjelmisto rakennettiin pääasiassa käyttämällä Angular ohjelmistokehystä frontendinä, Python Flask mikroverkkokehystä ohjelmointirajapintana, sekä MySQL tietokantana. Ohjelmistossa käytettiin myös useita erilaisia kirjastoja ja moduuleja, kuten JWT, Moment.js, SQLAlchemy, Jinja2, Weasyprint, I18n.

Työn tuloksena oli toimiva ohjelmisto, kaikkine toimintoineen. Työ soveltuu myös hyvin jatkokehitykseen.

Avainsanat (asiasanat)

Python Flask, Angular, MySQL, SQLAlchemy, JWT

Muut tiedot (salassa pidettävät liitteet)

jamk

Description

Laitila, Joona

Development of rental management software

Jyväskylä: JAMK University of Applied Sciences, April 2023, 40 pages.

Information and Communications Technology. Bachelor´s thesis.

Permission for open access publication: yes

Language of publication: Finnish

Abstract

The goal of the thesis was to develop software for the client, to ease the landlord's workload. The client acted as a landlord and managed several rental properties, due to the number of rental properties, the management and maintenance of rental apartments and lease agreements was challenging and laborious.

The idea of the software was to automate various things related to rent management, such as the execution of rent increases, the creation of lease agreements, the management of rented apartments and tenants.

With the software, it would be possible to use public interfaces when making rent increases, the index used for rent increases can be selected from a list, and after the selection is made, the software returns the current index score. With the software, you can also create new rental properties, add properties to them, view tenants' information, create rental agreements, and download a ready and completed rental agreement in printable form. The functions of the software would work in parallel and already at the stage of creating the lease agreement, the empty fields of the agreement will be filled in automatically if certain conditions are met. The software also includes management of users and access rights.

The software was mainly built using the Angular framework as frontend, the Python Flask micronetwork framework as API and MySQL as database. The software also used several different libraries and modules, such as JWT, Moment.js, SQLAlchemy, Jinja2, Weasyprint, I18n.

The result of the work was a functional software with all the required properties. The software is also well suited for further development.

Keywords/tags (subjects)

Python Flask, Angular, MySQL, SQLAlchemy, JWT

Miscellaneous (Confidential information)

Sisältö

1 Johdanto 4
1.1 Taustaa 4
1.2 Ongelma 4
1.3 Tavoite 4
2 Käytetyt teknologiat ja työkalut 5
2.1 Angular 5
2.1.1 Yleistä 5
2.1.2 Komponentit 6
2.1.3 Moduulit 7
2.1.4 Palvelut 8
2.2 Python Flask 9
2.3 SQLAlchemy 11
2.4 MySQL 12
2.4.1 Mikä on MySQL 12
2.4.2 Rakenne 14
2.5 JWT 14
2.5.1 Mikä on JWT 14
2.5.2 Käyttö 15
2.5.3 Rakenne 15
3 Toteutus 16
3.1 Projektin aloitus 16
3.2 Aikataulu 17
3.3 Angular selainpuoli 17
3.3.1 Rakenne 17
3.3.2 Palvelut 20
3.3.3 Pääkomponentit 25
3.3.4 Putket 30
3.4 Flask palvelinpuoli 31
3.4.1 Rakenne 31
3.4.2 Mallit 32
3.4.3 Templaatit 34
3.4.4 Reitit 36

4 Tulokset ja pohdinta 37

Lähteet 39

Liitteet 41

Kuviot

Kuvio 1. Suosituimmat verkkokehykset ja teknologiat (Stackoverflow developer survey 2022). 5
Kuvio 2. Yksinkertainen Angular komponentti (What is Angular 2022) 6
Kuvio 3. Yksinkertainen NgModulen määritys (Introduction to modules 2022) 8
Kuvio 4. Esimerkki palvelusta joka logittaa selaimen konsoliin (Introduction to services 2022) . 8
Kuvio 5. Esimerkki riippuvuus injektiosta (Dependency injection in Angular 2020) 9
Kuvio 6. Suosituimmat verkkokehykset ja kirjastot (Python development survey 2021) 10
Kuvio 7. Suosituimmat olio-relaatio kartoittajat (Python development survey 2021) 11
Kuvio 8. Suosituimmat tietokannat 2022 (The most popular databases 2022) 13
Kuvio 9. Yritykset jotka käyttävät MySQL:ää teknologioinaan (Who uses MySQL 2020)..... 14
Kuvio 10. Esimerkki JWT-tokenin otsikosta (Introduction to JSON Web Tokens N.d) 15
Kuvio 11. Esimerkki JWT-tokenin sisällöstä (Introduction to JSON Web Tokens N.d) 16
Kuvio 12. Esimerkki JWT-tokenin allekirjoituksesta (Introduction to JSON Web Tokens N.d)... 16
Kuvio 13. Palvelut kansio sisältöineen 18
Kuvio 14. Komponentti joka sisältää kaksi lapsi komponenttiä..... 18
Kuvio 15. Uudelleen käytettävä vahvistus dialogi 19
Kuvio 16. Pipes kansio joka pitää sisällään sovelluksessa käytety putket 19
Kuvio 17. AppModule määritykset 20
Kuvio 18. Esimerkki riippuvuus injektiosta (Understanding dependency injection 2022) 21
Kuvio 19. Authentikoinnin sieppaaja 21
Kuvio 20. JWT-token tallennettuna selaimen paikalliseen varastoon..... 22
Kuvio 21. JWT-tokenin dekoodaus..... 22
Kuvio 22. Funktio joka suorittaa käyttäjä tarkistuksen suojatuille reiteille..... 23
Kuvio 23. setSession metodi authentikointi palvelussa..... 23
Kuvio 24. Sovellus kutsuu addContract palvelua 24
Kuvio 25. addContract palvelu 24
Kuvio 26. Lomakeryhmän määritys..... 25
Kuvio 27. Create-contracts ngOnInit() metodi..... 26
Kuvio 28. Käyttäjänäkymä sopimusta luodessa 27
Kuvio 29. Metodi joka noutaa valitun asunnon tiedot 27

Kuvio 30. Vuokrankorotus indeksien käsittely....28
Kuvio 31. Rekisteröinti komponentin käyttäjänäkymä....29
Kuvio 32. Käyttäjän rekisteröimiseen tarkoitettu metodi....29
Kuvio 33. Yksinkertainen hakufiltteri putki....30
Kuvio 34. Palvelinpuolen kansiorakenne....31
Kuvio 35. Mallit kansio sisältöineen....31
Kuvio 36. Reitit kansion sisältö....32
Kuvio 37. Sovelluksen talo luokka.....34
Kuvio 38. Esimerkki Jinja2 templaatissa näytetystä datasta.....35
Kuvio 39. Funktio joka palauttaa tiedoston PDF-muodossa....35
Kuvio 40. Funktio joka palauttaa sopimukset JSON muodossa.....36
Kuvio 41. Esimerkki moduulista.....37

1 Johdanto

1.1 Taustaa

Vuokranantajana toimiminen voi olla monenlaista, helppoa, vaivalloista, stressaavaa tai vaikkapa täysin vaivatonta. Vuokranantajan työtehtäviin kuuluu kuitenkin monenlaista, mihin ulkopuolinen ei ole ikinä törmännytkään. Vuokrattavien kohteiden määrän kasvaessa, vuokranantajalla voi olla vaikeksia pysyä mukana kohteiden ja sopimusten tilasta, kuten esimerkiksi päättyvät sopimukset, huoneistoihin tulevat huollot, huoneiston asukkaat, asukkaiden yhteystiedot ja niin edespäin. Ohjelmisto antaa helpon tavan käsitellä näitä kaikkia yhtenä kokonaisuutena ja näin ollen helpottaa vuokranantajan työtaakkaa.

1.2 Ongelma

Opinnäytetyön tarkoituksena oli rakentaa toimeksiantajan kuvaama ohjelmisto. Ohjelmiston käyttötarkoituksena on pitää vuokranantaja ajan tasalla kaikista vuokranhallintaan liittyvistä asioista. Toimeksiantaja toimii vuokranantajana usealle eri kohteelle, usealla eri paikkakunnalla. Projektin idea syntyi, kun toimeksiantaja teki vuosittain suoritettavia vuokrantarkastuksia ja totesi sen manuaalisesti ja käsintehtävänä erittäin työlääksi.

1.3 Tavoite

Valmiin opinnäytetyön tarkoitus oli onnistuneesti helpottaa vuokranantajan toimenkuvaan kuuluvaa työtaakkaa. Aiemmin manuaalisesti hoidetut työt toteutetaan ohjelmistolla automaattisesti ja näin ollen vähennetään vuokranantajan työmäärää sekä nopeutetaan koko vuokranhallinta prosessia vuokranantajan näkökulmasta. Ohjelmistolla on helpotettu uusien sopimusten luomista ja vuokrankorotuksien tekemistä. Ohjelma tarjoaa useita erilaisia työkaluja vuokranantajan päivittäisen työn nopeuttamiseen, sekä auttaa pysymään ajantasalla erilaisista tärkeistä vuokrakohteisiin liittyvistä päivämääristä

Tässä opinnäytetyössä käydään läpi ohjelmiston kehitysprosessia, käytettyjä teknologioita, saavutettuja tavoitteita sekä pohdintaa.

