LAB University of Applied Sciences

Kasvaminen web-ohjelmistokehittäjänä

Oppimispäiväkirja

LAB-ammattikorkeakoulu
Tieto- ja viestintätekniikan insinööri (AMK)
2024
Panu Ohra-aho

Tiivistelmä

Tekijä(t)	Julkaisun laji	Valmistumisaika
Panu Ohra-aho	Opinnäytetyö, AMK	2024
	Sivumäärä	
	55	

Työn nimi

Kasvaminen web-ohjelmistokehittäjänä

Oppimispäiväkirja

Tutkinto ja koulutusala

Tieto- ja viestintätekniikka, Insinööri (AMK)

Toimeksiantajaorganisaatio (jos opinnäytetyöllä on toimeksiantaja)

Casamedia oy

Tiivistelmä

Opinnäytetyön tavoite on luoda uusia ominaisuuksia ja kehittää vanhoja ominaisuuksia eteenpäin asiakkaan ohjelmistoprojektissa. Opinnäytetyö on toteutettu päiväkirjamallisena mikä tarkoittaa, että opinnäytetyön oheen kuuluu päiväkirja 13 viikon työjakson ajalta. Opinnäytetyön toimeksiantajana toimi Casamedia oy.

Työn aikana tutustuttiin kaikkiin web-projektissa tarvittaviin työkaluihin, kuten Flutter-kehys, Dart-kieli ja Express-kehys. Työssä käydään myös läpi työkaluja ja menetelmiä, joista olisi ollut projektin teossa hyötyä. Tärkeimpänä ollen testausmenetelmät. Työjakson tehtäviin kuului ohjelmistokehitystä ja ohjelmiston virheiden korjaamista niin etu- kuin taustajärjestelmässä. Päivityksiä tehtiin sekä vanhojen tuotosten pohjalta että kokonaan alusta asti. Suurimpana esimerkkinä nostetaan esiin kuvanlataaja-komponentti ja kuvapankki.

Työn kohteena olleet ominaisuudet saatiin suurimmalta osin toteutettua. Opinnäytetyöprosessi opetti ammatillista työskentelyä osana suurempaa projektia, tiedonhaun tehostamista ja eri työkalujen tärkeyttä ohjelmistokehityksessä.

Asiasanat

CRUD, Dart, Flutter, komponentti, NPM, SDK

Abstract

Author(s)	Type of Publication	Published
Panu Ohra-aho	Thesis, UAS	2024
	Number of Pages	
	55	

Title of Publication

Growth as web-developer

Learning diary

Degree, Field of Study

Bachelor of Information and Communication Technologies

Organisation of the client (if the thesis work is commissioned by another party)

Casamedia oy

Abstract

The goal of this thesis is to develop new features and further develop existing features in the client's software project. The thesis is implemented with diary model, which means that a diary is included alongside the thesis covering a 13-week work period. The thesis was commissioned by Casamedia Oy.

During the project, all the tools necessary for the web project were explored, such as the Flutter framework, Dart language, and Express framework. The thesis also covers tools and methods that would have been useful in the project, with the most important being testing methods. The tasks during the work period included software development and bug fixing in both the front-end and back-end systems. Updates were made based on both previous work and from scratch. The most significant example highlighted is the image uploader component and the image bank.

Most of the features targeted in the project were successfully implemented. The thesis process taught professional working as part of a larger project, the improvement of information retrieval, and the importance of various tools in software development.

Keywords

CRUD, Dart, Flutter, NPM, SDK, widget

Sisällys

Sanasto

CRUD	Palvelinoperaatiot, joilla manipuloidaan tietokannan dataa.
Dart	Ohjelmointikieli, jolla Flutteria kirjoitetaan.
Flutter	Ohjelmointikehys, jolla käyttöliittymä luodaan.
Komponentti	Dart-olio, joka suorittaa yhtä tehtävää käyttöliittymässä.
NPM	Pakettienhallintaohjelma, jolla ladataan moduuleja.
SDK	Ohjelmistokehityspaketti, joka sisältää kehitystyökaluun tarvittavat tiedostot.

1 Johdanto

Tietotekniikan ala kasvaa koko ajan. Uusia ohjelmointikieliä, kehyksiä ja muita työkaluja luodaan jatkuvasti. Alan insinöörin on hyvä pysyä tietoisena uusista mahdollisuuksista, joita nämä työkalut tuovat tullessaan, mutta myös tarttua tilaisuuteen syventää omaa osaamistaan. Alati laajenevassa ympäristössä yksittäisen osa-alueen asiantuntemus on yhä arvokkaampaa.

Opinnäytetyön tavoite on luoda uusia ominaisuuksia ja kehittää vanhoja ominaisuuksia eteenpäin asiakkaan ohjelmistoprojektissa. Opinnäytetyö on toteutettu päiväkirjamallisena mikä tarkoittaa, että opinnäytetyön oheen kuuluu työpäiväkirja 13 viikon työjakson ajalta. Päiväkirja dokumentoi jokaisen työpäivän tavoitteen ja työn tuloksen. Päiväkirjasta nostetaan esiin mielenkiintoisimpia toteutuksia opinnäytetyössä.

Työjakso suoritettiin Casamedian palveluksessa. Casamedia on Suomessa toimiva digialan palveluita tarjoava yritys. Työ kohdistui ohjelmaprojektiin, jota oli luomassa useita muita työntekijöitä. Ohjelmaprojekti on yrityksille ja eri alojen osaajille suunnattu organisointisovellus. Sovelluksen pääominaisuuksiin kuuluu datan tallettaminen ja tiedon jakaminen muille sovelluksen käyttäjille tai yhteyshenkilöille.

Varsinaiset työtehtävät vaihtelivat uusien ominaisuuksien luomisesta virheiden korjaamiseen sovelluksessa niin käyttöliittymässä kuin taustajärjestelmässä. Tämä tarjosi alustan asettaa sopeutumiskyky testiin ja kerätä tietoa laajalti käytössä olleista työkaluista. Pääpaino on kuitenkin käyttöliittymän luonnissa Flutter-kehyksellä, mikä mahdollistaa yksityiskohtaisemman osaamisen kehittämisen.

Casamedia aloitti toimintansa vuonna 1991. Yritys tuotti aluksi pienimuotoisia digipalveluita ja tietotekniikan elektroniikkaa. Myöhemmin yritys on laajentunut moniosaisten tietotekniikan palvelujen tarjoajaksi. Yritys toimii tällä hetkellä Kankaanpäästä. (Juhantalo 2024.)

2 Työympäristön käyttöönotto

2.1 Tyhjän projektin luominen

Flutter tarvitsee toimiakseen Flutter SDK-tiedostot. Nämä voi ladata maksutta Flutterin verkkosivuilta. Lataamisprosessin jälkeen Flutter on teoriassa käyttökunnossa. Joissain tapauksissa voidaan joutua lataamaan muita tiedostoja, kuten lisäosia valitsemaansa koodieditoriin. Esimerkiksi jotta Flutter toimisi mutkattomasti Visual Studio Code-ohjelmalla, käyttäjän voi olla tarpeen ladata lisäosia tuohon ohjelmaan (Zakhour & Lougheed 2024).

Flutterin projektipohja luodaan ajamalla komentopaneelissa flutter create-komento. Esimerkkikomento näkyy kuvassa (Kuva 1). Komennon jälkeen paneeliin kirjoitetaan projektin nimi.

```
$ flutter create esimerkki
Creating project esimerkki...
```

Kuva 1. Komento Flutter-projektin luomiselle

Tämä luo valmiin Flutter-sovelluksen kansioon, jossa komento suoritettiin. Kyseinen sovellus sisältää otsikon, tekstiä ja painikkeen, jolla sovelluksen keskellä olevaa lukua voi kasvattaa. Pohjasovelluksen käyttöliittymä esitetään kuvassa (Kuva 2).

Kuva 2. Pohjasovelluksen käyttöliittymä

Tämän pohjasovelluksen tarkoituksena on antaa kaikki tarvittava toimivan Flutter-sovelluksen kehittämiseen ja tarjota harjoittelualusta aloitteleville ohjelmoijille (Windmill 2020, 55).

Työprojektissa Flutteria käytetään käyttöliittymän luomiseen ja Express-kehystä taustajärjestelmän luomiseen. Toisin kuin Flutterissa, Expressissä ei ole sisään rakennettua projektipohjaa. Express voidaan saada lataamisen jälkeen toimintakuntoon yhdellä tiedostolla ja muutamalla rivillä koodia. Kuva tällaisesta koodista on kuvassa (Kuva 3). Tämän koodin tulee pitää sisällään funktiot, jotka luovat palvelimen ja asettavat ohjelman kuuntelemaan viestejä tuon palvelimen kautta. Jotta tämä olisi mahdollista, tulee koodin sisältää palvelimen verkkotunnus ja portti. Ilman tätä palvelimen ulkoiset ohjelmat eivät voi lähettää pyyntöjä palvelimelle. Kuvassa 3 näkyvä esimerkki käyttää verkkotunnuksenaan localhostia. (Sufiyan 2024.) Työprojektissa taustajärjestelmää ajetaan NodeJs ajoympäristössä.

```
const express = require('express' 4.18.2 )
const app = express()
const port = 3000

app.get('/', (req, res) => {
  res.send('Hello World!')
})

app.listen(port, () => {
  console.log(`Example app listening on port ${port}`)
})
```

Kuva 3. Yksinkertainen palvelinohjelma Express-kehyksellä (Gascon & Seip 2024)

Työprojektin jakamiseen ja versiohallintaan käytettiin Gittiä. Tämän versiohallintatyökalun voi asentaa kahdella tavalla. Joko voi hyödyntää jotain lukuisista pakettienhallintaohjelmista kuten NPM (Node Package Manager), tai ladata lähdekoodin Gitin verkkosivuilta ja kääntää se manuaalisesti koneen käyttöön. (Chacon & Straub 2024, 18-19.) Koska NPM-systeemi oli jo ladattu työkoneelle, Git ladattiin sen avulla.

Ohjeet henkilökohtaisen tietokannan pystytykseen saatiin muiden työprojektiin liittyvien ohjeiden mukana. Työprojektin tietokantana käytettiin MongoDB-palvelua. Näiden työvaiheiden jälkeen voitaisiin aloittaa uuden projektin kehittäminen.

2.2 Työprojektin tuominen paikalliseen kansioon

Projektin kanssa työskentelyyn liityttiin, kun projekti oli jo käynnissä. Projektin luomiseen ei siis osallistuttu. Keskeneräinen projekti kloonattiin työkoneelle, jotta sitä voitaisiin lähteä

työstämään. Kloonaamiseen käytettiin GitHubin kloonaustyökalua (Chacon & Straub 2024, 52).