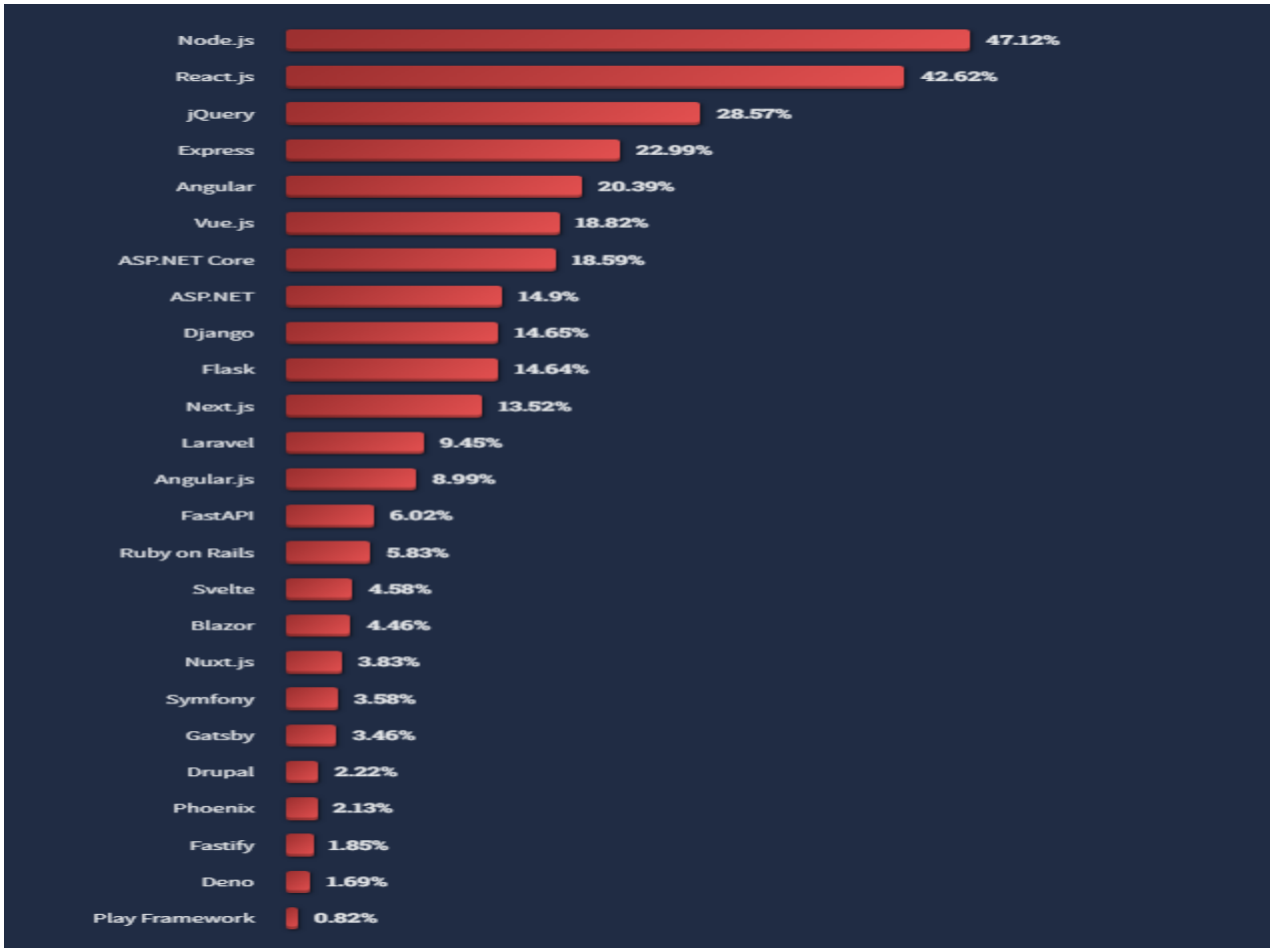
2 Käytetyt teknologiat ja työkalut

2.1 Angular

2.1.1 Yleistä

Angular on suosittu avoimen lähdekoodin kehys dynaamisten web-sovellusten rakentamiseen. Angularin on kehittänyt Google ja se perustuu TypeScript kieleen. Angular helpottaa laajojen ja monimutkaisten web-sovellusten luomista. Angularin sisältää komponentteja, palveluita, moduuleja, sekä malleja, näitä rakennuspalikoita hyödyntämällä käyttäjille avautuu helppo tapa luoda dynaamisia web-sovelluksia.

Stackoverflow:n vuonna 2022 suorittaman tutkimuksen perusteella Angular oli silloin maailman viidenneksi suosituin verkkokehys (ks kuvio 1).



Kuvio 1. Suosituimmat verkkokehykset ja teknologiat (Stackoverflow developer survey 2022)

Angular käyttää komponenttipohjaista arkkitehtuuria, jossa komponentit ovat pää rakennuspalikoita Angular-sovelluksessa ja jokainen komponetti on järjestetty NgModuleiksi. NgModule on Angularin moduulikonsepti, joka mahdollistaa sovelluksen loogisen järjestämisen. Moduulit ovat kokoelma komponentteja palveluita ja putkia, jotka ovat toiminnallisuutensa puolesta toisiinsa liittyineitä. Moduuleja voidaan siten tuoda myös toiseen moduuliin, joka mahdollistaa sovelluksen jakamisen pienempiin ja ylläpidettävämpiin osiin. Moduulit edesauttavat myös skaalautumisessa, kun samoja komponenttejä voidaan jakaa useiden eri moduulien kesken.

TypeScript kieli on laajennettu versio JavaScriptistä, se parantaa JavaScript kielen ominaisuuksia staattisella tyypityksellä, luokilla, rajapinnoilla, sekä muilla ominaisuuksilla. Angularissa TypeScriptiä käytetään sen vahvan tyypityksen ja monipuolisten ominaisuuksien takia, mikä auttaa käyttäjiä tekemään tehokkaampia, ylläpidettävämpiä ja turvallisempia sovelluksia. (Introduction to Angular concepts, 2022.)

2.1.2 Komponentit

Angular komponentti on keskeinen osa Angular sovellusta, joka hallinnoi osaa sovelluksen käyttöliittymästä. Se on Typescript luokka joka kapsuloi logiikan, datan, sekä toiminnot komponentin sisällä. Komponentti on vastuussa vain tietyn käyttöliittymän osan toiminnoista, kuten esimerkiksi, sivuston, lomakkeen tai alasvetovalikon (ks kuvio 2).

```
import { Component } from '@angular/core';

@Component({
  selector: 'hello-world',
  template: `
    <h2>Hello World</h2>
    <p>This is my first component!</p>
  `
})
export class HelloWorldComponent {
  // The code in this class drives the component's behavior.
}
```

Kuvio 2. Yksinkertainen Angular komponentti (What is Angular 2022)

Jokaisella Angular komponentilla on oma malli, joka on HTML-pohja, se määrittelee komponentin näkymän, rakenteen ja asettelun. Mallissa voidaan käyttää data sidoksia, kuten two-way binding, se mahdollistaa datan siirtelyn HTML-pohjan, sekä Typescriptin välillä. Tätä data sidosta hyödyntämällä pystytään helposti näyttämään tietoja loppukäyttäjälle ja sen avulla loppukäyttäjä voi tehdä muutoksia sovelluksen Typescript luokassa oleviin tietoihin käyttöliittymän avulla.

Angular komponenttiin kuuluu myös luokka, joka määrittelee komponentin ominaisuudet ja metodit. Luokassa voi määritellä komponentin käyttöliittymässä näkyvät syötteet ja tulosteet, näiden avulla komponentti voi vastaanottaa tietoa loppukäyttäjiltä ja jakaa sitä eteenpäin. Komponentit voivat myös lähettää dataa toisille komponenteille tai palveluille.

Angular komponentit ovat hyvin modulaarisia ja ne helpottavat sovelluksen ylläpitoa. Angular-komponentteja hyödyntämällä sovellus pystytään jakamaan pienempiin ja uudelleenkäytettäviin osiin, joka edesauttaa laajojen ja helposti skaalautuvien sovellusten rakentamista. (Introduction to Angular, 2022)

2.1.3 Moduulit

Angular moduuli on osa Angular-sovellusta, se määrittelee eri toiminnallisuudet sovelluksen eri osiin. Moduulit auttavat organisoimaan sovelluksen eri osat loogisiksi kokonaisuuksiksi. Moduulit tarjoavat tapoja helpottamaan laajemmankin sovelluksen ylläpitoa.

Angular moduuli on käytännössä TypeScript-luokka, joka on merkitty @NgModule-annotaatiolla. Moduuli voi sisältää esimerkiksi komponentteja, palveluita, putkia tai muita moduuleja. Moduuli tarjoaa myös tapoja käyttöön ottaa sovelluksessa erilaisia kolmannen osapuolen kirjastoja tai muita ulkoisia resursseja (ks kuvio 3).

src/app/app.module.ts

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
@NgModule({
  imports:      [ BrowserModule ],
  providers:    [ Logger ],
  declarations: [ AppComponent ],
  exports:      [ AppComponent ],
  bootstrap:    [ AppComponent ]
})
export class AppModule { }
```

Kuvio 3. Yksinkertainen NgModulen määritys (Introduction to modules 2022)

Moduuli määrittelee sovelluksen riippuvuudet ja käyttää tarjoajia (providers), esimerkiksi, se tarjoilee instansseja palveluista, joita sovelluksen eri osat käyttävät. Moduuleja voi myös käyttää laiskasti lataamaan (lazy load) sovelluksen eri osia, mikä voi parantaa suorituskykyä vähentämällä latausaikaa. (Introduction to Angular, 2022)

2.1.4 Palvelut

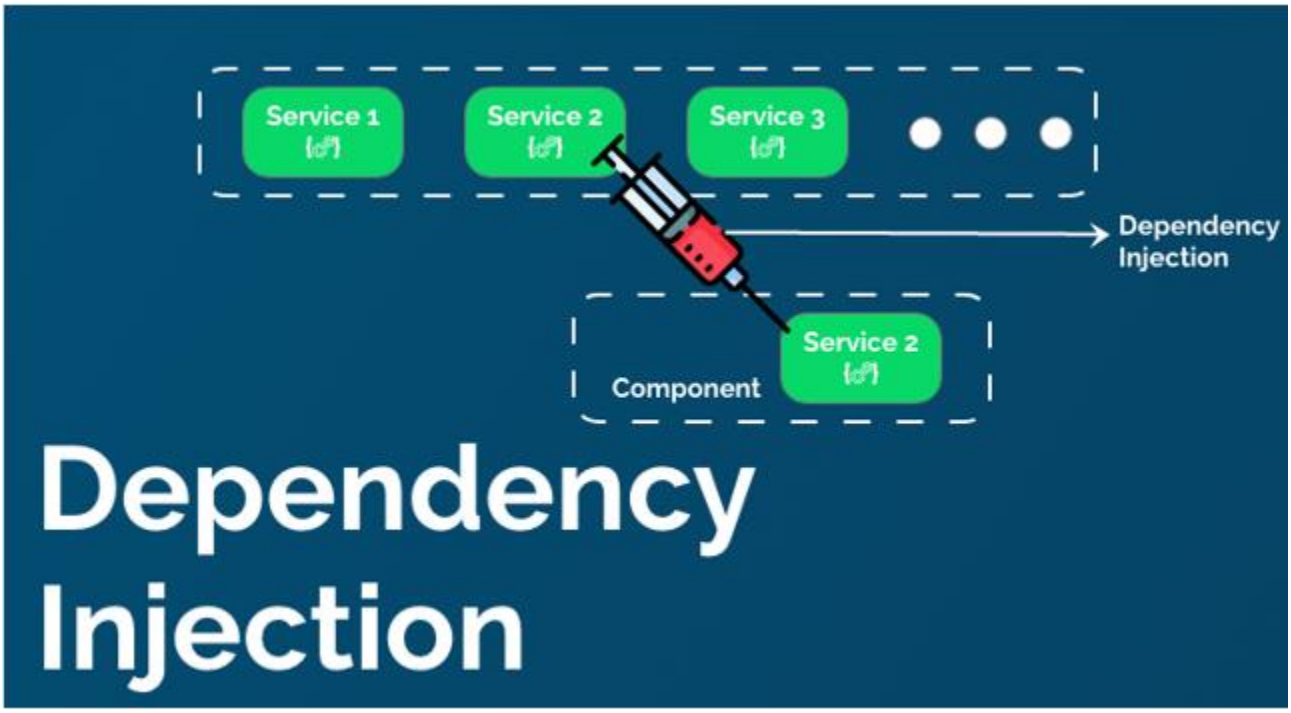
Angular palvelut ovat Typescript luokkia, jotka käsittelät data ja sisältävät eri toiminnallisuuksia. Palveluita voidaan helposti jakaa eri osiin Angular sovelluksessa. Palveluita käytetään tyypillisesti hallinnoimaan datan käyttöä tai kommunikoimaan ulkoisten palveluiden kanssa, kuten päätelaitteen ja palvelimen välinen keskustelu (ks kuvio 4).

src/app/logger.service.ts (class)

```
export class Logger {
  log(msg: any)   { console.log(msg); }
  error(msg: any) { console.error(msg); }
  warn(msg: any)  { console.warn(msg); }
}
```

Kuvio 4. Esimerkki palvelusta joka logittaa selaimen konsoliin (Introduction to services 2022)

Angular palveluita voidaan helposti käyttää useissa eri komponenteissä, sekä toisissa palveluissa. Angular palvelut luodaan injektoitavina luokkina, tällä tavalla ne voidaan injektoida toisiin komponentteihin käyttämällä riippuvuusinjektio toimintoa (ks kuvio 5). Se helpottaa palveluiden käyttämistä toisissa komponenteissa tai palveluissa, lisäksi sen avulla kehittäjät voivat varmistaa, että komponenti ja palvelut pysyvät toisistaan erillään. Tämä kaikki mahdollistaa sovelluksen helpomman ylläpidon, vianmäärityksen ja testauksen.



Kuvio 5. Esimerkki riippuvuus injektiosta (Dependency injection in Angular 2020)

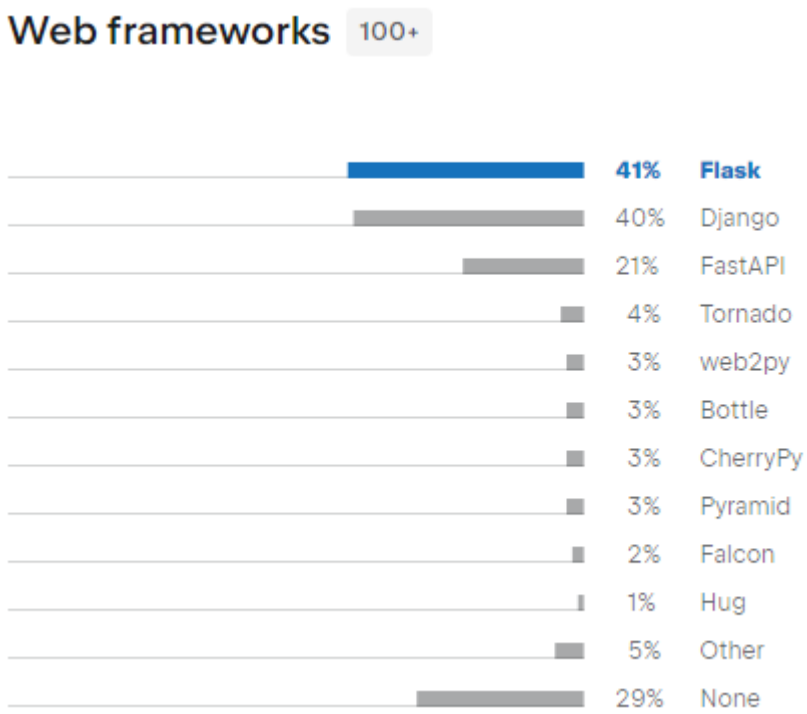
Angular palvelut ovat tärkeä osa helposti ylläpidettävien ja skaalautuvien Angular sovellusten rakentamista. Ne Tarjoavat helpon tavan jakaa toiminnallisuuksia sovelluksen eri osiin. (Introduction to Angular, 2022)

2.2 Python Flask

Flask on Pythonin verkkokehys, jonka avulla voit rakentaa web-sovelluksia. Verkkokehys on kokoelma erilaisia kirjastoja, sekä moduuleja. Nämä yhdessä mahdollistavat web-kehittäjille helpon tavan rakentaa sovelluksia. Flaskin vahvuudet ovat siinä, että se on helppokäyttöinen, nopea ja joustava. Flask tarjoaa vain välttämättömimmät työkalut web-sovellusten rakentamiseen, kuten

pyyntöjen reitittämisen, HTTP pyyntöjen ja vastausten käsittelyn ja mallien renderöinnin. Flask on kuitenkin hyvin pitkälle laajennettavissa ja erilaisia kolmannen osapuolen kirjastoja on saatavilla monenlaisia toimintoja varten. Nämä kaikki tekevät Flaskista hyvin suositun vaihtoehdon kaiken kokoisten web-sovellusten kehittämiseen.

Vuonna 2021 Jetbrainsin suorittaman tutkimuksen perusteella 41% Python web-kehittäjistä käytti Flask mikrokehystä pääasiallisena web-kehitys tekniikoinaan. (ks kuvio 6) (Python development survey 2021)



Kuvio 6. Suosituimmat verkkokehykset ja kirjastot (Python development survey 2021)

Yksi Flaskin tärkeimpänä ominaisuutena on sen yksinkertaisuus. Se on helppo ottaa käyttöön, sekä sen käyttö voi olla helppoa jopa uudellekkin web-kehittäjälle. Flask web-sovelluksia voidaan rakentaa vaivattomasti ja nopeasti, sillä kehyksen minimaalisuus mahdollistaa sen, että kehittäjät voivat keskittyä vain tärkeimpiin toiminnalisuuksiin web-sovelluksissaan.

Flask on rakennettu Werkzeug WSGI (Web Server Gateway Interface) työkalupakin ja Jinja2-template enginen päälle. Werkzeug tarjoaa matalan tason käyttöliittymän HTTP-pyyntöjen ja vastausten

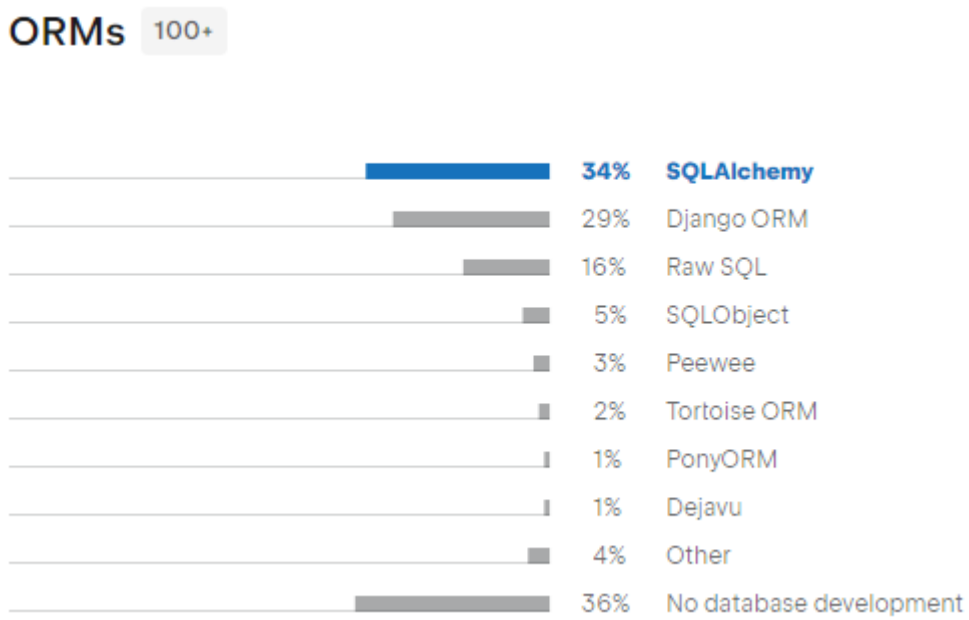
käsittelyyn, kun taas Jinja2 mahdollistaa dynaamisten HTML-mallien luomisen. (Jinja2 Explained in 5 Minutes 2018)

Flask ei vaadi mitään tiettyä hakemistorakennetta, joten tiedostoja ja kansioita voidaan järjestellä tarpeen mukaan. Lisäksi Flask tukee erilaisia HTTP-pyyntömenetelmiä, kuten GET, POST, PUT ja DELETE. (What is Flask Python, 2021)

2.3 SQLAlchemy

SQLAlchemy on avoimen lähdekoodin Python kirjasto, jota käytetään tietokantojen suunnittelussa ja rakentamisessa. Se tarjoaa vaihtoehtoisen muodon tietokantakyselyille, hyödyntäen Object-Relational Mapping (ORM) tekniikkaa luodakseen sillan olio-ohjelmoinnin sekä relaatio tietokannan välille. ORM tekniikka mahdollistaa tietokantakyselyiden tekemisen käyttämällä olioita SQL-kyselyiden sijasta.

Vuonna 2021 Jetbrainsin suorittaman tutkimuksen perusteella 34% Python web-kehittäjistä käytti. SQLAlchemyä pääasiallisena olio-relaatio kartoittajanaan (ks kuvio 7). (Python development survey 2021)



Kuvio 7. Suosituimmat olio-relaatio kartoittajat (Python development survey 2021)

Kaikkiaan SQLAlchemy mahdollistaa olioiden vertailun tietokanta tauluihin ja näiden olioiden instanssien vertailua taulujen riveihin. Tämä tapahtuu käyttäen deklaratiivistä syntaksia, mikä tarkoittaa, että kehittäjät määrittelevät tietokantataulujen rakenteen Python luokkina, jolloin SQLAlchemy generoi SQL-kyselyt tietokannan muokkaamista varten.

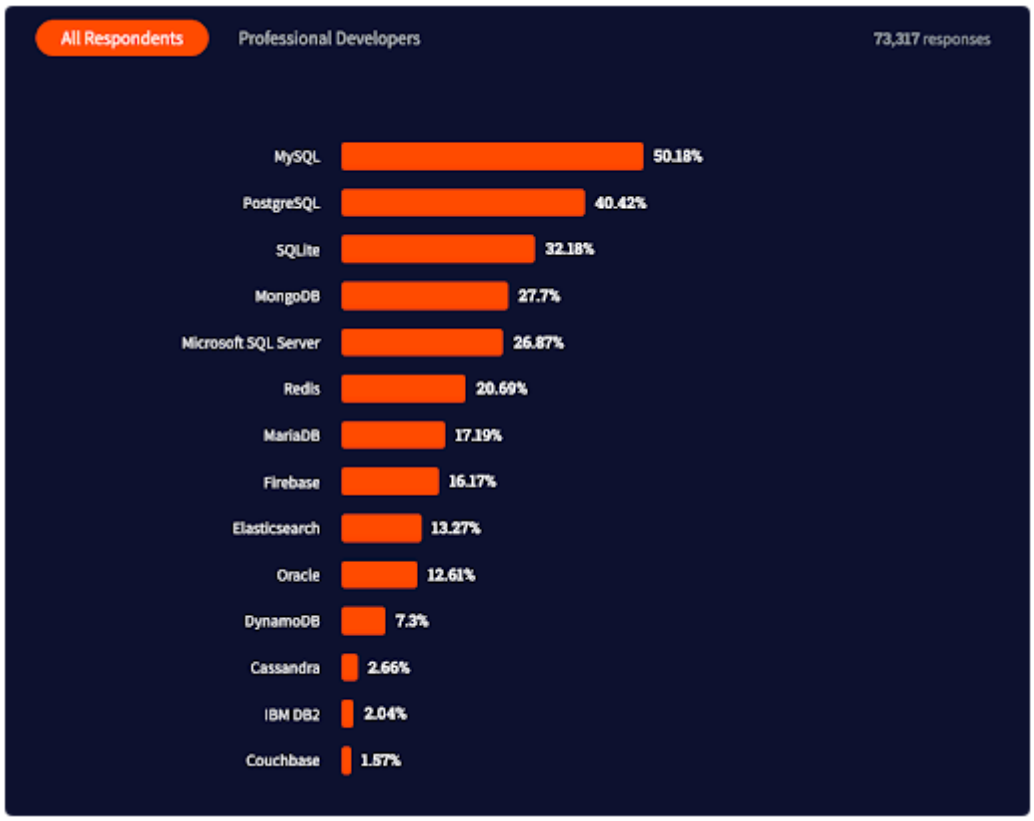
SQLAlchemy tukee suurta valikoimaa erilaisia SQL-tietokantoja, kuten SQLite, MySQL ja PostgreSQL, siksi yksi SQLAlchemyn vahvuuksista onkin sen joustavuus. Käyttäjät voivat myös valita käyttävätkö he sitä vain generoimaan SQL-kyselyitä vai täysimittaisena ORM:ina.

SQLAlchemy sisältää myös laajan valikoiman eri työkaluja ja laajennuspaketteja tietokantojen kanssa työskentelyyn. Esimerkiksi Flask-SQLAlchemy laajennus integroi SQLAlchemyn Flask web-kehykseen, mikä helpottaa tietokantojen kanssa kommunikoivan Flask-sovelluksen rakentamista. (SQLAlchemy 2.0 Documentation, 2023)

2.4 MySQL

2.4.1 Mikä on MySQL

MySQL on avoimen lähdekoodin relaatiotietokannan hallintajärjestelmä (RDBMS), se on yksi maailman eniten käytetyistä tietokantajärjestelmistä (ks kuvio 8). MySQL merkittävimpiä etuja on sen korkea suorituskyky, sen järjestelmä on varustettu tehokkaalla palvelinryhmällä, tämä mahdollistaa suurten datamäärien käsittelyn helposti ja nopeasti.



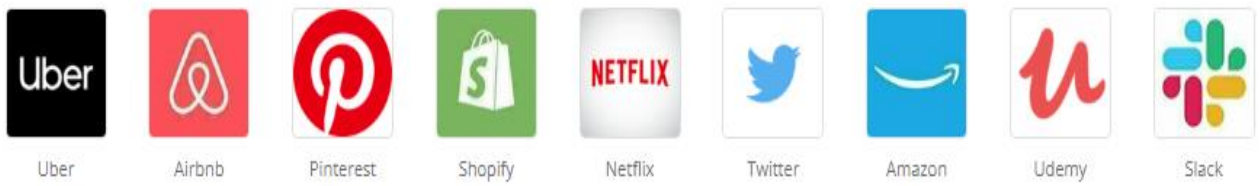
Kuvio 8. Suosituimmat tietokannat 2022 (The most popular databases 2022)

MySQL järjestelmän merkittävimpiä etuja on sen korkea suorituskyky ja turvallisuus. MySQL järjestelmä on varustettu tehokkaalla palvelin ryhmällä (server cluster), mikä mahdollistaa suurten datamäärien käsittelyn nopeasti ja tehokkaasti. (What is MySQL N.d.)

MySQL tarjoaa myös useita turvallisuus ominaisuuksia:

- Käyttöoikeudet: MySQL tarjoaa kattavan käyttöoikeuksien hallintajärjestelmän.
- Käyttäjätilien hallinta: MySQL mahdollistaa käyttäjätilien luomisen ja hallinnoinnin, mukaan lukien salasanasääntöjen asettamisen, salasanojen vanhentamisen ja käyttäjätilien lukitsemisen epäonnistuneiden kirjautumisyritysten jälkeen.
- Salaus: MySQL tukee siirtyvien ja levossa olevien tietojen salausta. MySQl voi käyttää SSL/TLS salausta palvelimen ja asiakkaiden välillä siirrettävien tietojan salaamiseen. Lisäksi MySQL tukee myös muita salauslaajennuksia kuten TDE (Transparent Data Encryption), TDE:tä voi käyttää salaamaan levossa olevia tietoja.
- Auditointi: MySQL:ssä on sisäänrakennetut auditointiominaisuudet, jotka mahdollistavat käyttäjätoiminnan ja tietokantaan tehtyjen muutoksien lokitietojen tallentamisen. Tämä ominaisuus auttaa paljastamaan mahdolliset tietoturva loukkaukset.
- Varmuuskopiointi ja palautus: MySQL tarjoaa useita varmuuskopiointi- ja palautusvaihtoehtoja, mukaan lukien täydelliset varmuuskopiot, inkrementaaliset varmuuskopiot ja ajastetut varmuuskopioinnit sekä palautukset.

Yhteenvetona voidaan todeta, että MySQL:n turvallisuusominaisuudet tekevät siitä luotettavan ja turvallisen RDBMS-vaihtoehdon kaikenkokoisille yrityksille. (Major Advantages of Using MySQL 2023)



Kuvio 9. Yritykset jotka käyttävät MySQL:ää teknologioinaan (Who uses MySQL 2020)

2.4.2 Rakenne

MySQL tietokannan rakenne koostuu yhdestä tai useammasta taulusta. Jokainen taulu koostuu sarakkeista, joita voidaan kutsua myös kentiksi. Sarakkeet edustavat erilaisia ominaisuuksia tai tallennettuja tietoja taulussa, jokaisella sarakkeella on tietty tietotyyppi, se voi olla esimerkiksi numeerinen, merkkijono tai aika.

Taulut voivat olla myös suhteessa toisiinsa avainten kautta. Ensisijaiset avaimet ovat yksilöllisiä tunnisteita, jotka annetaan kullekkin tietueelle, kun se lisätään tauluun. Vierasavaimia käytetään linkittämään tietueita yhdestä taulusta toiseen ja ne perustuvat yhteiseen avaintunnisteeseen.

2.5 JWT

2.5.1 Mikä on JWT

JWT (JSON Web Token) on avoin standardi (RCF 7519), se määrittelee kompaktin tavan välittää tietoa JSON-oliona. JWT:tä käytetään yleensä web-sovelluksissa käyttäjän tunnistamiseen. JWT-tokenin etuihin kuuluu sen helppokäyttöisyys, käyttäjä voidaan varmentaa ilman turhia tietokantakyselyitä, tämä tekee siitä kevyen ja tehokkaan tavan käyttäjän tunnistamiseen. JWT on laajalti

käytetty standardi turvallisessa tiedonsiirrossa kahden osapuolen välillä. (Introduction to JSON Web Tokens N.d)

2.5.2 Käyttö

Kun käyttäjä kirjautuu web-sovellukseen, sovellus voi generoida JWT-tokenin, joka sisältää käyttäjään liittyviä tietoja, kuten käyttäjätunnuksen tai roolin. JWT-token voidan lähettää takaisin käyttäjälle ja tallentaa paikallisesti esimerkiksi eväisteisiin tai paikalliseen tallennustilaan (local storage). Seuraavaksi JWT-token voidaan sisällyttää käyttäjän tekemiin pyyntöihin, joka mahdollistaa palvelimelle käyttäjän tunnistautumisen. (Introduction to JSON Web Tokens N.d)

2.5.3 Rakenne

JWT-token koostuu kolmesta osasta: otsikko (header), sisältö (payload) ja allekirjoitus (signature). Otsikko sisältää tokenin metadata, kuten tyypin ja algoritmin, hash-algoritmiä käytetään yleensä JWT-tokenin salaukseen. Salaus algoritminä voi toimia esimerkiksi: HMAC, SHA256 tai RSA (ks kuvio 10). JWT tokenin sisältö sisältää varsinaisen datan, kuten käyttäjän tunnukset. Sisältöön kuul-uvat ne tiedot mitä siirrellään kohteiden välillä. (ks kuvio 11). Allekirjoitus taas varmentaa, että token on aito eikä sitä ole muokattu välityksen aikana (ks kuvio 12). (Introduction to JSON Web Tokens N.d)

Kuvio 10. Esimerkki JWT-tokenin otsikosta (Introduction to JSON Web Tokens N.d)

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

Kuvio 11. Esimerkki JWT-tokenin sisällöstä (Introduction to JSON Web Tokens N.d)

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret)
```