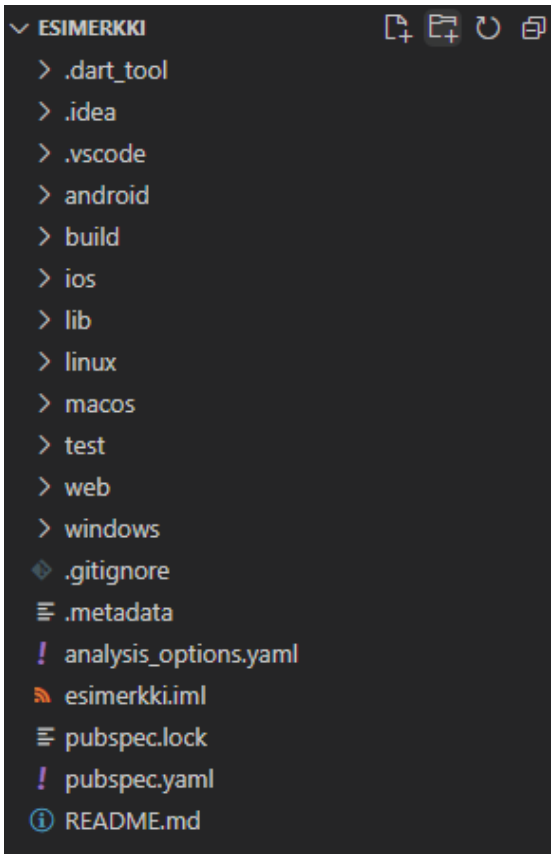
Koska ainoa GitHub-käyttäjä oli tehty oppilaitoksen tunnuksilla, oli syytä tehdä henkilökohtainen käyttäjä työtehtäviä varten. Gitissä on mahdollista luoda tunnisteita käyttäjille työkoneen tiedostoihin, jotta ulkoisten versioiden kanssa työskentely olisi mahdollisimman saumatonta. Tavallisesti puskiessa dataa ulkoiseen versioon tulee käyttäjän syöttää käskyynsä käyttäjänsä nimi ja salasana. Tämä on pidemmän päälle vaivanloista ja aikaa vievää, joten tähän tarvittiin parempi järjestelmä.

Koska Git-tunnukset olivat oppilaitoksen käyttäjän tunnukset, niitä ei voitu hyödyntää työtehtävässä. Niiden vaihtaminen on mahdollista, mutta sitä ei haluttu tehdä, koska useimmat Gitin seuraamat tiedostot työkoneella ovat yhteydessä oppilaitoksen käyttäjään. Tämän takia päätettiin luoda erillinen avainpari, jonka avulla voitaisiin tunnistautua, kun työprojektiin tehtyjä muutoksia halutaan puskea ulkoiseen versioon.

Mainituilla avainpareilla tarkoitetaan SSH-tunnisteita, joiden avulla luotetut käyttäjät voivat tunnistautua puskiessaan dataa Gitin palvelun kautta. SSH-avaimet korvaavat käyttäjän henkilökohtaiset tiedot tunnistautumisen aikana. SSH-avainpariin kuuluu julkinen pari ja yksityinen pari. Julkinen pari asetetaan GitHub-käyttäjään ja yksityinen asetetaan työkoneelle. Tällöin tuolta koneelta voidaan lähettää pyyntöjä julkisella avaimella varustettuun GitHub-käyttäjään. Avainten käyttämiseen voidaan myös lisätä muita turvatoimia kuten salasana. (GitHub Inc. a.)

2.3 Projektin tiedostorakenne

Flutter create-komennolla luodussa projektissa on useita eri kansioita. Suurin osa niistä pitävät sisällään tiedostoja ohjelman ajamiseen ja luomiseen eri alustoilla. Nämä kansiot ovat nimeltään android, ios, linux, macos, web ja windows. (Feizollah 2023.) Tiedostorakenne näkyy kuvassa (Kuva 4).



Kuva 4. Esimerkkiohjelman tiedostorakenne

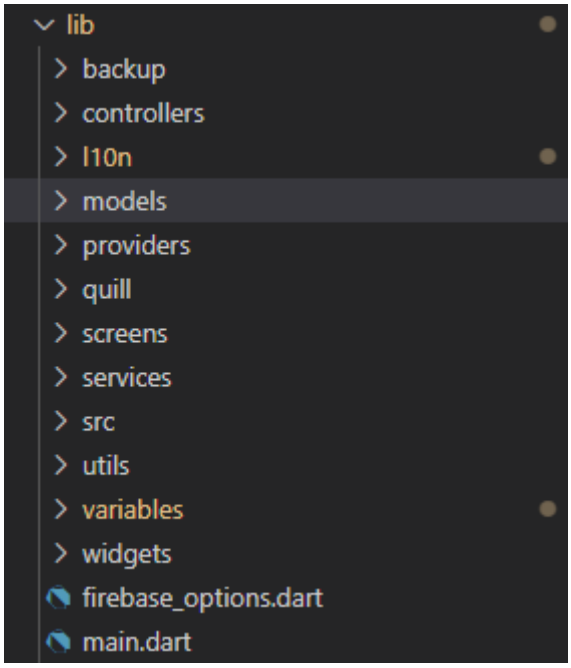
Ohjelmoinnin kannalta tärkein kansio on lib-kansio (Kuva 4). Tämä kansio sisältää suurimman osan varsinaisesta Dart-koodista, jolla projektin käyttöliittymä luodaan. Se myös sisältää sovelluksen aloitustiedoston main.dart. Tästä tiedostosta sovelluksen ajaminen alkaa. (Feizollah 2023.) Muut keskeiset tiedostot ja kansiot projektissa ovat seuraavat:

- Test-kansio pitää sisällään tiedostoja, joiden avulla voidaan kirjoittaa testejä sovellukselle.
- Build-kansio sisältää eri resurssitiedostoja kuten kuvia, sekä eri käyttöjärjestelmissä ajettavat sovelluksen versiot.
- Pubspec.yaml-tiedosto pitää sisällään tietoa sovelluksesta, kuten sovelluksen nimi, kuvaus, versionumero ja sovelluksen käytössä olevat kirjastot ja resurssit
- Analysis_options.yaml-tiedosto rajaa mitä asioita analyysiohjelma tarkkailee sovelluksen ohjelmoinnin aikana. (Hrácek 2023.)

Työprojektissa on kaikki yllä mainitut kansiot ja tiedostot, mutta myös useita muita tiedostoja ja kansioita, joita on luotu projektin edetessä. Useimmat näistä sijoittuvat lib-kansioon.

Työprojektin lib-kansion asettelu seuraa arkkitehtuuria, jossa eri osiot projektista on jaettu omiin alakansioihinsa. Alakansiot ovat nähtävissä kuvassa (Kuva 5). Näiden kansioiden tarkoitus on jakaa projektin eri osat ymmärrettäviksi kokonaisuuksiksi. Kansioita ovat muun muassa seuraavat:

- Screens-kansio sisältää lähinnä tiedostoja, jotka luovat ohjelman eri ikkunat eli suurimmat osat.
- Models-kansio pitää sisällään useimmat oliot, joita projektin koodissa hyödynnetään.
- Providers-kansio on omistettu yksinomaan välittäjäolioille.
- Widgets-kansioon kuuluvat ne komponentit, joita ei voida kategorisoida mihinkään yllä mainituista.



Kuva 5. Työprojektin lib-kansion tiedostorakenne

Työprojektiin kuuluu myös useita muita kansioita, joilla on omat tarkoituksensa, mutta niiden määrän takia jokaista ei käydä tässä tekstissä läpi. Lähes kaikki työhön liittyvät tehtävät vaativat työskentelyä vain yllä kuvattujen kansioiden kanssa.

Flutter-sovelluksen tiedostojen järjestämiselle on useita eri tapoja. Parasta tapaa ei ole, koska jokaisella projektilla on omat tarpeensa. Tärkeämpää onkin, että valittua tiedostoarkkitehtuuria seurataan projektissa. (Bizzotto 2022.) Työprojektin tiedostoarkkitehtuuria voisi verrata Bizzotton kuvaamaan kerrosarkkitehtuuriin.

Kerrosarkkitehtuurissa tiedostot jaetaan kansioihin niiden tarkoituksen pohjalta. Käyttöliittymän luonnista vastuussa olevat tiedostot sijoitetaan yhteen kansioon, välittäjäoliot toiseen ja niin edelleen. Tällöin ohjelman muokkaaja tietää suoraan mistä kansiosta etsiä mitäkin tiedostoa. Kerrosarkkitehtuuri helpottaa myös ohjelman laajentamista. Yksittäisiä uusia tiedostoja voidaan lisätä vapaasti vaarantamatta arkkitehtuurin eheyttä. Tämä tuo mukanaan haasteita sovelluksen kasvaessa. Yhden ominaisuuden toiminnasta saattaa olla vastuussa yksi tiedosto jokaisesta kerroksesta. (Bizzotto 2022.) Tämä ongelma korostui etenkin silloin, kun yksi ominaisuus oli jaettu useampaan tiedostoon. Esimerkiksi yhden alasivun tai kookkaan ponnahdusikkunan logiikkaa saatettiin jakaa useampiin tiedostoihin. Tämä hidasti työtä useampaan otteeseen.

Tätä ongelmaa yritettiin joissain paikoin lieventää luomalla eri ominaisuuksille omia alakansioitaan. Esimerkiksi kalenterisivun kaikkia komponentteja ei asetettu widget-kansioon vaan niille tehtiin omia alakansioitaan screens-kansioon. Tämä lähestymistapa lainaa piirteitä ominaisuuslähtöisestä arkkitehtuurista, jossa jokainen ominaisuuden osa säilötään yhteen kansioon (Bizzotto 2022). Tätä taktiikkaa käytettiin vain käyttöliittymästä vastuussa olevien tiedostojen organisoinnissa. Erinäisiä alakansioita luotiin screens- ja widgets-kansioiden sisälle, mutta esimerkiksi providers-kansion sisältöä ei sekoitettu kummankaan edellä mainitun kansion sisällön kanssa.

Tämä lähestymistapa helpotti joissain tapauksissa ominaisuuksien muokkaamista, mutta nosti esiin uuden ongelman projektin arkkitehtuurin yhdenmukaisuudessa. Esimerkiksi screens-kansiossa tulisi olla vain sovelluksen kokonaisia sivuja luovaa logiikkaa. Tämän lähestymistavan vuoksi screens-kansioon tehtiin alakansioita, joissa oli widget-kansioon kuuluvaa materiaalia. Tästä syystä suurempien ominaisuuskokonaisuuksien kohdalla saattoi ilmetä, että kahdessa ns. pääkansiossa oli oma alakansionsa, jotka sisälsivät saman ominaisuuskokonaisuuden logiikkaa. Tämän ongelman muodostuminen on huomattavasti vaikeampaa puhtaassa kerrosarkkitehtuurissa. Jokaisella uudella ominaisuudella on oma paikkansa ja täten kansiorakenteen ristiriitoja tai päällekkäisyyksiä syntyy harvemmin. Tästä huomatiin, miten tärkeää on valita sopiva arkkitehtuuri projektille, tutustua tarkasti projektin arkkitehtuuriin ja seurata valittua arkkitehtuuria mahdollisimman tarkasti.

3 Käyttöliittymän toteutus

3.1 Flutter

Flutter on Dart-ohjelmointikieleen perustuva avoimen lähdekoodin käyttöliittymäkehys, joka on luotu mobiili- ja verkkoapplikaatioiden kehittämistä varten. Sillä on sisäänkirjoitettuna laaja kirjasto valmiita komponentteja ja se on käyttökelpoinen yleisimmissä mobiilikäyttöjärjestelmissä ja Googlen verkkoselaimessa. (Windmill 2020, 3-4.)

Flutter sisältää tarvittavat työkalut toimivat käyttöliittymäsovelluksen luomiseen. Flutterissa on valmiita käyttöliittymän osia, renderöintimoottori, testikehys, reititin ym. (Windmill 2020, 3-4.)

3.1.1 Komponentit ja tilat

Flutterin toiminnallisuudet painottuvat komponentteihin (widget). Nämä ovat Dart-olioita, joiden on määrä suorittaa jokin yksittäinen tehtävä. Esimerkiksi tekstikomponentti ottaa vastaan merkkijonon ja esittää sen käyttöliittymässä, kun taas säiliökomponentti ottaa vastaan toisen komponentin ja määrittää sille korkeimmat mahdolliset ulottuvuudet. Flutterissa käyttöliittymä luodaan komponenttien avulla. Niitä yhdistelemällä voidaan luoda monimutkaisia kokonaisuuksia. (Windmill 2020, 12-13.)

Toinen Flutterin keskeisimmistä ominaisuuksista on tilat. Tiloilla tarkoitetaan ohjelman muuttujia, jotka muuttuessaan vaikuttavat siihen, miltä ohjelma näyttää käyttöliittymässä. Esimerkiksi yksi työprojektin ohjelma luo listaan komponentteja oliolistan pohjalta. Jos listasta poistetaan yksi olio, tulee tilan muuttua, jotta käyttöliittymä heijastaa tämän listan muutoksen. Esimerkki tilan muutoksesta on kuvassa (Kuva 6). Tilojen avulla saadaan käyttöliittymä reagoimaan käyttäjän toimintoihin. (Windmill 2020, 16.)

```
IconButton(
  onPressed: () async {
    bool? temp = await confirmationDialog(context);
    if(temp != null && temp) {
      //Tilan muutos
      setState(() {
        service.comments[program.id]!.removeAt(i);
      });
      await provider.updateService(service);
    }
  },
  icon: const Icon(Icons.delete, color: Colors.red)
) // IconButton
```

Kuva 6. Tilaa muuttaja funktio

3.1.2 Käyttöliittymän päivittämisen perusteet

Kun komponenteista luodaan käyttöliittymää, se alkaa yhdestä komponentista. Tämän komponentin sisälle, eli lapsiksi asetetaan lisää komponentteja, kunnes ne muodostavat laajenevan monihaaraisen rakenteen. Tätä kutsutaan komponenttipuuksi. (Windmill 2020, 18-20.)

Aina kun tila muuttuu tällaisessa puussa, Flutter luo sen alusta asti uudelleen aloittaen ensimmäisestä komponentista. Kun vastassa on komponentti, johon tilan muutos vaikuttaa, Flutterin järjestelmä luo sen uudelleen muutoksen mukaan ja jatkaa puuta eteenpäin. Tällä tavalla vain ne osat puuta, jotka kaipaavat muutosta saavat niin sanotusti tarvitsemansa huomion. Muut ovat kopioita vanhasta puusta. Tällä tavalla Flutter onnistuu luomaan uudestaan suuria kokonaisuuksia nopeasti. (Windmill 2020, 18-20.)

Yksi Flutterin käyttöliittymän asettelun keskeisimmistä periaatteista on ulottuvuushierarkia. Komponentit seuraavat hierarkiassa ylempien eli äitikomponenttien ulottuvuuksia ja jos lapsikomponentin ulottuvuudet asetettaan äitikomponentin ulottuvuuksien yli, tämä joko johtaa lapsikomponentin ulottuvuusarvojen mitätöimiseen tai virheviestiin. Ulottuvuuksien mitätöiminen johtaa lapsikomponentin kasvattamiseen niin suureksi kuin mahdollista. (Windmill 2020, 75.) Ennen kuin tämä periaate ymmärrettiin kokonaisuudessaan, käyttöliittymän asettelussa aiheutettiin useita virheitä. Näitä virheitä korjattiin menetelmillä, jotka lisäsivät turhaa koodia. Vasta kun lapsi- ja äitikomponenttien suhde ymmärrettiin täysin, saatettiin säästää koodirivejä ja suoraviivaistaa käyttöliittymän asettelua.

Komponentit voidaan jakaa kahteen ryhmään tilasidonnaiset- ja tilariippumattomat komponentit. Tilasidonnaiset komponentit kykenevät muuttumaan sen mukaan, miten niiden tilat muuttuvat. Näiden avulla tehdään suurin osa interaktiivisista käyttöliittymistä. Tilariippumattomat komponentit taas eivät ole sidonnaisia tilan muutoksiin. Nämä ovat hyödyllisiä niissä tilanteissa, kun ollaan varmoja, että komponentin muuttujia ei tarvita sen ulkopuolella, tai ulkoisten muuttujien ei tarvitse vaikuttaa sen ulkomuotoon. (Windmill 2020, 16.)

Työprojektissa käytettiin lähes poikkeuksetta tilariippuvaisia komponentteja. Tämä johtui osin siitä, ettei tilariippumattomien komponenttien etuja sisäistetty kokonaan, mutta suurempi syy oli, että useimmat työtehtävät vaativat interaktiivisia käyttöliittymän osia. Yleisimpänä esimerkkinä on ponnahdusikkuna, joka sisältää lomakkeen jonkin olion luomiseksi. Tämä olio saattoi olla käyttäjä, resurssi tai resurssin huolto. Tällaisen lomakkeen luominen vaati useimmiten useita tekstikenttiä, pudotusvalikoita ja valintaruutuja. Tällaisen ponnahdusikkunan on oltava väistämättä tilariippuvainen. Sama ei päde ponnahdusikkunoihin,

joiden tarkoitus on esittää tietoa halutusta objektista. Tällaiset olivat myös yleisiä, kun haluttiin esittää suuria määriä olioita käyttöliittymän listassa, mutta noiden olioiden sisältämää tietoa ei saatu järkevästi aseteltua tuohon listaan. Listan objektia painamalla avattiin tarkemmat tiedot sisältävä ponnahdusikkuna. Koska tällaisen ponnahdusikkunan esittämää tietoa ei ollut tarkoitus muokata, se ei toimiakseen vaatinut juuri koskaan tiloja. Tästä huolimatta osa näistä ponnahdusikkunoista kirjoitettiin tilariippuvaisiksi, vaikka se ei olisi ollut tarpeellista.

3.1.3 Käyttöliittymän yhdistäminen tietokantaan

Työprojekti hyödyntää http-kirjastoa Dart-ohjelmointikielelle. Tämä kirjasto mahdollistaa http-kutsujen lähettämisen Dart-sovelluksen kautta. Kirjasto antaa työkalut luoda CRUD-pyyntöjä. Http-kutsufunktio tällä kirjastolla vaatii toimiakseen kaksi muuttujaa, osoitemuuttujan ja sisältömuuttujan. Yksinkertainen http-kutsu esitetään kuvassa (Kuva 7). Osoite määrittää, mihin verkko-osoitteeseen pyyntö lähetetään ja sisältömuuttuja sisältää pyyntöön liittyvän datan. Tämä pyyntöfunktio palauttaa pyynnön vastauksen. (Quinlan 2024.) Tämän vastauksen voi tallettaa muuttujaan, jonka avulla vastauksen dataa voidaan käsitellä.

```dart
static Future<void> simplePost(dynamic dataToPost) async {
  try {
    //Luo osoitemuuttujan
    Uri requestUrl = Uri.parse(_productsUrl);

    //Lähettää pyynnön ja palauttaa vastauksen
    var response = await http.post(requestUrl,
        //Sisältömuuttuja
        body: jsonEncode(dataToPost.toJson()),
    );

    //Esittää vastauksen konsolissa
    logger.d(response);
  } catch (error) {
    logger.e('Error: $error');
    rethrow;
  }
}
```

Kuva 7. Esimerkki http-kirjastolla luodusta kutsufunktiosta

Eri http-kutsuja käsittelevät funktiot keskitettiin työprojektissa yhden tiedoston yhteen luokkaan nimeltä ApiService. Tämän luokan avulla http-kutsujen logiikka levitetään ohjelman käyttöön. Tällainen koodin keskittäminen tekee koodin käyttöönotosta ja uusien

taustajärjestelmän ominaisuuksien lisäämisestä suoraviivaisempaa, mutta se johti siihen, että ApiServicen tiedosto kasvoi nopeasti. Tämä teki virheiden korjaamisesta ja muusta koodin kanssa työskentelemisestä työlästä, kun oikeaa funktiota etsittiin lukuisten samantapaisten funktioiden joukosta. Joissain tapauksissa olisi voinut olla hyödyllistä jakaa tuon tiedoston sisältöä useampiin tiedostoihin. Tämä toisaalta johtaisi yhden suuren tiedoston sijasta kymmeniin pienempiin tiedostoihin, joista oikean etsiminen olisi yhtä lailla työlästä.

3.2 Flutterin komponentit

Flutter sisältää suuren määrän valmiita komponentteja. Suurin osa niistä keskittyy käyttöliittymän visualisointiin tai interaktiivisuuden mahdollistamiseen. Usein sanotaan, että kaikki Flutterissa on komponentteja ja lähes kaikki Flutter-sovelluksen osat ovatkin komponentteja. (Windmill 2020, 60.)

Näistä yksinkertaisemmista komponenteista pyritään Flutter-ohjelmoinnissa rakentamaan kokonaisuuksia, jotka suorittavat jonkin yhden tehtävän sovelluksessa. Tämä voi olla hakuikkuna käyttäjän yhteystiedoille tai ponnahdusikkuna, joka välittää tietoa käyttäjälle tai pyytää käyttäjältä toimintoa. Näistä suuremmista komponenttikokonaisuuksista varsinainen Flutter-sovellus rakennetaan.

3.2.1 Navigointi

Navigointi perustuu Flutterissa polkuihin. Nämä ovat ainutlaatuisia merkkijonoja, joiden avulla haluttu komponentti voidaan tunnistaa ja tavoittaa. Polkuja hyödynnetään navigaattorikomponentin avulla. Tämä komponentti sisältää navigaatiota hallitsevia funktioita. (Windmill 2020, 195-196.) Ainoat navigaatiokomponentit, joiden kanssa työskenneltiin työprojektin aikana, olivat pushNamed ja pop. Esimerkki pushNamed-funktiosta esitetään kuvassa (Kuva 8). PushNamed ottaa parametrikseen polun ja vaihtoehtoisesti listan argumentteja ja rakentaa niiden pohjalta polulla osoitetun komponentin. Pop siirtyy niin sanotusti navigaatiohistoriassa yhden askeleen taaksepäin ja kasaa käyttöliittymän edellisen sivun pohjalta. Tätä funktiota on myös mahdollista hyödyntää esimerkiksi ponnahdusikkunoiden sulkemiseen. (Windmill 2020, 195-196.)

```
Navigator.of(context).pushNamed(
  CompanyProfilePage.routeName,
  arguments: {
    'company': chosenCompany ?? companies.first,
    'pageView': "settings"
  },
);
```

Kuva 8. PushNamed-funktio

Useissa kohdissa työprojektia haluttiin navigoida sivulle, joka vaati toimiakseen vähintään yhden muuttujan arvon. Tuon syöttämiseen voidaan käyttää navigaatiokomponentin navigaatiofunktion arguments-parametriä (Kuva 8). Tämä parametri on Map-olio, jonka avain on merkkijono ja arvo on dynaaminen muuttuja. Nämä arvot purkava logiikka kirjoitetaan samalla, kun polku määritellään. Tälle on muutamia funktioita. Työprojektissa hyödynnettiin ModalRoute.of()-funktiota, joka palauttaa navigaatio-ohjelmalle syötetyt argumentit Map-oliona. Yksi työprojektin polun määrittely esitetään kuvassa (Kuva 9). Tästä oliosta ne voidaan lisätä polun määrittelyn yhteydessä kutsuttavaan komponenttiin. (Windmill 2020, 203.)