Kuvio 12. Esimerkki JWT-tokenin allekirjoituksesta (Introduction to JSON Web Tokens N.d)

3 Toteutus

3.1 Projektin aloitus

Työskentelyn alkaessa projektista ei ollut valmiina mitään muuta kuin idea, kävimme toimeksiantajan kanssa läpi projektin projektin toteutusta ja aikataulua. Toimeksiantaja ehdotti palvelin puolelle teknologioiksi Python Flaskia, mutta antoi vapaat kädet käyttöliittymä puolella teknologian valinnassa. Kirjoittaja päätyi valitsemaan Angularin käyttöliittymä puolelle, valinta kohdistui Angulariin, koska toimeksiantajalla oli laaja kokemus Angularista ja tarvittaessa apua oli saatavilla helposti, jos siihen oli tarvetta.

Projektin suunnittelu alkoi tietokantarakenteen mallinnuksella, tietokannan suunnittelussa käytin avuksi Draw.io sovellusta, jolla rakensin pienimuotoisen UML diagrammin. Tämä diagrammi sisälti raakaversion tietokannan rakenteesta, joka toimi perustana sen rakentamiselle. Vaikka toimeksiantaja ei vaatinut mitään tarkempaa vaatimusmäärittelyä tai projektisuunnitelmaa, kävimme kuitenkin yhdessä läpi projektin etenemistä, sekä seuraavia vaiheita, tällä tavoin projekti eteni sulavasti ja aikataulussa.

Projektin edetessä kommunikaatio toimeksiantajan kanssa oli tiivistä ja avointa. Työskentelimme lähes vierekkäin, joten toimeksiantajan oli helppo seurata projektin etenemistä, keskustelimme useasti mahdollisista haasteista ja ratkaisuista. Toimeksiantajan kanssa yhteistyö oli mutkatonta sillä hän antoi kirjoittajalle vapauden toteuttaa projektia itsenäisesti, mutta oli kuitenkin tarvittaessa valmiina opastamaan ja neuvomaan. Toimeksiantajan kokemus Angularista osoittautui arvokkaaksi resurssiksi projektin toteutuksessa. Vaikka laajempaa vaatimusmäärittelyä tai projektisuunnitelmaa ei ollut, tiivis kommunikaatio ja yhteistyö toimeksiantajan kanssa mahdollistivat projektin sujuvan etenemisen ja lopulta onnistuneen lopputuloksen.

3.2 Aikataulu

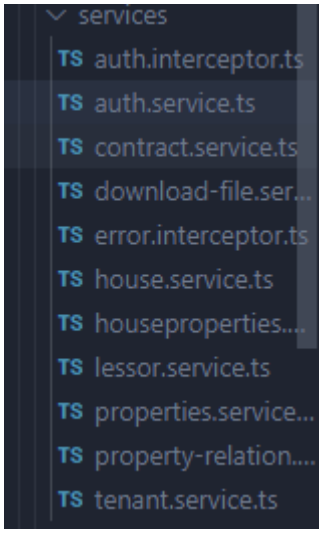
Projekti aloitettiin varsinaisesti marraskuun ensimmäisellä viikolla 2022. Projektin toteutukseen ei varattu mitään tiettyä aikamäärää, vaan sen parissa sai työskennellä koko työharjoittelujakson ajan ja tarvittaessa pitempäänkin, joten projektin toteutukseen oli aikaa vähintäänkin viisi kuukautta. Projektin edetessä saatoimme vaihtaa käytettyjä teknologioita meille paremmin soveltuviin, sekä uusia toimintoja keksittiin vauhdista lisää. Aikaa projektin parissa kului enimmäkseen projektin toteutukseen tai erilaisten toteutustapojen testailuun.

3.3 Angular selainpuoli

3.3.1 Rakenne

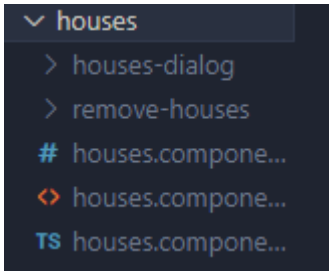
Projektihakemisto koostuu pääpiirteittäin neljästä osiosta, palvelut, komponentit, putket ja moduulit.

Palvelut sisältävät sovelluksessa käytetyt HTTP-pyyntöjen ja vastausten hallinnan. Projektissa palveluita on käytössä lähes joka komponentissa. Projektin alkuvaiheessa rakensin HTTP-pyynnöt suoraan komponenttien sisälle, mutta projektin edetessä ja sen kasvaessa kävi ilmi, että palveluita käyttämällä projektin rakenteesta saa huomattavasti selkeämmän ja ylläpidettävämmän, kun samoja toimintoa ei tarvitse kirjoittaa useampaan eri paikkaan vaan se onnistuu kätevästi palveluita hyödyntämällä.



Kuvio 13. Palvelut kansio sisältöineen

Komponentit taas sisältävät, ohjelmistossa käytettävät lomakkeet, sekä käyttäjänäkymät. Usea pääkomponentti sisältää myös lapsi komponentin (ks kuvio 14), mitä kutsutaan myös dialogiksi, dialogi sisältää yleensä erilaisia valintaikkunoita komponentin sisällä tai erilaisia uudelleen käytettäviä vahvistus ikkunoita (ks kuvio 15).

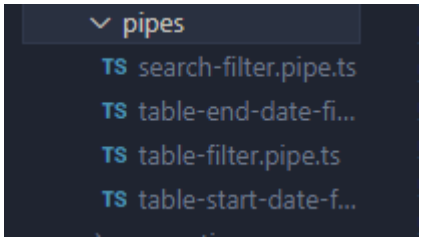


Kuvio 14. Komponentti joka sisältää kaksi lapsi komponenttiä

```
export class ConfirmationDialogComponent implements OnInit {

  message: string = "Are you sure you want to delete?";
  // confirmButtonText = "Yes";
  // cancelButtonText = "Cancel";

  errorMessage: any;

  constructor(@Inject(MAT_DIALOG_DATA) public data: any, private dialogRef: MatDialogRef<ConfirmationDialogComponent>) {
    if (data) {
      this.message = data.message || this.message;
      if (data.buttonText) {
        this.data.confirmButtonText = data.buttonText.ok || this.data.confirmButtonText;
        this.data.cancelButtonText = data.buttonText.cancel || this.data.cancelButtonText;
      }
    }
  }

  onConfirmClick(): void {
      this.dialogRef.close(true);
  }
```

Kuvio 15. Uudelleen käytettävä vahvistus dialogi

Pipes eli putket kansio sisältää erilaisia filtteröinti työkaluja (ks kuvio 16), sovelluksessa on käytössä hakukenttiä joilla voi filtteröidä hakutuloksia tai muita käyttäjälle näkyviä tietoja, käytössä on myös ngx kirjastosta löytyvä translate putki, jota käytetään kääntämään käyttäjälle näkyviä tietoa toiselle kielelle.



Kuvio 16. Pipes kansio joka pitää sisällään sovelluksessa käytety putket

Tärkein sovelluksessa käytetyistä moduuleista on AppModule. AppMdule on sovelluksen juurimoduuli, joka sisältää sovelluksen komponentit, palvelut sekä muut moduulit. Moduulit koostuvat useista komponenteistä, palveluista ja direktiiveistä, ne on ryhmitelty yhteen tiettyä toimintoa tai tarkoitusta varten (ks kuvio 17).

```
@NgModule({
  declarations: [
    AppComponent,
    LoginComponent,
    ContractsComponent,
    HousesComponent,
    ContractsDialogComponent,
    HousesDialogComponent,
    TenantsComponent,
    TenantsDialogComponent,
    ConfirmationDialogComponent,
    PropertiesComponent,
    SearchFilterPipe,
    RemovePropertiesDialogComponent,
    EditPropertiesDialogComponent,
    CreateContractsComponent,
    RelationDialogComponent,
    AllowanceDialogComponent,
    TermsAndConditionsDialogComponent,
    RegisterComponent,
    RemoveHousesComponent,
    UnauthorizedComponent,
    LessorsComponent,
    EditLessorDialogComponent,
    ExpirationDialogComponent,
    RemoveLessorDialogComponent,
    TableFilterPipe,
    TableStartDateFilterPipe,
    TableEndDateFilterPipe,
    EditContractDialogComponent,
  ],
```

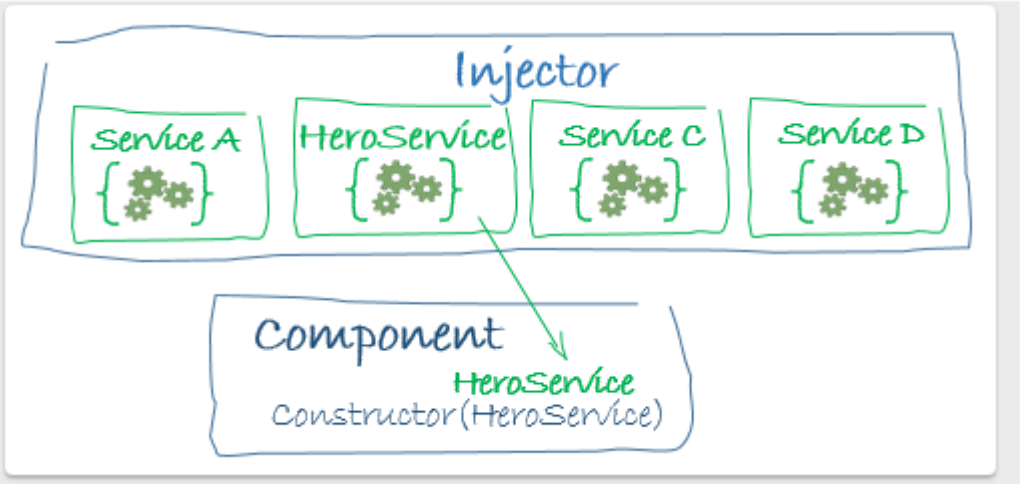
Kuvio 17. AppModule määritykset

3.3.2 Palvelut

Palvelut kansio sisältää useita erilaisia yksittäisiä palveluita, yksittäiset palvelut on nimetty komponenttien mukaan helpomman ylläpidettävyyden vuoksi (ks kuvio 13). Yksi palvelu pitää yleensä sisällään tietyn komponentin datan käsittelyt. Sovellus sisältää palvelut kaikille eri osa-alueen komponenteille, kuten käyttäjien hallinta, virheiden hallinta, relaatioiden hallinta, sekä pelkästään komponenttien käytössä olevat toiminnot.

Kun uuden komponentin instanssi luodaa, se määrittelee mitä palveluita, sekä muita riippuvuuksia kyseinen komponentti tarvitsee tarkistamalla sen konstruktorin parametrit. Angularin havaitessa komponentin riippuvuuden, se tarkistaa ensin injektorilta, että onko kyseisen palvelun instanssi jo olemassa. Jos instanssia ei ole vielä luotu, injektori luo sen ja lisää sen injektoitavaksi ennen palve-

lun palauttamista. Kun kaikki edellä mainitut on tehty, Angular pystyy kutsumaan konstruktoria, käyttäen palveluita argumentteinä (Understanding dependency injection 2022).



Kuvio 18. Esimerkki riippuvuus injektiosta (Understanding dependency injection 2022)

Projekti sisältää yksitoista palvelut tiedostoa, joista useat sisältävät useita metodeja eri tarkoituksiin. Ensimmäinen tärkeä palvelu toimii authentikoinnin sieppaajana, eli käytännössä palvelun tarkoituksena on tarkkailla lähteviä HTTP-pyyntöjä. Sieppaaja asettaa jokaiseen HTTP-pyyntöön Authorization otsakkeen, johon syötetään parametrinä JWT-token.

```
@Injectable()
export class AuthInterceptor implements HttpInterceptor {

  constructor(private router: Router) { }

  intercept(request: HttpRequest<any>,
    next: HttpHandler): Observable<HttpEvent<any>> {

    const idToken = localStorage.getItem('id_token');

    if (idToken) {
      const cloned = request.clone({
        headers: request.headers.set('Authorization',
          'Bearer ' + idToken)

      })

      return next.handle(cloned)
    }
    else {
      return next.handle(request)
    }
  }
}
```

Kuvio 19. Authentikoinnin sieppaaja

Parametrinä syötettyyn JWT-tokeniin sisältyy kolme arvoa (ks kuvio 19), ensimmäinen id_token on salattu token, joka sisältää tokeniin asetetun datan. Id_token dekoodataan palvelimen puolella (ks kuvio 20), jolloin saadaan tarkistettua JWT-tokenin sisältö. JWT-tokenin sisältöön kuuluu is_admin ja expires_at arvot. Is_admin on käytössä käyttäjän varmennuksessa, palvelin lukee sen arvon, jolloin se saa selville, että mitä pyyntöjä käyttäjä on oikeutettu tekemään. Jos käyttäjällä ei ole oikeuksia haluttuun pyyntöön, palvelin palauttaa virheilmoituksena virhekoodin 401 Unauthorized, joka tarkoittaa sitä että kyseisellä käyttäjällä ei ole oikeutta haluttuun pyyntöön (ks kuvio 21). Expires_at arvo on palvelimen puolella asetettu JWT-tokenin vanhentumisaika, jos vanhentumisaikaa ei päivitetä selainpuolelta sovellus kirjaa käyttäjän ulos kun vanhenemisaika umpeutuu. Vanhentumisajan umpeutuessa sovellus aukaisee dialogin, joka kysyy käyttäjältä, että haluaako hän pysyä kirjautuneena, jos käyttäjä jättää vastaamatta sovellus kirjaa käyttäjän ulos 60 sekunnin kuluttua.

Sovelluksessa käyttäjärooleja hallinoidaan tietokannassa vain yhdellä arvolla, is_admin on Boolean arvo, jossa 1 tarkoittaa true ja 0 tarkoittaa false, is_admin arvon ollessa 1, käyttäjällä on oikeudet kaikkiin pyyntöihin, jolloin sovellus on kokonaan käytettävissä. Is_admin arvon taas ollessa 0, käyttäjän oikeudet on rajattu vain tiettyihin pyyntöihin.

Key	Value
id_token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX2lkIjoiYzgzMW...
is_admin	true
expires_at	1677847663716

Kuvio 20. JWT-token tallennettuna selaimen paikalliseen varastoon

```
decode_data = jwt.decode(
    split[1], app.config['SECRET_KEY'], algorithms=['HS256'])
user = User.query.filter_by(
    public_id=decode_data.get('user_id')).first()
```

Kuvio 21. JWT-tokenin dekoodaus

```
def token_required(is_admin=False):
    def token_required_inner(f):
        @wraps(f)
        def decorated(*args, **kwargs):
            res = validate_token(request, is_admin)
            if not res.get('user'):
                return jsonify(res.get('message')), 401
            return f(res.get('user'), *args, **kwargs)
        return decorated
    return token_required_inner
```

Kuvio 22. Funktio joka suorittaa käyttäjä tarkistuksen suojatuille reiteille

Toinen tärkeä palvelu on authentikointi palvelu, authentikointi palvelun hoitaa käyttäjän rekisteröimisen, sekä kirjautumisen. Palvelu lähettää käyttäjää luodessa tiedot palvelimelle, jolloin palvelin lisää ne tietokantaan. Rekisteröinnin jälkeen käyttäjää voi käyttää sisäänkirjautumiseen. Sisäänkirjautumista tehdessä lomakkeeseen syötetään käyttäjänimi ja salasana, jolloin palvelu lähettää ne palvelimelle. Palvelin tarkistaa tietokannasta, että löytyykö tietokannasta vastaavia tunnuksia jos tunnukset on syötettyä oikein sovellus kirjaa käyttäjän sisälle. Palvelin palauttaa kirjautumisen yhteydessä JWT-tokenin.