Tämä mahdollisti navigoinnin muuntamisen sivun tilan hallinnaksi. Erään työprojektin sivu on jaettu useaksi alasivuksi. Näillä alasivuilla ei ole omaa polkua. Ne luodaan sen mukaan minkä alasivun tunniste on asetettu pääsivun senhetkiseksi tilaksi. Tämä tunniste olisi kuvassa 9 nimeltään pageView. Tällä tavalla vältyttiin kymmenien ylimääräisten polkujen lisäämiseltä. Tavallisesti kun pääsivulle siirrytään, pageView-muuttujaa ei ole määritelty. Tällöin pääsivu avaa ensimmäisen alasivunsa käyttäjälle. Käyttäjä voi tämän jälkeen navigoida haluamalleen alasivuille pääsivun oman logiikan avulla. Haluttiin kuitenkin antaa käyttäjälle mahdollisuus siirtyä suoraan alasivulle, ilman että sinne siirrytään pääsivun ensimmäisen alasivun kautta. Jotta tämän pääsivun alasivuille voitaisiin navigoida pääsivun ulkopuolelta, on pääsivun polun navigaatioon lisättävä argumentti, johon voi asettaa halutun alasivun tunnisteen. Kun tämä tunniste syötetään pääsivulle, sen tila muuttuu ja sivu näyttää oikean alasivun.

```
CompanyProfilePage.routeName: (ctx) {
  final args =
      ModalRoute.of(ctx)!.settings.arguments as Map<String, dynamic>?;
  return args != null
      ? CompanyProfilePage(
          company: args['company'] as Company,
          pageView: args['pageView'],
        )
      : const ProfilePage();
},
```

Kuva 9. Esimerkki polun määritelmästä

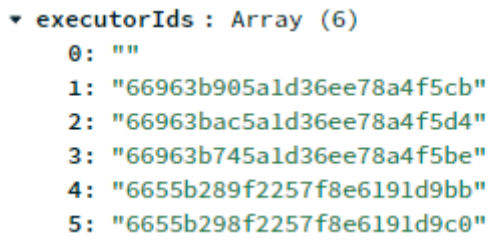
3.2.2 Välittäjät ja kuluttajat

Työprojektissa hyödynnettiin provider-paketin tuomia välittäjä- (provider) ja kuluttajakomponentteja (consumer). Nämä ovat kaksi tilamanipulaation osaa, joiden on tarkoitus helpottaa tilojen käsittelyä komponenttien kanssa. Nämä komponentit toimivat siten, että välittäjä

ilmoittaa datassa tapahtuneesta muutoksesta, johon kuluttaja reagoi muuttamalla tilaansa. Tähän liittyy myös kolmas komponentti, joka on muutoksen ilmoituksen välittäjä. Tätä komponenttia käytetään määrittämään välittäjät ennen niiden käyttöä. (Lougheed 2024b.)

Työprojektissa jokaisella tietokantaan tallennettavalla oliolla on oma välittäjänsä. Tämä välittäjä sisältää funktioita, jotka hakevat dataa tietokannasta ja muuttujia, jotka pitävät tuota dataa sisällään ohjelman ollessa käynnissä. Näiden välittäjien avulla mikä tahansa komponentti voi päästä käsiksi tietokannan dataan kutsumalla oikean välittäjän käyttöönsä. Kuluttajan avulla komponentti voi ikään kuin asettua kuuntelemaan välittäjässä tapahtuvia muutoksia. Kuluttaja pitää sisällään rakentajaa, joka rakentaa ja palauttaa komponentin tilamuutoksen mukaan. Kuluttajalle on erikseen ilmoitettava, kuinka montaa ja mitä välittäjiä sen on kuunneltava. (Lougheed 2024b.)

Koska välittäjät toimivat myös porttina tietokannan dataan, tarvittiin niitä muissakin konteksteissa kuin tilamanipulaatiossa. Monissa kohtaa työprojektia eri oliot pitävät sisällään toisten olioiden tunnisteita. Esimerkki tällaisesta oliosta tietokannan objektina esitetään kuvassa (Kuva 10). Nämä ovat merkkijonoja, joilla objekti tunnistetaan tietokannassa. Tällä tavalla luotiin erilaisia olioryhmiä tai olioita, jotka voivat näennäisesti pitää sisällään toisia olioita. Kun olion sisällä olevia olioita tarvittiin, ne haettiin niitä käsittelevän välittäjän listoilta tunnisteen avulla. Jos olion sisäinen olio poistetaan, tämä ei kuitenkaan poista tuon olion tunnistetta toisesta oliosta. Pitemmän ajan päästä tämä johtaa suureen määrään tyhjiä tunnisteita tietokannassa.



Kuva 10. Esimerkki huolto-olion tekijälistasta tietokannassa

Tällaisessa tilanteessa Provider.of-funktiosta on hyötyä. Tämän funktio kutsuu välittäjää ja antaa pääsyn välittäjän dataan ja funktioihin. Kun olion sisäinen olio poistetaan, voidaan poistettavan olion tunnisteet poistaa muista olioista hakemalla ne välittäjiensä listoista. Tämän Provider.of-funktio mahdollistaa. Provider.of-funkiton käyttökohde esitetty kuvassa (Kuva 11). (Lougheed 2024b.)

```
allContacts = await Provider.of<ContactProvider>(context, listen: false)
    .fetchContacts();
```

Kuva 11. Esimerkki Provider.of-funktion käytöstä

3.2.3 Teemat

Teemakomponentti pitää sisällään väri- ja fonttimuuttujia, joiden tarkoitus on suoraviivaistaa ja yhdenmukaistaa sovelluksen ulkoasun luomista. Oletusarvoisesti kukin komponentti käyttää sitä teemaa, joka on ensimmäisenä määritelty sen ylemmissä komponenteissa. (Windmill 2020, 108.)

Työprojektissa useimmat teemat käsittelivät väriä valoisassa tai tummassa skaalassa sekä eri konteksteissa käytettäviä fontteja. Joillekin komponenteille tehtiin yksityiskohtaisempia teemoja, kuten tekstinappuloille tai muille painikkeille. Yleiset väriskaalat asetettiin heti niin sanotusti ohjelman juureen MaterialApp-komponenttiin. Tällä komponentilla on teemoille tarkoitettuja muuttujia, joista jokainen ohjelman komponentti hakee teemadatansa lähtökohtaisesti (Windmill 2020, 109).

3.2.4 Asettelu

Työprojektin asettelussa hyödynnetään pitkälti Flutterin sisäänrakennettuja komponentteja. Yleisin sivujen pohja on Scaffold-komponentti, joka on tarkoitettu helpottamaan sivujen luontia. Sen rakenteeseen kuuluu yläpalkki, alapalkki, näiden väliin jäävä sivun sisältö ja avattava valikko. Scaffold-komponenttiin kuuluu myös muita parametrejä yksityiskohtaisemman ulkoasun luomiselle. Tämä komponentti toi yhdenmukaisuutta käyttöliittymän arkkitehtuurille ja helpotti eri ohjelmoijien koodin ymmärtämistä, kun koodin pohjalla oli lähes aina sama runko. Kaikki Scaffold-komponentin parametrit ovat vapaaehtoisia käyttää. Työprojektissa ainoat osat, joita käytettiin jokaisessa sen ilmentymässä, olivat yläpalkki ja sisältö. (Windmill 2020, 104-105.) Osa Scaffold-komponentista on esitetty kuvassa (Kuva 12).

```
return Scaffold(
  appBar: AppBar(
    title: const Text('About'),
    leading: IconButton(
      icon: const Icon(Icons.arrow_back),
      onPressed: () {
        Navigator.pop(context);
      },
    ), // IconButton
  ), // AppBar
  body: SingleChildScrollView(
```

Kuva 12. Scaffold-komponentti

Työprojektin sovelluksen on tarkoitus toimia monenlaisilla käyttöjärjestelmillä ja monen kokoisilla ruuduilla. Käytännössä tämä tarkoittaa sitä, että sovelluksen asettelun tulee olla

responsiivinen leveillä ja kapeilla ruuduilla. Tähän tarkoitukseen on Flutterissa olemassa MediaQuery-komponentti. Esimerkki MediaQuery-komponentin käytöstä on esitetty kuvassa (Kuva 13). Tämän komponentin avulla voidaan laskea käytössä olevan ruudun leveys pikseleinä ja tuon muuttujan avulla laskea komponentin halutut ulottuvuudet. Esimerkiksi, jos halutaan, että tekstinsyöttölomake on leveällä näytöllä viidennes ruudun leveydestä, mutta kapeammalla näytöllä lähes koko ruudun levyinen, on tämä MediaQuery-komponentilla mahdollista. (Windmill 2020, 110.) Jotta käyttöliittymän ulottuvuudet pysyisivät joka ruudulla halutun mittaisina, lähes kaikkien leveys- ja korkeusarvojen määrittämiseen käytetään työprojektissa tätä komponenttia.

Joissain tapauksissa pelkän leveyden muuttaminen ei riittänyt. Yksi yleisimmistä esimerkeistä on erinäiset joukot komponentteja tai painikkeita, jotka asetetaan riviin käyttöliittymässä. Usein kapeammilla näytöillä ei ole tarpeeksi tilaa esittää näitä komponentteja rivissä, joten ne on asetettava jonoon. Tätä varten luotiin funktio, joka ottaa vastaan listan komponentteja ja asettaa tuon listan riviin, jos näyttö ei ole kapea ja jonoon jos näyttö on kapea.

```
//Checks if screen is narrow
bool narrowScreen(BuildContext context) {
  return MediaQuery.of(context).size.width < MediaQuery.of(context).size.height;
}
```

Kuva 13. Työprojektin yleisimpiä MediaQueryn käyttökohteita

3.2.5 Listat

Yksi yleisimmistä tarpeista työprojektissa on esittää oliolistan sisältö käyttöliittymässä. Näitä olioita on kaikenlaisia huolloista kommentteihin, mutta periaate on niiden esittämisessä sama. Koko lista on esitettävä, oli listan pituus mikä hyvänsä.

Yksi yleisimmistä ratkaisuista tähän on Flutterin ListView. ListView-komponentin implementointi esitetty kuvassa (Kuva 14). Se ottaa parametrina vastaan numeron ja luo tuon numeron mukaan komponentteja käyttöliittymään pystysuuntaisena listana. Komponentti on verrattavissa for-silmukkaan toimintaperiaatteensa puolesta. Komponentin ominaisuuksiin kuuluu mahdollisuus vierittää listaa ylhäältä alas, jolloin listan pituudella ei ole esteitä. (Windmill 2020, 126-127.)

```
child: ListView.builder(
  shrinkWrap: true,
  //Listan objektien lukumäärä
  itemCount: filteredUsers.length,
  //Listan objektin luonnin logiikka
  itemBuilder: (context, index) {
    CompanyUser user = filteredUsers[index];
    return ListTile(
      title: Text(user.name),
      onTap: () {
        setState(() {
          personController.text = user.id!;
        });
        Navigator.of(context).pop();
      },
    ); // ListTile
  },
), // ListView.builder
```

Kuva 14. ListView-komponentti

Flutterissa on myös muita vieritettävän näkymän mahdollistavia komponentteja. Toinen yleinen komponentti työprojektissa on SingleChildScrollView-komponentti. SingleChildScrollView on esitetty kuvassa (Kuva 15). Tämä komponentti ottaa lapsekseen toisen komponentin ja tekee siitä vieritettävän (Hickson 2024). Tätä käytetään silloin kun rullattava kohde ei ole lista tai, kun listan asettelussa on joitain vaatimuksia, jotka eivät ole saavutettavissa ListView-komponentilla.

```
body: SingleChildScrollView(
  padding: const EdgeInsets.all(16.0),
  child: Form(
    key: _formKey,
    child: buildForm(context),
  ), // Form
), // SingleChildScrollView
```

Kuva 15. SingleChildScrollView

3.2.6 Dialogit

Dialogit ovat Flutterissa ponnahdusikkunoita. Vaikka dialogin voi luoda niin sanotusti puhtaalta pöydältä, löytyy tälle Flutterista valmiita pohjia. (Austin ym. 2024a.) Työprojektissa käytetyin pohja on AlertDialog-komponentti. Tämä on esitetty kuvassa (Kuva 16). Tätä suosittiin sen tarjoaman asettelupohjan takia.

AlertDialog on rakenteeltaan samankaltainen kuin edellä mainittu Scaffold. Sillä on otsikko ylimpänä, toimintavalikko alimpana ja sisältö näiden kahden välissä. Vaikka tämän komponentin alkuperäinen tarkoitus on ilmoittaa käyttäjälle jostain odottamattomasta ja kysyä mitä käyttäjä haluaa tehdä, käytetään tätä työprojektissa lähes jokaisen ponnahdusikkunan pohjana. (Austin ym. 2024b.)

```
return showDialog<bool>(
  context: context,
  builder: (BuildContext context) {
    return AlertDialog(
      //otsikko
      title: Center(child: Text(AppLocalizations.of(context)!.areYouSure,
      //toimintavalikko
      actions: <Widget>[
        Row(
          mainAxisAlignment: MainAxisAlignment.spaceEvenly,
          children: [
            TextButton(
              style: TextButton.styleFrom(
                textStyle: Theme.of(context).textTheme.labelLarge,
              ),
              child: Text(AppLocalizations.of(context)!.yes),
              onPressed: () {
                Navigator.of(context).pop(true);
              },
            ), // TextButton
```

Kuva 16. AlertDialog

Dialogi saadaan esiin showDialog-komponentin avulla (Kuva 16). Tämä komponentti lähtökohtaisesti palauttaa halutun dialogin. Sen avulla voidaan myös määrittää, jos dialogin halutaan palauttavan jokin arvo. (Austin ym. 2024a.) Esimerkkinä työprojektissa melko laajasti käytetty varmistusdialogi, joka ilmestyy, kun käyttäjä on tekemässä jotain peruuttamatonta. Kuvankaappaus tästä ponnahdusikkunasta on kuvassa (Kuva 17). Tämä dialogi kysyy käyttäjältä, onko tämä varma, että haluaa suorittaa aiotun toiminnon ja palauttaa totuusarvomuuttujan riippuen käyttäjän vastauksesta. Tämän totuusarvon mukaan voidaan joko ajaa aiottu toiminto loppuun tai jättää se tekemättä.

Kuva 17. Varmistusdialogi

3.3 Dart

Dart on oliopohjainen ohjelmointikieli, joka on luotu käyttäjäsovellusten ohjelmoimiseen. Dart on niin sanottu korkeamman tason ohjelmointikieli. Sillä on sisään rakennettuna laajat kirjastot, joissa on paljon funktioita ja luokkia. (Belanger ym. 2024.)

Dartiin on rakennettu tyhjän muuttujan estojärjestelmä. Tyhjän muuttujan estojärjestelmä tarkoittaa, että ohjelma ei aja koodia, joka ei ota huomioon tyhjien muuttujien mahdollisuutta. Dartissa ei oletuksena ole mahdollista asettaa tyhjää tietoa muuttujaan, jota ei ole erikseen luokiteltu mahdollisesti tyhjäksi. Tämän voi kiertää erilaisilla operaattoreilla tai if-lauseilla. Tämän turvajärjestelmän voi myös asettaa halutessaan pois päältä. (Belanger ym. 2024.)

Tämä ominaisuus on ollut yksi hyödyllisimmistä, mutta myös aikaa vievimmistä asioista Dartilla ohjelmoidessa. Useimmissa tapauksissa on hyvä, jos kääntäjä ilmaisee tyhjän muuttujan mahdollisuuden. Työprojektissa käytetään tyhjiä muuttujia melko aktiivisesti. Esimerkiksi jos funktiossa halutaan olevan mahdollisuus, ettei se palauta mitään, voidaan sen sisälle kirjoittaa logiikkaa, joka palauttaa tarvittaessa tyhjän muuttujan. Esimerkiksi työprojektin olioiden luonti-ikkunoissa oli oltava mahdollisuus sille, että olion luominen keskeytetään. Tällöin tuo ikkuna asetettiin palauttamaan tyhjä muuttuja. Tästä syystä tyhjille muuttujille tuli kirjoittaa ne huomioon ottavaa logiikkaa hyvin tiheään. Joissain tapauksissa lyhyt rivi saattaa paisua, kun tavanomaiseen logiikkaan liitetään komentoja sille, miten toimitaan tyhjän muuttujan kohdalla. Joka tapauksessa tästä ominaisuudesta koettiin olevan enemmän hyötyä kuin haittaa, joten sitä ei poistettu missään vaiheessa käytöstä.

Toinen tämän kaltainen ominaisuus on Dartin tyypitys. Tyypitys tarkoittaa, että jokaisella muuttujalla tulee olla tyyppi ja tuo muuttuja ei voi pitää sisällään kuin oman tyyppistään dataa. Se vähentää virheiden määrää huomattavasti, kun virheet ilmoitetaan jo kääntäjässä. (Windmill 2020, 31.) Tämä tarkoittaa myös sitä, että koodista voi tulla

vaikealukuisempaa. Esimerkiksi, jos funktiolla täytyy olla useampi parametri, jokainen parametri tulee ilmoittaa tyypin kanssa. Varsinkin silloin, jos tyypitykset ovat sanoina pitkiä, voi tämä johtaa siihen, että yhdellä funktiolla on useampi rivi pitkästi kirjoitettuja parametrejä. Tätä ongelmaa voisi teoriassa lieventää lyhentämällä muuttujien nimiä tai luotujen olioiden nimiä. Työprojektissa tämä ongelma ilmeni usein silloin, kun jokin funktio vaati usean välittäjäolion parametrikseen. Välittäjäolioiden nimet jouduttiin usein kirjoittamaan pitkiksi selkeyden vuoksi. Yksi esimerkki näistä on ServiceProgramProvider. Tällaisilla pitkillä luokkanimillä tyypitettyjä muuttujia jouduttiin useampaan otteeseen asettamaan funktioiden parametreiksi, mikä johti kookkaisiin funktion määritelmiin. Esimerkki tällaisesta funktiosta on esitetty kuvassa (Kuva 18).

```
Widget searchFunctionality(
    ServiceProgramProvider provider,
    ResourceProvider resourceProvider,
    ServiceProvider serviceProvider,
    CompanyConProvider coCoProvider,
    CompanyUserProvider coUserProvider,
    int index) {
  ServiceProgram temp = provider.servicePrograms[index];
  bool found = false;
  bool statusMatch = false;
```

Kuva 18. Kookkaita tyypityksiä

Muuttujanimien lyhentäminen korjaisi koodin rönsyilemisongelman. Se tosin aiheuttaisi sen, että muuttujien nimistä olisi vaikea päätellä, mikä muuttujan tarkoitus koodissa on. Esimerkiksi muuttujan nimeltä serviceProgramProvider voisi muuttaa muotoon spp. Tätä kirjainyhdistelmää voi olla kuitenkin vaikea tulkita, jos kirjoittaja ei ole ohjelman kanssa ennestään tuttu tai on työskennellyt koodin kanssa kauan sitten. Tästä syystä työprojektissa useimmiten käytettiin pitempiä muuttujanimiä, jos se oli tarpeen.

Dartissa on mahdollisuus kirjoittaa myös dynaamisia muuttujia. Tälle on oma tyyppinsä. Dynaaminen muuttuja voi ottaa sisäänsä minkä tahansa muun tyyppisen muuttujan dataa. (Windmill 2020, 31.) Dynaamiset muuttujat mahdollistavat niin sanottujen universaalien funktioiden tekemisen. Universaalit funktiot ottavat vastaan dynaamisia muuttujia ja suorittavat niillä jonkin toiminnon. Esimerkkinä työprojektiin lisätty komponentti, joka esittää sille annetun listan olioita käyttöliittymässä. Jotta komponentti voisi esittää mitä olioita tahansa, tuli sen luovan funktion parametriksi asettaa dynaamisten muuttujien lista. Tätä komponenttia hyödynnettiin, kun käyttäjän haluttiin valita yksi olio listasta. Listan oliota painamalla, lista palauttaa tuon olion.

4 Taustajärjestelmän toteutus

4.1 MondoDB

MongoDB on tietokanta, joka tallentaa JSON-dataa. Se on niin sanottu dokumenttipohjainen tietokanta, eli se tallentaa saamansa datan taulun sijasta dokumenteille. Nuo dokumentit kasataan kokoelmiin ja näiden kahden osan avulla muodostuu tietokanta. (Gills 2023.)

MongoDBn isäntäyhtiö tarjoaa MongoDBn sekä verkossa, että avoimena lähdekoodina ladattavaksi käyttäjän koneelle (Gills 2023). Koska opinnäytetyön käytännönosuuden aikana käytettiin MongoDBn verkkoversiota, eli MongoDB Atlasta, tullaan tässä tekstissä keskittymään vain tähän versioon ja kutsumaan sitä tästä eteenpäin vain MongoDBksi.

Dokumenttien sisältöä voidaan verrata Map-olioihin. Dokumentissa on kenttiä, jotka tunnistetaan merkkijonolla ja jotka pitävät sisällään varsinaisen datan. Datan syöttämiseen MongoDB hyödyntää JSON-datan BSON-versiota. BSON on eräänlainen binääriversio JSON:ista. Tätä hyödynnetään, koska BSON tarjoaa laajemman muuttujavalikoiman. (Gills 2023.) Työprojektissa dokumentin kentistä kasataan tallennettavat oliot. Yksi dokumentti pitää sisällään yhden olion ja noista dokumenttijoukoista koostuvat kokoelmat. Kokoelmat pitävät sisällään useita dokumentteja (Gills 2023). Kokoelmat saavat nimensä lähtökohtaisesti tallennettavien olioiden nimestä.