Kun kirjautumisen yhteydessä selainpuoli vastaanottaa JWT-tokenin, palvelu kutsuu setSession metodia, metodi tallentaa JWT-tokenin selaimen paikalliseen varastoon ja viimeisenä käynnistää ajastimen, joka vahtii tokenin vanhentumisaikaa. Palvelu sisältää myös julkisen isLoggedIn metodin, tätä metodia kutsutaan aina kun sovelluksessa vaihdetaan sivua tai se päivitetään, metodin ainoa toiminto on tarkistaa kirjautumisen tila.

```
public setSession(authResult: any) {
  const expiresAt = moment().add(authResult.expiresIn, "second");

  localStorage.setItem("id_token", authResult.token);
  localStorage.setItem("expires_at", JSON.stringify(expiresAt.valueOf()));
  localStorage.setItem("is_admin", authResult.is_admin);

  this.startExpirationTimer();
}
```

Kuvio 23. setSession metodi authentikointi palvelussa

Kun valmis vuokrasopimus kirjataan, se lähetetään palveluita käyttämällä palvelimelle (ks kuvio 24). Palvelin ottaa sopimuksen tiedot ylös ja tallentaa ne tietokantaan. Selainpuoli käyttää addContract palvelussa observable ominaisuutta (ks kuvio 25). Observable on toiminto jota käytetään vahtimaan kommunikointia sovelluksen eri osien välillä (Observables in Angular 2022).

```
  this.contractService.addContract(newContract).subscribe((r) => {
    this.post_id = r;
    console.log(this.post_id);
    this.downloadFile(this.post_id);
  });
}
```

Kuvio 24. Sovellus kutsuu addContract palvelua

```
addContract(serializedForm: any): Observable<any> {
  return this.http
    .post<any>("http://127.0.0.1:5000/contracts/add", serializedForm)
    .pipe(map((response: any) => response));
}
```

Kuvio 25. addContract palvelu

Observable tarkoituksena on tässä kohtaa varmentaa milloin palvelin vastaa pyyntöön, kun createContract pyyntö lähetetään palvelimelle ja palvelin saa sen käsiteltyä, se palauttaa selainpuolelle sopimuksen pää avaimen. Kun selainpuoli vastaanottaa avaimen onnistuneesti, se lähettää uuden pyynnön palvelimelle. Tämä pyyntö sisältää ainoastaan juuri luodun sopimuksen pää avaimen. Palvelin hakee tietokannan useista tauluista kaiken sopimukseen liittyvän datan ja lisää ne HTML-tiedostoon. Palvelin kasaa tiedoston PDF-tiedostoksi ja palauttaa sen ladattavassa muodossa selainpuolelle.

3.3.3 Pääkomponentit

Sovellus sisältää lukuisia komponentteja, mutta pääpiirtettäin ne ovat hyvin yksinkertaisia. Sovellus on jaettu useisiin osiin komponenttien avulla, joista tärkeimpiin komponentteihin kuuluu sopimusten, talojen ja ominaisuuksien luomiseen tarkoitetut komponentit, sekä vuokralaisten tietojen hallinnoimiseen tarkoitettu komponentti.

Sopimusten luomiseen tarkoitettu komponentti, joka on sovelluksessa nimetty create-contracts komponentiksi sisältää sen sopimuksen luomiseen tarvittavan älyn, lomakkeet sekä lomakkeen tyylittelyt. Komponentti sisältää myös kaksi lapsi komponenttiä, jotka hallinnoivat lomakkeeseen kuuluvia dialogeja. Komponentin Typescript tiedosto on vastuussa dynaamisista muuttujista, palvelukutsuista, ulkoisista API pyynnöistä, sekä lomakkeen toiminnallisuudesta. Typescript tiedostossa määritellään komponentin lomakeryhmä, sekä tarvittaessa asetetaan ryhmän kentille oletusarvot (ks kuvio 26).

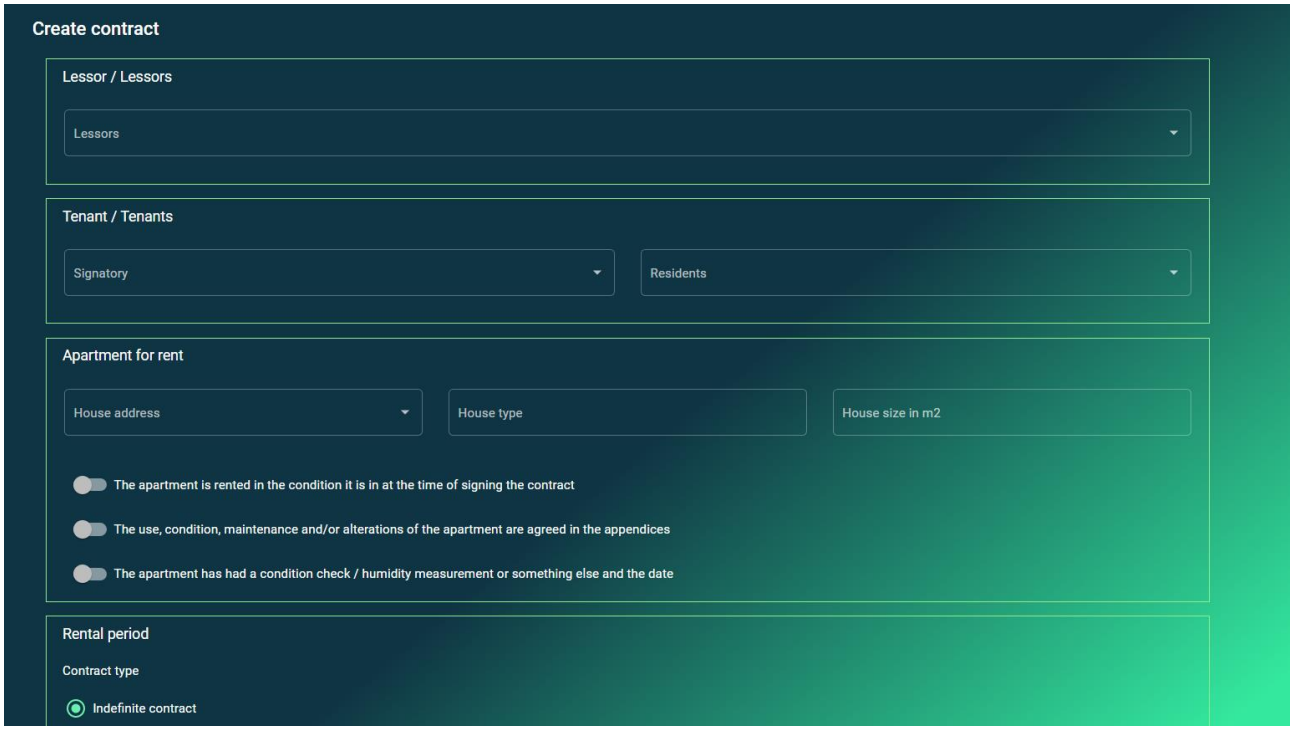
```
contractForm = new FormGroup({
  //Vuokranantaja / Vuokranantajat
  lessor_id: new FormControl(),

  //Vuokralainen / vuokralaiset
  signatory_id: new FormControl(),
  resident_id: new FormControl(),

  //Vuokrahuoneisto
  house_id: new FormControl(),
  house_type: new FormControl(),
  house_size: new FormControl(),
  house_pricePerSquare: new FormControl(),
  rentalStateChecked: new FormControl(false),
  rentalAgreementChecked: new FormControl(false),
  rentalCheckupChecked: new FormControl(false),
  rentalCheckupDate: new FormControl(),

  //Vuokra aika
  startDate: new FormControl(),
  // signDate: new FormControl(),
  endDate: new FormControl(),
  indefiniteContract: new FormControl(true),
  otherNoticePeriod: new FormControl(false),
  otherNoticePeriodDate: new FormControl(),
  otherDueDate: new FormControl(false),

  //Vuokran määrä
  house_rent: new FormControl(),
```

Kuvio 26. Lomakeryhmän määritys

Komponentin HTML-templaatti muodostuu useista eri lomakekentistä, lomakekentät ovat dynaamisia ja niistä suurimmalla osalla on jonkinlaista älyä. Kenttä voi koostua esimerkiksi listasta, jonka palvelin noutaa tietokannasta ja palauttaa sen selainpuolelle. Lomake on luotu mahdollisimman automatisoiduksi, sillä sen tarkoituksena on helpottaa vuokrasopimuksien luomista. Kun lomakesivu ladataan Angular kutsuu ngOnInit metodia, ngOnInit on Angularin elinkaaren koukku (lifecycle hook) eli metodi, jota kutsutaan kun Angular saa luotua komponentin ja asetettua sen ominaisuuden. Komponentissa ngOnInit() metodiin on asetettu palvelu kutsuja (ks kuvio 27). Palvelut palauttavat komponentille kaikki vuokrattavissa olevat asunnot, kaikkien vuokralaisten tiedot, kaikki vuokranantajat, sekä listan erilaisista vuokrankorotuksiin käytettävistä indekseistä.

```
ngOnInit() {
  this.initialValues = this.contractForm.value;
  this.houseService.returnAvailableHouse().subscribe((r) => {
    this.houses = r;
  });
  this.tenantService.returnTenant().subscribe((r) => {
    this.residents = r;
    this.signatories = r;
  });

  this.lessorService.getLessors().subscribe((r) => {
    this.lessors = r;
  });

  this.getIndexesList();
}
```

Kuvio 27. Create-contracts ngOnInit() metodi



Kuvio 28. Käyttäjänäkymä sopimusta luodessa

Lomakkeen avautuessa komponentti on jo vastaanottanut kaikki lomakkeen täyttämiseen tarvittavat tiedot. Sopimukseen käyttäjä voi valita haluamansa asunnon esimerkiksi osoitteen, koon tai tyypin perusteella, käyttäjän valitessa talon, komponentti kutsuu metodia getHouseData(), joka palveluita hyödyntämällä hakee kyseisen asunnon tiedot palvelimelta (ks kuvio 29).

```
getHouseData(house_id: any) {
  this.houseService.getHouseData(house_id).subscribe((r) => {
    this.house_data = r;
    this.contractForm.controls["house_size"].setValue(this.house_data.size);
    this.contractForm.controls["house_type"].setValue(this.house_data.type);
    this.contractForm.controls["house_rent"].setValue(this.house_data.price);
    this.contractForm.controls["house_pricePerSquare"].setValue(
      this.house_data.price * this.house_data.size
    );
  });
}
```

Kuvio 29. Metodi joka noutaa valitun asunnon tiedot

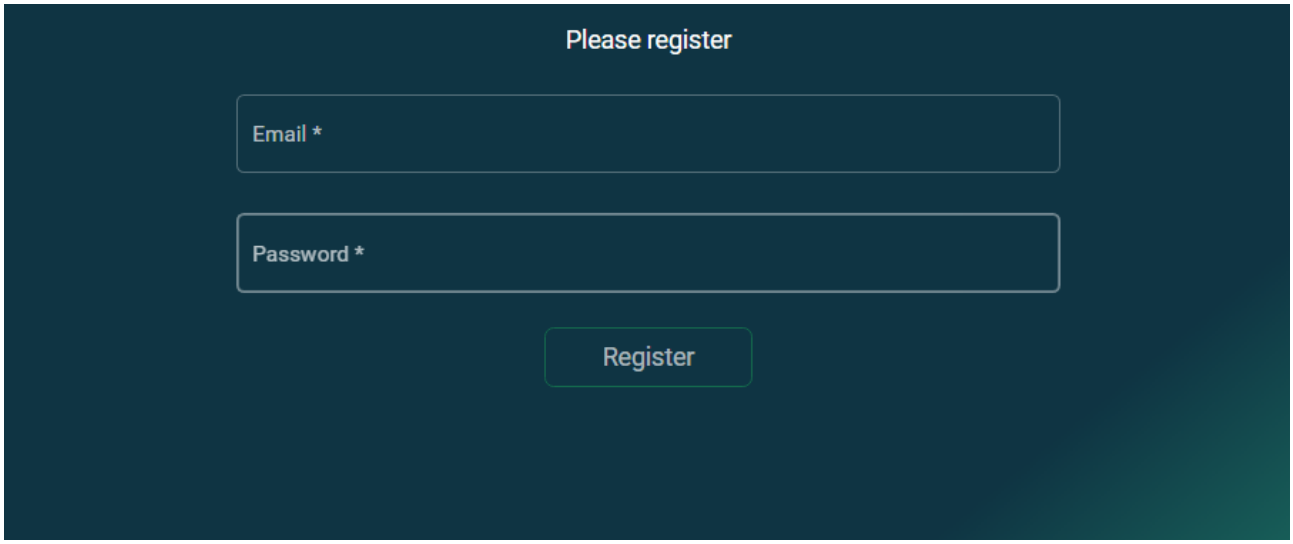
Kun palvelin palauttaa selaimelle asuntoon liittyvät tiedot, metodi asettaa vastaanottamansa tiedot lomakekenttiin. Näin ollen käyttäjän valitessa asunnon, sovellus täydentää automaattisesti taloon tietoihin liittyvät lomakekentät. Edellämainittua periaatetta on käytetty koko lomakesivun rakentamiseen. Käyttäjän valitessa vuokranantajan, vuokranantajaan liittyvät kentät täyttyvät automaattisesti, sekä käyttäjän valitessa vuokralaisen vuokralaiseen liittyvät kentät täyttyvät automaattisesti.

Lomakkeen vuokrankorotuksiin sijoittuvassa osiossa, käyttäjä pystyy valitsemaan pudotusvalikosta erilaisia vuokrankorotuksiin käytettäviä perusindeksejä. Perusindeksit on noudettu Tilastokeskuksen julkisesta rajapinnasta, perusindeksin valinta toimii sovelluksessa samalla periaattella kuin asuntojen, vuokralaisten tai vuokranantajien valinta. Käyttäjä poimii pudotusvalikosta haluamansa indeksin, indeksin valinnassa kutsutaan metodia, joka lähettää valitun indeksin parametreinä palvelimelle. Kun palvelin vastaanottaa parametrit se lähettää ne pyyntönä eteenpäin Tilastokeskusen avoimelle rajapinnalle, joka palauttaa kyseisen vuokrankorotus indeksin viimeisimmän pisteluvun. Palvelin vastaanottaa indeksin pisteluvun ja palauttaa sen selainpuolelle, metodi hyödyntää taas Observable ominaisuutta ja vastauksen saadessaan se asettaa valitun indeksin pisteluvun lomakekenttään.

```
getIndexesList() {
  this.contractService.getIndexesList().subscribe((r) => {
    this.indexList = r;
  });
}

selectIndex(index: any) {
  this.contractService.selectedIndex(index).subscribe((r) => {
    this.selectedIndex = r;
    this.contractForm.controls["indexValue"].setValue(
      this.selectedIndex.value[0]
    );
  });
}
```

Kuvio 30. Vuokrankorotus indeksien käsittely

Pääkomponentteihin kuuluu myös rekisteröinti ja kirjautumis komponentit, rekisteröinti komponentti sisältää lomakkeen johon käyttäjä saa syötettyä sähköpostiosoitteensa, sekä haluamansa salasanan. Käyttäjän painaessa rekisteröinti painiketta sovellus käyttää authentikointipalvelua joka lähettää käyttäjän syöttämät tiedot palvelimelle, jonka jälkeen palvelin kirjaa tunnukset tietokantaan. Käyttäjien kirjaamisessa tietokantaan on palvelin puolella käytetty Flask-Bcrypt kirjastoa, kirjaston tarkoituksena on muuttaa rekisteröidyn käyttäjän julkinen avain salatuksi merkkijonoksi, sekä tallentaa salasana salatussa muodossa tietokantapalvelimelle, näin ollen jos joku ulkopuolinen taho jostain syystä pääsisi käsiksi tietokantapalvelimelle, tallennetut käyttäjätunnukset pysyvät kuitenkin turvassa salauksen ansiosta. Kun käyttäjä rekisteröityy uusilla tunnuksilla onnistuneesti, sovellus navigoi sen automaattisesti kirjautumissivustolle (ks kuvio 32).



Kuvio 31. Rekisteröinti komponentin käyttäjänäkymä

```
registerUser() {
  let accountData = this.registerForm.getRawValue()
  this.authService.createAccount(accountData).subscribe(r => {
    this.router.navigate(['/login'])
    console.log(r)

  })

}
```

Kuvio 32. Käyttäjän rekisteröimiseen tarkoitettu metodi

Kirjautumiseen käytettävä komponentti on rakenteeltaan hyvin samanlainen rekisteröintikomponentin kanssa. Kirjautumis komponentti sisältää lomakekentät sähköpostille ja salasanalle. Kun käyttäjä on syöttänyt tunnuksensa ja painaa kirjautumis painiketta, sovellus kutsuu logIn metodia, logIn metodi vastaanottaa lomakkeelle syötetyn datan ja syöttää ne parametreinä kirjautumis palvelulle. Palvelu lähettää pyynnön palvelimelle, pyyntö sisältää parametreinä käyttäjän syöttämät tunnukset, palvelin tarkistaa tietokannasta käyttäjätunnuksien todenperäisyyden, jos käyttäjä löytyy tietokannasta palvelin palauttaa käyttäjienhallintaan käytettävän JWT-tokenin.

3.3.4 Putket

Sovelluksessa on käytössä putkia erilaisia haku ja filtteröinti toimintoja varten. Putki on toiminnaltaan suhteellisen yksinkertainen, putkelle annetaan parametreinä ‘items’ taulukko jonka käyttäjä haluaa filtteröidä, taulukon lisäksi sille annetaan string tyyppinen ‘searchText’ muuttuja, joka toimii haun ehtona. Kun käyttäjä kirjoittaa hakukenttään tekstiä, putki palauttaa uuden taulukon, joka sisältää vain ne ‘items’ taulukon elementit, jotka sisältävät ‘searchText’ merkkijonon kirjainkoosta riippumatta.

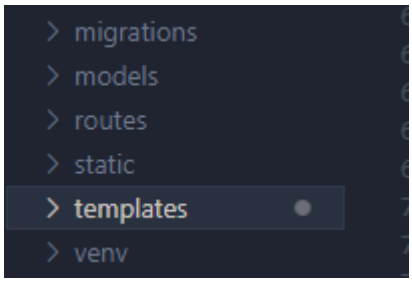
```
transform(items: any[], searchText: string): any[] {
  if (!items) {
    return [];
  }
  if (!searchText) {
    return items;
  }
  searchText = searchText.toLocaleLowerCase();

  return items.filter(it => {
    return it.toLocaleLowerCase().includes(searchText);
  });
}
```

Kuvio 33. Yksinkertainen hakufiltteri putki

3.4 Flask palvelinpuoli

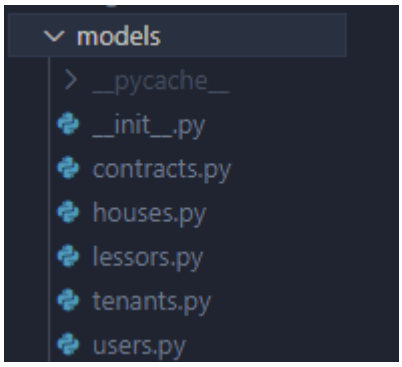
3.4.1 Rakenne

Sovelluksen palvelinpuoli on jaoteltu viiteen osioon, mallit, templaatit, reitit, sekä migraatiot.



Kuvio 34. Palvelinpuolen kansiorakenne

Mallit koostuvat tietokannan määrittelyihin käytetyistä tiedostoista. Nämä tiedostot sisältävät tietokanta taulujen sarakkeiden ja niiden tyyppien määrittelyt. Mallit on jaettu useisiin eri tiedostoihin ylläpidettävyyden helpottamiseksi (ks kuvio 35), sovelluksesta löytyy omat tiedostot jokaista eri taulua varten. Yksittäinen malli voi koostua useasta eri luokasta, jolloin yksi luokka muodostaa yhden taulun. Malli tiedosto sisältää tyypillisesti yhden pääluokan ja lisäksi sille tarkoitettujen relaatioiden hallintaan käytettävän ala-luokan.

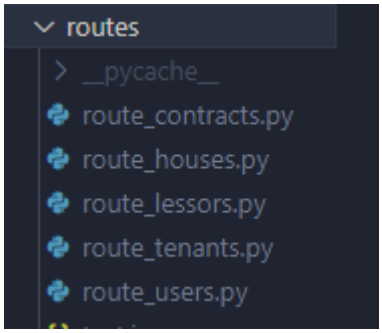


Kuvio 35. Mallit kansio sisältöineen

Templaatit on kansio, jonne on tallennettu kaikki palvelinpuolen HTML-dokumentit, templaatteina sovelluksessa toimii valmiin vuokrasopimuksen tiedot tai suoritetun vuokrantarkastuksen tiedot.

Nämä dokumentit renderöidään käyttämällä Jinja2 templaatti moottoria. Jinja2 moottoria mahdollistaa Python-koodin upottamisen HTML-dokumentteihin.

Reitteihin kuuluvat kaikki palvelinpuolella määrittellyt reitit. Sovelluksessa reittien tarkoituksena on vastaanottaa selainpuolelta saapuvia pyyntöjä ja käsitellä ne halutulla tavalla. Reitit on jaettu sovelluksessa useaan eri tiedostoon, jotta sovelluksen ylläpidettävyys säilyy helppona. Sovellus sisältää omat reitti tiedostonsa kaikille sovelluksen eri osa-alueille. Sovellus on rakennettu siten, että selainpuolella käytettävät palvelut lähettävät tyypillisesti pyyntöjä vain yhteen reitti tiedostoon kuuluvien reittien kanssa. Esimerkkinä selainpuolen sopimus palvelu lähettää pyyntöjä ainoastaan reitteihin, mitkä on palvelin puolella nimetty sopimuksen reiteiksi.



Kuvio 36. Reitit kansion sisältö

Migraatiot sisältävät kaikkien tietokantaan ajettujen muutosten tiedot. Kun tietokantaan lisätään uusi solu, kokonaan uusi taulu tai ajetaan päivitys johonkin valmiiksi olemassa olevaan tauluun, siitä luodaan uusi migraatio tiedosto. Tietokannan sisältäessä dataa, joka halutaan säilyttää tietokantaa päivitettäessä on migraatioiden hyödyntäminen mielestäni paras vaihtoehto. Jos tietokantaa päivitettäessä jokin menee vikaan, migraatioilla saa helposti otettua käyttöön vanhemman version tietokannasta, tällöin riski tiedon menettämiseen on huomattavasti pienentynyt.

3.4.2 Mallit

Sovelluksessa malleina toimivat tiedostot taloille, sopimuksille, vuokranantajille, vuokralaisille sekä käyttäjille. Mallin tärkein ominaisuus on määritellä tietokanta taulun sisältämät kentät, sekä niiden tyypit. Malliin sisältyvät luokat toimivat tietokannassa tauluina, mallissa määritellään tauluille nimet, sekä ominaisuudet. Mallia on myös mahdollista muokata jälkikäteen, jos ylläpitäjä haluaa

lisätä tietokanta tauluun sarakkeen, se onnistuu tekemällä muutoksen ensin malliin ja sen jälkeen ajamalla sen migraationa tietokantaan. Tällä tavoin ylläpitäjä pystyy myös palaamaan vanhaan tietokanta version virheen sattuessa. Malli tiedosto sisältää myös yleensä konstruktorin, jota kutsutaan kun luokasta luodaan olio. Konstruktorin tarkoituksena on alustaa olio kyseisestä luokasta.

Käytän esimerkkinä sovelluksen talo luokkaa. Tämä esimerkki määrittelee talo-luokan, siihen sisältyy useita db.Column määritteitä, joilla määritellään tietokantataulun sarakkeita. Sarakkeet vastaavat talo-luokan ominaisuuksia, jolloin jokainen sarakemäärittely kuvaa tietokantarakennetta ja sarakkeiden ominaisuuksia, kuten tyyppiä tai kokoa. Luokan id ominaisuus on määritelty 'primary_key=True' parametrillä, mikä tarkoittaa että se on taulun ensisijainen avain.

'__init__' -funktio on luokan konstruktori. Konstruktori luo kutsuttaessa uuden talo olion. Konstruktorin tarkoituksena on alustaa oliolle sen ominaisuudet, eli tässä esimerkissä osoite, saatavuus, hinta, koko ja tyyppi, nämä ominaisuudet vastaavat tietokantataulun sarakkeita.