Projektin aikana käytössä oli sekä henkilökohtainen tietokanta että yleinen tietokanta. Yleinen tietokanta on tietokanta, johon projektin jaettu versio yhdistyy. Tästä syystä kenen tahansa työntekijän muutokset tietokannan datassa näkyvät kaikille yleisen version käyttäjille. Jos tehtävänä ei ole muokata tietokantaa, tämä ei ole ongelma ja saattaa joissain tapauksissa säästää aikaa, kun voi käyttää muiden tallentamia olioita omissa testeissään.

Jos taas tehtävänä on muokata tai päivittää tietokannan toimintaa, niin henkilökohtainen tietokanta on lähes välttämätön. Julkista tietokantaa käsittelevää koodia ei voida muokata suoraan vaan tuosta koodista on otettava kopio. Tuo kopio on saatava toimimaan erillisessä tietokannassa, jotta tehtyjen muutosten vaikutuksia voitaisiin tarkkailla. Tällöin työntekijän muutokset eivät voi vaikuttaa yleiseen tietokantaan. Testaaminen myös helpottuu, sillä ongelmatilanteissa voi tarkistaa tietokannastaan, millaisessa muodossa data varsinaisesti tallentuu. Työtehtävien monipuolisuuden ja edellä mainitun vapauden takia, käytettiin henkilökohtaista tietokantaa lähes kaikissa työtehtävissä. Olivat ne sitten tietokannan tai käyttöliittymän päivittämistä.

4.2 JavaScript

JavaScript on ohjelmointikieli, jota käytetään verkkosivujen interaktiivisuuden ehostamiseen. Se mahdollisti verkkosivujen muutosten visualisoinnin ilman verkkosivun uudelleen lataamista. (Haverbeke 2024a.)

JavaScript on ohjelmointikielenä melko vapaamuotoinen. Se ei aiheuta poikkeuksia käyttäjän koodista kovinkaan helposti. Tästä syystä käyttäjä voi esimerkiksi kirjoittaa funktiokutsun funktiolle, jota ei ole olemassa ja virhe paljastuu vasta ohjelman ajaessa. Tämä on osittain seurausta siitä, että JavaScript on niin sanottu dynaaminen ohjelmointikieli. Sen muuttujilla ei tarvitse olla ennalta määritettyä datatyyppiä. Tämä mahdollistaa useamman datatyypin datan asettamisen samaan muuttujaan. Tämä ei tosin ole mahdollista samanaikaisesti. (Haverbeke 2024b.)

Toinen syy edellä mainitulle ominaisuudelle on se, että JavaScriptillä ei varsinaisesti ole kääntäjää. JavaScript tulkataan ja ajetaan JavaScript-tulkilla. Tämän ohjelman tarkoitus on ajaa JavaScript koodia. Suurimmassa osassa käytössä olevilla verkkoselaimilla on oma JavaScript-tulkkinsa. Tästä syystä oletusarvoisesti JavaScriptin ajamiseen tarvitaan verkkoselain. Tämän takia useimmat syntaksivirheet eivät näy ohjelmoijalle ennen ohjelman ajamista. Syntaksin oikeellisuus tarkastetaan JavaScript-tulkin toimesta, kun ohjelmaa ajetaan. (Javapoint 2024a.) JavaScriptissä on mahdollisuus rajoittaa tätä ominaisuutta. Jos JavaScript-tiedostoon lisää ”use strict”-direktiivin, käytössä oleva tulkki estää koodin ajamisen, jos koodi sisältää esimerkiksi muuttujia, joita ei ole määritelty. Tämä ominaisuus on käytössä useimmissa tulkeissa. (Javapoint 2024c.)

4.3 NodeJs

Työprojektissa käytetään taustajärjestelmän ajamiseen NodeJs-ajoympäristöä. Sen ominaispiirre on, että se voi ajaa JavaScript-ohjelmaa selaimen ulkopuolella. (Sufiyan 2024.) Tämä ominaisuus avaa mahdollisuuden valita kehittäjän käytössä olevat moduulit ja kirjastot. Kunkin selaimen käyttämä versio JavaScriptistä määrittää moduulit kehittäjän käytössä JavaScript-tulkkinsa puitteissa, kun taas NodeJs ei rajoita JavaScript-tulkin versiota. NodeJssästä puuttuu joitakin ominaisuuksia, jotka selaimesta löytyvät. Yksi näistä on esimerkiksi valmis pohja rakentaa käyttöliittymää minkä selain tarjoaa dokumentin muodossa. (Shew 2024.)

NodeJsässä funktiot ja luokat järjestetään moduuleihin. Nämä ovat tiedostoja ja kansioita, jotka pitävät sisällään haluttuja funktioita. Lähtökohtaisesti eri moduulien funktiot eivät voi vuorovaikuttaa toistensa kanssa. Moduulit saatetaan toistensa käyttöön require-funktiolla.

Joitain osia koodista ei tarvitse tuoda moduulista moduuliin. Nämä oliot tai funktiot ovat niin sanotusti julkisia, eli ne ovat kaikkien projektin moduulien käytössä. Moduuleja voidaan lisätä projektiin pakettihallintaohjelman avulla. (Sufiyan 2024.)

NodeJssän toimintaperiaate perustuu vahvasti asynkroniseen tiedonkäsittelyyn. NodeJs käsittelee jokaisen palvelimelle tulleen pyynnön asynkronisesti samalla pitäen silmällä uusia palvelimelle tulevia pyyntöjä. Tästä syystä NodeJssää pidetään nopeana palvelimen ajoympäristönä. (Sufiyan 2024.)

4.4 Express

Express on JavaScript-kehys joka on luotu NodeJssän pohjalle helpottamaan internetviestien välittämistä ja vastaanottamista käsittelevän logiikan kirjoittamista. Expressin voi asentaa projektiin NPM-pakettihallintaohjelman avulla. Express sisältää funktioita tunnistamaan ja suorittamaan CRUD-pyyntöjä ja kuuntelemaan noita pyyntöjä halutun portin kautta. (University of Helsinki 2022.)

Expressin toiminta nojaa vahvasti kolmeen muuttujaan

- Pyyntömuuttuja sisältää taustajärjestelmälle lähetettyyn pyyntöön liittyvän datan.
- Vastausmuuttuja sisältää taustajärjestelmän lähettämään vastaukseen liittyvän datan.
- Osoitemuuttuja määrittää mihin osoitteeseen lähetetty pyyntö laukaisee funktion kutsun.

Ainoa pakollinen muuttuja niin sanotun kommunikointifunktion toiminnan kannalta on osoitemuuttuja. Ilman sitä taustajärjestelmä ei tiedä mikä funktio tulisi kutsua pyynnön tullessa. Osoitteeseen voidaan myös tavallaan lisätä pyynnön kannalta oleellista dataa. Tälle datalle annetaan paikka osoitteessa kaksoispistesyntaksin avulla. Tämä data on osa pyyntömuuttujan params-oliota ja siitä voidaan käsitellä tuon olion avulla. (University of Helsinki 2022.) Tätä ominaisuutta käytettiin työprojektissa eritoten, kun taustajärjestelmään lähetettiin käyttäjän tietoja koskevia pyyntöjä. Eri käyttäjät erotetaan toisistaan tunnisteen avulla ja tuo tunniste lähetettiin taustajärjestelmään kaksoispistesyntaksin avulla.

Expressissä voidaan lisätä palautusmuuttujalle statuskoodi. Tämä palautetaan palautusmuuttujan mukana http-statuskoodina. Se ilmaisee, miten palvelimelle lähetetyn pyynnön käsittely onnistui. (University of Helsinki 2022.) Tästä on hyötyä virhetilanteissa. Taustajärjestelmän palauttamasta statuskoodista voidaan suurin päätellä, mistä virheessä on kyse. Esimerkiksi jos statuskoodi on 404, tämä tarkoittaa sitä, että http-pyynnölle annetulle

osoitteelle ei löytynyt vastinetta palvelimelta. Tästä voidaan päätellä, että ongelma koodissa on todennäköisesti väärin kirjoitettu pyynnön osoite tai taustajärjestelmän polun määritys.

Työprojektissa hyödynnetään myös express-async-handler-kirjastoa. Express-async-handler on Express-kehyksen pohjalta luotu kirjasto, jonka tarkoitus on suoraviivaistaa poikkeustilanteiden käsittelyä Express-funktioissa. (Abazhenov 2021).

5 Versiohallinta

5.1 Keskitetty ja hajautettu versiohallinta

Versiohallinta on palvelu, jonka tarkoitus on tehdä jonkin digitaalisen projektin eri versioiden ylläpitämisestä yksinkertaisempaa. Kun projektia on tekemässä useampi kuin yksi henkilö, versiohallinta voi myös pitää kirjaa siitä, kuka on tehnyt mitäkin muutoksia projektiin ja milloin. (Ackermann ym. a.) Jos päivitys aiheuttaa ongelmia projektissa, voi tuon päivityksen tekijän löytää helpommin ja pyytää häntä joko täsmentämään tekemäänsä muutosta tai korjaamaan sen aiheuttaman haitan.

Versiohallinnassa on kaksi päälinjaa, keskitetty versiohallinta ja hajautettu versiohallinta. Keskitetyssä versiohallinnassa projektista on yksi kokonainen versio, jonka erillisiä osia kukin projektin kanssa työskentelevä muokkaa. Hajautetussa versiohallinnassa jokaisella projektia muokkaavalla yksilöllä on oma kopionsa koko projektista. (Ackermann ym. a.)

Keskitetyn versiohallinnan etu on se, että se mahdollistaa selkeän jaottelun siitä, kuka työskentelee minkäkin projektin osan kanssa. Täten ristiriitojen syntyminen on vaikeampaa, kun päivityksiä tehdään. Koska projekti on oletuksena tallennettu kokonaisena yhteen paikkaan, on vaara, että projekti saatetaan menettää lähes kokonaisuudessaan. (Ackermann ym. a.) Tämä voidaan kiertää tekemällä projektista säännöllisiä varmuuskopioita.

Hajautetun versiohallinnan etu on, että koko projektin menettäminen on lähes mahdotonta. Jokaisen työntekijän oma versio toimii varmuuskopiona. Haittapuolena on se, että mikään ei estä jokaista työskentelijää muokkaamasta samoja tiedostoja eri tavoilla, joka aiheuttaa ristiriitoja, kun muutoksia tuodaan yhteen. (Ackermann ym. a.) Tämän haittapuolen voi kiertää pitämällä yllä tarkkaa työnjakoa ja pysymällä ajan tasalla projektin päivityksistä.

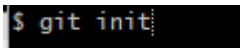
5.2 Git

Git on avoimeen lähdekoodiin perustuva versionhallintatyökalu. Se on ilmainen ladata ja käyttää missä tahansa projektissa ja kontekstissa. (Ackermann ym. b.) Siihen sisältyy ylempänä mainitut ominaisuudet. Kahdesta edellä mainitusta versiohallinnan linjasta Git lukeutuu hajautettuun versiohallintaan.

Gitin ensisijainen ominaispiirre on haarautumisjärjestelmä. Tällä järjestelmällä käyttäjä voi tehdä kopion projektistaan, josta tulee oma haaransa. Tämä haara on kokonaan itsenäinen versio alkuperäisestä projektista ja siihen tehdyt muutokset eivät vaikuta alkuperäiseen versioon tai muihin haaroihin. Eri haaroihin tehdyt muutokset voidaan myöhemmin yhdistää toisiinsa, jolloin ne jakavat toistensa muutokset. (Ackermann ym. b.) Näiden haarojen avulla

Git voi jäljitellä keskitetyn versiohallinnan piirteitä. Jos yhtä haaraa käytetään vain yhden projektin osan muokkaamiseen ja toista haaraa toisen, nämä muutokset ovat epätodennäköisemmin toistensa kanssa ristiriidassa.

Git voidaan asettaa seuraamaan mitä tahansa kansiota ajamalla git init-komento tietokoneen komentopaneelissa. Esimerkki tästä komennosta on kuvassa (Kuva 19). Tämä luo tuohon kansioon Gitin tarvitsemat tiedostot. Tuosta kansiosta tehdään projektin ensimmäinen haara. Tätä haaraa kutsutaan eri nimillä eri käyttöjärjestelmissä. Tässä tekstissä sitä kutsutaan tästä eteenpäin päähaaraksi. Päähaara toimii Gitin versiohallinnan pohjana. Kaikki muut haarat polveutuvat siitä. (Chacon & Straub 2024, 22.)



Kuva 19. Git init

Kun Git seuraa jotakin projektia se tallentaa tuon projektin historian kokonaisuudessaan sille koneelle, jossa projekti on. Tämä määrä tietoa saadaan tallennettua siten, että Git ei tallenna projektin tiedostoja kokonaisuudessaan, vaan tallentaa eräänlaisia jäljennöksiä niistä. Nämä jäljennökset pitävät sisällään vain projektiin tehdyt muutokset. Jos muutoksia ei ole, Git ei tallenna muuttumattomia tiedostoja vaan osoitteen tiedoston vanhempaan versioon. Tietokanta pitää projektia tallessa, sillä jos Gitin tietokantaa ei hävitetä, voi projektin palauttaa tuon tietokannan avulla. (Chacon & Straub 2024, 15.)

On huomattu, että on työn kannalta edullista työstää yhtä asiaa kerrallaan ja palauttaa niitä sen mukaan, kun ne valmistuvat. Tällä tavalla jo valmiiksi saadut osat ohjelmasta eivät katoa, jos työn myöhemmissä vaiheissa on syytä palauttaa projekti edelliseen versioon. Tähän voi olla useita syitä. Useimmissa tapauksissa edellisen version palautus tai muutosten poistaminen halutaan tehdä, kun tehtyjä muutoksia ei tarvitakaan tai tehdyt muutokset estävät jonkin projektin osan toiminnan. Joskus tehdyt muutokset voidaan kumota manuaalisesti, mutta joissain tilanteissa tämä ei ole järkevää tai mahdollista. Tällaisiin tilanteisiin on päädytty esimerkiksi silloin, kun jotain ongelmaa on yritetty ratkaista monella eri tavalla ja samalla muutettu useita osia projektista. Myöhemmin on huomattu, että ongelman voi ratkaista helpommin tai ongelman ratkaisusta on luovuttu. Tällöin tehdyt muutokset ovat tulleet tarpeettomiksi.

5.2.1 Ulkoiset versiot

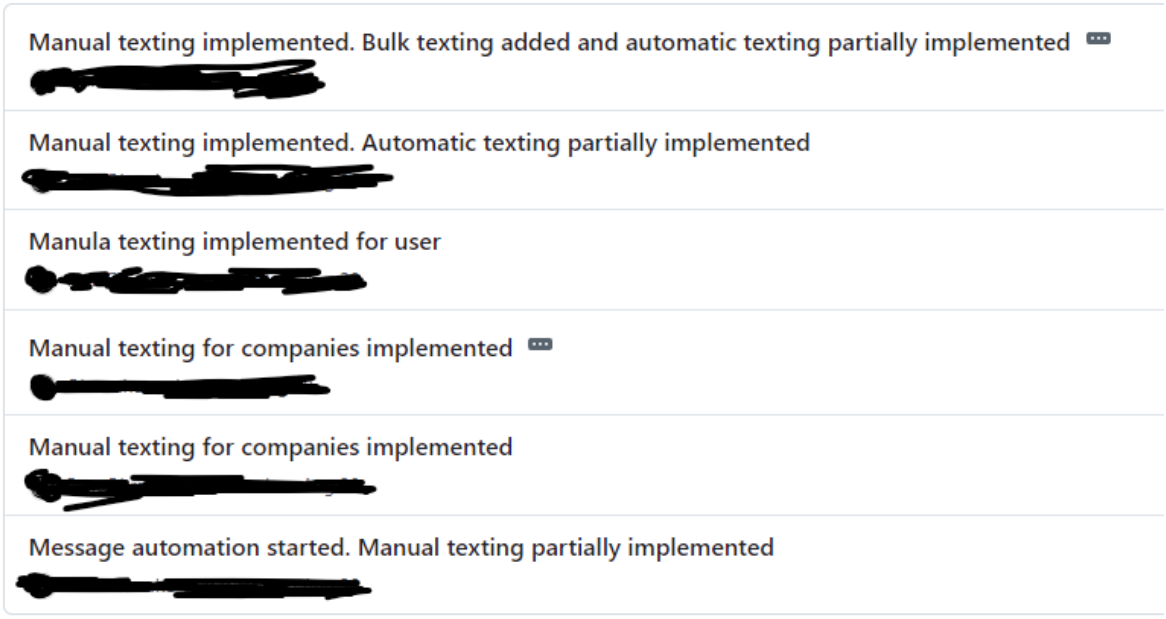
Versiohallinnassa ulkoiset versiot ovat versioita projektista, jotka on sijoitettu työkoneen ulkopuolelle. Yleensä ne ovat jonkin palvelimen muistissa. Nämä ovat useimmissa tapauksissa projektin lopullisia versioita, joihin kaikkien työntekijöiden päivitykset lisätään. Ulkoiset

versiot ovat myös yleensä se versio projektista, jonka uudet työntekijät kloonaavat omalle koneelleen.

Gitissä on sisäänrakennettuna toimintoja, jotka auttavat työskentelemään ulkoisten versioiden kanssa. Näitä komentoja on erilaisia, mutta käytännössä ne kaikki joko vetävät dataa ulkoisesta versiosta käyttäjän versioon tai työntävät käyttäjän version dataa ulkoiseen versioon. (Chacon & Straub 2024, 52.) Tällä tavalla jokainen projektin kanssa työskentelevä voi itsenäisesti päivittää projektia ja ottaa muiden päivitykset käyttöönsä halutessaan.

5.2.2 GitHub

GitHub on Gitin ympärille luotu verkkopalvelu, jonka tarkoituksena on helpottaa Gitin käyttöä ja toimia Git-projektien jakoalustana (Github Inc. b). GitHubiin on luotu käyttöliittymiä ajamaan Gitin komentoja. Näistä päällimmäisenä on Gitin versiohistorian esittäminen. GitHubissa koko projektin versiohistoriaa voidaan tarkastella versio versiolta. GitHub voi myös visualisoida kunkin palautuksen tekemät muutokset. (GitHub Inc. c.) Esimerkki versiohistoriasta nähtävissä kuvassa (Kuva 20).



Kuva 20. GitHubin versiohistoria

GitHubiin on lisätty toiminnallisuuksia suoraviivaistamaan eri kehittäjien yhteistyötä. Keskeisin esimerkki tästä on sen visualisoimat puskut ja palautukset sekä niihin lisätty vetopyyntö. Vetopyyntö on projektin haaraa puskettaessa luotu välivaihe, joka esittää dataa puskusta. Näitä ovat esimerkiksi puskettavan haaran eroavaisuus puskun alla olevaan haaraan ja mahdolliset ristiriidat näiden haarojen välillä. Vetopyynnöissä voi myös avata keskusteluja muiden projektin kanssa työskentelevien kanssa. (GitHub inc. d.)

Jotta projektin historia pysyisi mahdollisimman selkeänä on hyvä lisätä Gitin palautuksiin kommentti. Puhdas Git antaa tälle mahdollisuuden palautuksen yhteydessä. GitHubissa on samantapainen ominaisuus, mutta GitHub vaatii kommentin lisäämistä jokaiseen palautukseen. (GitHub Inc. c.) Jotta tästä ominaisuudesta olisi hyötyä, tulisi kommentin selittää selkeästi, mitä muutoksia palautus projektiin teki. Tällä tavalla historiasta voi lukea mitä kukin palautus on saanut aikaan ilman, että koodia täytyy analysoida.

Työprojektin aikana GitHubia käytettiin projektin yleisen version ylläpitämiseen ja jakamiseen. Tähän versioon jokainen työntekijä lopulta palautti työnsä tuloksen. Vaikka GitHub tarjoaakin työkaluja moderoida mitä päivityksiä yleiseen versioon lisätään, tätä ei tapahtunut projektin aikana. Kukin kehittäjä nautti luottamusta siitä, että hänen kirjoittamansa koodi toimisi muun sovelluksen kanssa ja jos koodissa ilmenikin virheitä, kirjoittajan oletettiin joko omatoimisesti tai pyynnöstä korjaavan virheen. Tämä mahdollisti päivitysten lisäämisen nopeasti ja sulavasti, mutta johti silloin tällöin ongelmiin. Joskus yleinen projekti jouduttiin palauttamaan vanhempaan versioon, kun pusketut muutokset aiheuttivat tunnistamattomia virheitä yleisen version toiminnassa. Näiden palautusten takia saatettiin menettää tehtyä työtä.

6 Testausmenetelmät

6.1 Yleiset testausmenetelmät

Työprojektissa ei hyödynnetty mitään varsinaista testausmenetelmää. Kukin kehittäjä testasi tekemänsä ominaisuudet itsenäisesti. Tällaiseen manuaaliseen testaamiseen kuluu poikkeuksetta aikaa, kun eri arvoja syötetään ohjelmalle ja tarkastellaan niiden tuloksia. Joissain tapauksissa ohjelma tulee käynnistää uudestaan, jos testin vaikutukset pitää pyyhkiä tai tarkastella, miten sivun uudelleen lataaminen vaikuttaa ohjelman toimintaan. Nämä testit joudutaan usein toistamaan, kun koodia muokataan. Useimmissa tapauksissa testit ovat hyvin samanlaisia joka testauskerralla. Tällaisissa tilanteissa automaattinen testausjärjestelmä olisi voinut säästää paljon aikaa.

Tietokoneohjelmien automaattista tai ennalta kirjoitettua testausta kutsutaan ohjelmistotestaukseksi. Tämä tarkoittaa varsinaisesta ohjelmasta erillään olevia funktioita tai ohjelmia, jotka kutsuvat testattavan ohjelman funktioita ja vertaavat kutsuttujen funktioiden antamia tuloksia ennalta määriteltyihin mallituloksiin. Testaamisen tarkoituksena on varmistaa, että kirjoitettu ohjelma toimii odotusten ja vaatimusten mukaan. Testauksella voidaan myös havaita ohjelmassa ilmeneviä virheitä, kun ohjelmaa muokataan. (Testsigma Technologies Inc.)