'obj_to_dict' -metodi palauttaa dictionary muodossa olion attribuutit ja arvot. Tätä metodia käytetään, kun olion tiedot halutaan muuntaa JSON-muotoon, kun olion arvot saadaan helposti muunnettua JSON-muotoon, ne voidaan helposti lähettää selainpuolelle.

```
class House(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    address = db.Column(db.String(50))
    isAvailable = db.Column(db.Boolean)
    price = db.Column(db.Integer)
    size = db.Column(db.Integer)
    type = db.Column(db.String(30))

    def __init__(self, address, isAvailable, price, size, type):
        self.address = address
        self.isAvailable = isAvailable
        self.price = price
        self.size = size
        self.type = type

    def obj_to_dict(self):
        return {
            "id": self.id,
            "address": self.address,
            "isAvailable": self.isAvailable,
            "price": self.price,
            "size": self.size,
            "type": self.type
        }
```

Kuvio 37. Sovelluksen talo luokka.

3.4.3 Templaatit

Templaatit sisältävät Jinja2 templaatteina toimiva HTML tiedostoja, sovelluksessa näitä templaatteja käytetään pohjana ladattaville PDF-tiedostoille. Templaatteihin on rakennettu HTML-dokumentti, jonka kentät täytetään käyttäjän syöttämällä datalla. Data noudetaan templaatteihin tietokanta kyselyillä ja se tallennetaan Jinja2 muuttujiin, kun tietokantakysely on tehty ja tietokannasta noudettu data on tallennettu muuttujaan, se voidaan näyttää HTML-dokumentin kentissä (ks kuvio 38).

```
<div class="row justify-content-md-center">
    <h4>Indeksit</h4>
    <table class="table table-bordered">
        <thead>
            <tr>
                <th class="col">Indeksi</th>
                <th class="col">Alkuperäinen indeksin pisteluku</th>
                <th class="col">Uusi indeksin pisteluku</th>
            </tr>
        </thead>
        <tbody>

            <tr>
                <td>{{contract.indexSelected}}</td>
                <td>{{contract.indexValue}}</td>
                <td>{{review.newIndexValue}}</td>
            </tr>

        </tbody>
    </table>
```

Kuvio 38. Esimerkki Jinja2 templaatissa näytetystä datasta.

Sovelluksessa on käytössä kaksi templaattia, ensimmäinen muuttaa HTML-dokumentin PDF-muotoon ja palauttaa sen palvelimelle, käyttäjän luodessa uuden vuokrasopimuksen kenttien tiedot tallentuvat tietokantaan, josta ne haetaan templaatill. Tämän jälkeen palvelin muuttaa sen PDF-muotoon ja palvelin palauttaa sen selainpuolelle ladattavana tiedostona, tämän takia käyttäjä saa luomansa vuokrasopimuksen tulostettavana versiona automaattisesti.

```
def returnReviewPdf(id):
    contractToDownload = Contract.query.filter(Contract.id == id).first()

    reviewRelations = Rentreviewrelations.query.filter(Rentreviewrelations.contract_id == id).order_by(desc(Rentreviewrelations.id)).first()


    review = Rentreviews.query.filter(Rentreviews.id == reviewRelations.review_id).first()

    contractHouse = House.query.filter(
        House.id == contractToDownload.house_id).first()


    html = render_template(
        'review.html', contract=contractToDownload, review=review, house=contractHouse)

    review = HTML(string=html).write_pdf()

    response = make_response(review)
    response.headers['Content-Type'] = 'application/pdf'
    response.headers['Content-Disposition'] = f'inline; filename=review_{id}.pdf'

    return review
```

Kuvio 39. Funktio joka palauttaa tiedoston PDF-muodossa

Toinen templaatti toimii hyvin samalla periaattella kuin ensimmäinen. Kun käyttäjä suorittaa vuokrantarkastuksia jo olemassa oleviin sopimuksiin, vuokrantarkastuksessa käytetyt indeksit tallentuvat uusilla arvoilla tietokantaan, palvelin noutaa tietokannasta uudet arvot ja upottaa ne HTML-dokumenttiin. Kun templaatin kentät on täytetty, palvelin muuttaa sen PDF-tiedostoksi ja palauttaa sen selainpuolelle ladattavassa PDF-muodossa. Templaatti on luotu niin, että käyttäjä pystyy lähettämään lataamansa tiedoston suoraan kirjeenä tai sähköpostina vuokralaisille, tiedosto näyttää juuri suoritetun vuokrantarkastuksen vuokrankorotukset ja niihin käytetyt perustelut, sekä arvot.

3.4.4 Reitit

Reitit kansio sisältää Python-moduuleja, joihin kuuluu kaikki sovelluksessa määritellyt reitit. Moduulit on eritelty toisistaan niiden käyttötarkoituksen perusteella, joista tärkeimpinä on vuokrasopimuksien, vuokrakohteiden, käyttäjien ja vuokralaisten hallinnassa käytetyt Moduulit. Selainpuolella määritelly palvelut kutsuvat aina vain yhtä reittiä kerrallaan. Esimerkiksi: kun käyttäjä on kirjautunut sisään ja avaa sopimukset välilehden sovelluksesta, selainpuoli lähettää pyynnön palvelimelle tiettyä URL-osoitetta käyttäen. Palvelin näkee että kyseiseen URL-osoitteeseen on tullut pyyntö ja suorittaa siihen sidotun funktion (ks kuvio 40).

```
@bp_contracts.route('/contracts/return', methods=['GET'])
@token_required(is_admin=False)
def getContracts(current_user):
    contracts = Contract.query.all()
    contractsList = []
    for i in contracts:
        contractsList.append(i.obj_to_dict())

    return json.dumps(contractsList)
```

Kuvio 40. Funktio joka palauttaa sopimukset JSON muodossa.

Reitit kansion moduulit käyttävät Flaskin “app.route()” dekoraattoria määrittääkseen polun ja sen toiminnallisuuden. Kun käyttäjä navigoi sivulla olevaan polkuun, palvelin käynnnistää kyseisen moduulin ja suorittaa sen määrittelemät toiminnot. Yleisesti nämä sisältävät tietokannan kyselyitä, käyttäjän syötteen käsittelyjä tai sivun renderöintiä.

Moduulit käyttävät myös muita Flaskin toimintoja, kuten “render_template()” funktiota, joka mahdollistaa HTML-sivujen renderöinnin sovelluksessa. Funktio ottaa vastaan templaatin nimen, sekä datan jolloin se voidaan renderöidä HTML-muotoon. Moduuleissa hyödynnetään myös “request” komentoa käyttäjän syöttämien tietojen noutamiseen. Yhteenvetona siis reitit kansio sisältää moduuleja, jotka määrittelevät sovelluksen eri reittien käyttäytymisen ja toiminnallisuuden,

```python
@bp_contracts.route('/indexes/api/get', methods=['GET'])
def indexApiGet():
    resp = requests.get(
        'https://pxdata.stat.fi:443/PxWeb/api/v1/fi/StatFin/khi').json()

    dataList = []
    for i in resp:
        if 'kuukausi' in i['text']:
            d = {
                "text": i['text'][8:].split(',')[0],
                "id": i['id'],
                "updated": i['updated']
            }
            dataList.append(d)

    return dataList
```

Kuvio 41. Esimerkki moduulista.

4 Tulokset ja pohdinta

Sovellus täytti toimeksiantajan asettamat vaatimukset, projektia tehdessä ilmaantui myös välillä uusia potentiaalisia toimintoja, joita rakennettiin tarvittaessa lisää. Sovelluksen perustoiminta sisälti kaikki vaadittavat ominaisuudet, vuokranantajana toimiva toimeksiantaja saa automatisoinnilla nopeutettua työtehtäviänsä, mikä oli koko projektin tavoitteena. Vuokranhallintaan käytettävä aika pieneni vaikka vuokrattavien kohteiden määrä kasvaisi entisestään.

Sovellus soveltuu loistavasti jatkokehitystä varten, vaikka toimintoja on jo automatisoitu suuresti aikaisempaan manuaaliseen työskentelyyn verrattuna, voisi automaattisesti suoritettavia toimintoja olla silti enemmänkin, kuten vuokrantarkastuksien suorittaminen ajastimella vuosittain ja tarkastuksen jälkeen, tiedot voisivat lähteä automaattisesti sähköpostilla.

Opinnäytetyön suunnittelu aloitettiin helmikuussa 2023, sovellus oli silloin vielä rakentamisvaiheessa, sovelluksen rakennus alkoi lokakuussa 2022 ja se valmistui huhtikuussa 2023. Kirjoittaja aloitti opinnäytetyön parissa työskentelyn helmikuun 2023 loppupuolella, projektin ollessa vielä kesken. Kirjoitusprosessia hidasti työharjoitteluun, sekä yhden kurssin tehtäviin vaadittava aika, joka vei suurimman osan kirjoittajan vapaa-ajasta harrastuksien lisäksi. Täyspäiväisesti kirjoittaja pääsi työskentelemään opinnäytetyön parissa vasta huhtikuun 2023 puolessa välissä.

Kirjoittaja oli todella tyytyväinen valittuihin teknologioihin ja oppi paljon ohjelmistokehitykseen liittyvistä prosesseista ja menetelmistä. Angularin ja Python Flaskin yhdistelmä toimii erinomaisesti yhdessä, koska ne tarjosivat kattavan ratkaisun web-sovellusten rakentamiseen. Angularilla saatiin luotua käyttäjäystävällinen käyttöliittymä, joka kommunikoi taustapalveluiden kanssa Python Flaskin avulla vaivattomasti. Python Flask puolestaan tarjosi mahdollisuuden rakentaa palveluita, jotka vastaavat käyttöliittymän tarpeisiin ja pystyvät hallitsemaan tietokannan käyttöä.

Lähteet

Dependency injection in Angular. 2020. Referred 21.2.2023. https://www.technoarchsoftwares.com/blog/dependency-injection/

Introduction to Angular concepts. 2022. Referred 21.2.2023. https://angular.io/guide/architecture

Introduction to JSON Web Tokens. N.d. Referred 27.2.2023. https://jwt.io/introduction

Introduction to modules. 2022. Referred 21.2.2023. https://angular.io/guide/architecture-modules

Introduction to services. 2022. Referred 21.2.2023. https://angular.io/guide/architecture-services

Jinja2 Explained in 5 Minutes. 2018. Referred 16.4.2023. https://codeburst.io/jinja-2-explained-in-5-minutes-88548486834e

Major Advantages of Using MySQL. 2023. Referred 23.2.2023. https://www.datamation.com/storage/8-major-advantages-of-using-mysql/

Observables in Angular. 2022. Referred 14.3.2023. https://angular.io/guide/observables-in-angular

Python development survey. 2021. Referred 22.2.2023. https://lp.jetbrains.com/python-developers-survey-2021/

SQLAlchemy 2.0 Documentation. 2023. Referred 23.2.2023. https://docs.sqlalchemy.org/en/20/intro.html

Stackoverflow Developer Survey. 2022. Referred 23.2.2022. https://survey.stackoverflow.co/2022/#most-popular-technologies-webframe

The Most Popular Databases for 2022. 2022. Referred 23.2.2023. https://learnsql.com/blog/most-popular-databases-2022/

Understanding dependency injection. 2020. Referred 1.3.2023. https://angular.io/guide/dependency-injection

What is Angular. 2022. Referred 21.2.2023. https://angular.io/guide/what-is-angular

What is Flask Python. 2021. Referred 22.2.2023. https://pythonbasics.org/what-is-flask-python/

What is MySQL. N.d. Referred 23.2.2023. https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html

Who uses MySQL. 2020. Referred 23.2.2023. https://stackshare.io/mysql

The Most Popular Databases for 2022. 2022. Referred 23.2.2023. https://learnsql.com/blog/most-popular-databases-2022/

Liitteet