Lähes mitä tahansa ohjelman osa-aluetta voidaan testata oikealla työkalulla. Ohjelmistotestauksen menetelmillä voidaan myös simuloida ohjelman suorituskykyä koettelevia tilanteita, kuten usean viestin lähettämistä tai kuvien lataamista lyhyellä aikavälillä. Testauksessa pyritäänkin varmistamaan, että kaikki ohjelman osa-alueet toimivat, kun ohjelmaa päivitetään. Tapoja päästä tähän tavoitteeseen on yhtä monta kuin on testityökaluja, mutta näiden työkalujen tarkoitukset voidaan jakaa karkeasti viiteen eri kategoriaan

- Toiminnallisuustestaus testaa ohjelman yleistä toimivuutta ja tarkistaa tekeekö jokin funktio tai vastaava sitä mitä sen on tarkoitus tehdä.
- Suorituskyvyn testaus testaa ohjelman suorituskykyä ääritilanteissa.
- Käytettävyystestaus testaa ohjelman toimintaa, kun ohjelmaa käyttää ihminen.
- Turvallisuustestaus testaa mahdollisia heikkouksia ohjelmassa.
- Yhteensopivuustestaus testaa ohjelman toimivuutta eri käyttöliittymissä tai ajoympäristöissä.

Järjestelmätestaus voidaan suorittaa joko manuaalisesti tai automaattisesti testityökalun mukaan. Näiden kahden ero on, että manuaalisessa testauksessa testi täytyy käynnistää

käyttäjän toimesta, kun taas automaattisessa testauksessa testi tapahtuu aina tietyn vaatimuksen täytyttyä. Tämä vaatimus saattaa olla tiedoston tallentaminen, ohjelman ajaminen tai uudelleen käynnistäminen. (Testsigma Technologies Inc.)

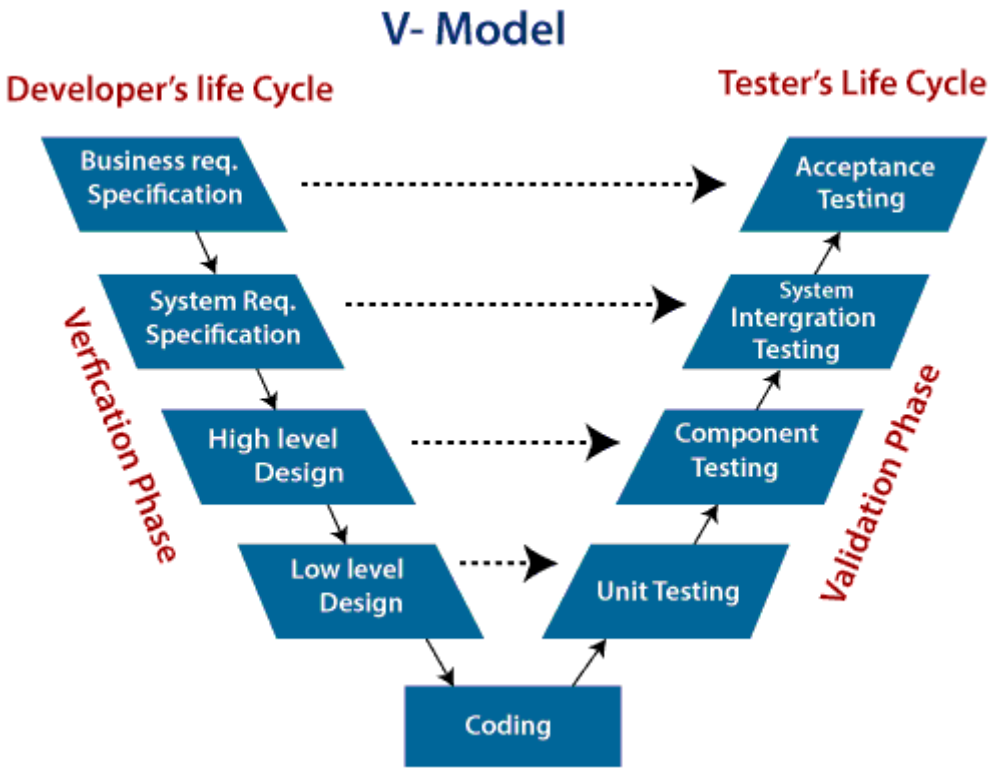
6.2 V-malli

Yksi suurimpia syitä sille, miksi ohjelmistotestausta ei implementoitu työprojektiin oli se, ettei sitä oltu tehty projektin aikaisemmissa vaiheissa. Ohjelmien suunnittelu ja ohjelmistotestauksen implementointi ovat vahvasti kytköksissä toisiinsa. Suunnitteluvaiheessa tehdyt päätökset vaikuttavat siihen, miten ja millaisia testifunktioita myöhemmin luodaan. Tätä kuvastaa niin sanottu ohjelmistotestauksen v-malli, joka on yksi ohjelmistokehityksen elinkaarista (Javapoint 2024b). Visualisointi V-mallista on nähtävissä kuvassa (Kuva 21).

V-malli jakautuu kolmeen osioon:

- vahvistusvaihe
- validointivaihe
- implementointivaihe

Vahvistusvaihe keskittyy ohjelmiston suunnitteluun ja testauksen suunnitteluun. Implementointivaiheessa suunnitellut ominaisuudet ja testit luodaan ja validointivaiheessa testit ajetaan. Implementointivaihe on mallin näkökulmasta irrallinen osa verrattuna kahteen muuhun vaiheeseen. (Javapoint 2024b.)



Kuva 21. V-Model (Javapoint 2024b)

Kuten kuvassa 21 nähdään vahvistusvaihe ja validointivaihe ovat yhteydessä toisiinsa. Vahvistusvaiheen aikana suunnitellaan niin ohjelmisto kuin ohjelmiston testit. Tämä suunnittelu jakautuu eri vaiheisiin:

- asiakkaan tarpeen analysointi
- systeemin tarpeiden analysointi
- korkeamman tason suunnittelu
- alemman tason suunnittelu

Asiakkaan tarpeen analysointi alkaa tarpeen määrittämisestä. Tässä vaiheessa pyritään kartoittamaan mahdollisimman tarkasti, mitä asiakas tarkalleen haluaa ja mitä vaatimuksia tarpeen toteuttamiselle on. Kun projektin parametrit on saatu selville, voidaan siirtyä seuraavaan vaiheeseen. (Javapoint 2024b.) Systeemin tarpeiden analysoinnissa määritellään mitä ohjelma tarvitsee tarpeen tyydyttävää implementointia varten. Tässä vaiheessa päätetään esimerkiksi, mitä ohjelmointikieltä tai kehystä käytetään ja miten taustajärjestelmän kanssa menetellään. (Swain 2024.) Näiden päätösten puitteissa voidaan alkaa suunnitella ohjelman varsinaisia osia. Korkeamman tason suunnittelussa keskitytään ohjelman suurempiin osiin, kuten ohjelmiston arkkitehtuuriin, käytettävien moduulien kartoittamiseen,

rajapintoihin ja ohjelmiston riippuvaisuuksiin. Alemman tason suunnittelu keskittyy yksittäisten komponenttien ja moduulien toimintaan. (Javapoint 2024b.)

Jokaisen osion yhteydessä on kyseisen osion testauksen suunnittelu. Kun osiossa saadaan selville ohjelmiston vaatimuksia, voidaan noiden vaatimusten pohjalta suunnitella testejä varsinaiselle ohjelmistolle. Pienin näistä on yksikkötestaus, joka testaa alemman tason suunnittelun tuotoksia. Nämä ovat yksittäisiä funktioita, luokkia tai moduuleja. Tästä korkeampi testaus on integrointitestaus, joka on yhteydessä korkeamman tason suunnitteluun. Sen tavoitteena on varmistaa, toimivatko ohjelman eri osat yhdessä. Systeemitestauksen aikana pyritään testaamaan koko ohjelmiston toiminta kaikkine komponentteineen ja yhteyksineen. Viimeisenä testinä suoritetaan hyväksyntätestaus. Tämä testi pyrkii varmistamaan suorittaako ohjelma sille alun perin annetun tehtävän onnistuneesti ja asiakkaan vaatimusten mukaan. Tämä testi määrittää onko ohjelmisto valmis vai ei. (Swain 2024.)

Työprojektin työprosessi oli verrattavissa tähän malliin, mutta testauksen puuttuessa mallia seurattiin puoliksi. Aluksi saatiin tieto uuden toiminnallisuuden tarpeesta ja tämän toiminnallisuuden vaatimat osat kartoitettiin. Jos valmista kaavaa niiden toteuttamiselle ei vielä ollut olemassa, niin yksittäisten osien tarpeet pilkottiin erillisiksi ominaisuuksiksi. Tämän jälkeen noita osia alettiin luomaan. Prosessin loppuun kuului vielä luodun päivityksen testaus, mutta se suoritettiin manuaalisesti.

6.3 Flutterin testausmenetelmät

Flutter sisältää valmiita työkaluja testien luomiselle. Nämä työkalut mahdollistavat edellisessä luvussa kuvattujen testien luomisen, mutta Flutteriin kuuluu myös erillinen versio yksikkötestauksesta komponentteja varten.

Komponenttitestauksen tarkoitus on testata yksittäisen komponentin toimintaa. Tämän testin laajuus on sidonnainen testattavan komponentin laajuuteen. Tähän testiin voi lukeutua myös käyttöliittymän toimintaa koskevat testit. Esimerkki yksinkertaisesta komponenttitestistä on kuvassa (Kuva 22). (Lougheed 2024a.) Kuvan pumpWidget-funktio luo testattavan komponentin (Lougheed 2024c).

```
void main() {
  testWidgets('MyWidget has a title and message', (tester) async {
    // Create the widget by telling the tester to build it.
    await tester.pumpWidget(const MyWidget(title: 'T', message: 'M'));
  });
}
```

Kuva 22. Komponenttitesti (Lougheed 2024c)

Flutterissa testaukset tapahtuvat testifunktioiden kautta. Näihin funktioihin kirjoitetaan, mitä funktiota tai komponenttia halutaan testata, mitä parametrejä tuossa testissä käytetään ja mikä tulos näistä tulisi saada. Yksikkötestauksissa tämä on vielä suoraviivaista, mutta mitä laajempialaisiin testeihin siirrytään, sitä enemmän eri lisäfunktioita ja testifunktioiden yhdistelmiä tarvitaan. Esimerkiksi komponenttitestauksessa komponenttien tila ei vaihdu automaattisesti, vaikka muuttavaa funktiota testissä kutsuttaisiinkin. Tilan muuttamiseen tarvitaan erillinen funktio, johon on kirjoitettava, milloin komponentin tilaa on tarkoitus muuttaa.

Integraatiotestaus ajaa sarjan eri yksikkötestejä ja komponenttitestejä. Se voidaan asettaa tarkkailemaan niiden vaikutuksia muihin testeihin. Tällä tavalla voidaan luoda testiverkosto, joka käy koko ohjelman läpi. Integrointitestauksessa on myös mahdollista asettaa testille testiympäristö, kuten käyttöjärjestelmä, mobiililaite tai muu vastaava ympäristö, jossa ohjelmaa voidaan ajaa. (Lougheed 2024a.)

Flutter kykenee suorittamaan nämä testit automaattisesti, kun muutoksia tehdään. Oikeanlaiset testit voivat korvata manuaaliset testaukset tietyissä tilanteissa ja säästää aikaa ohjelmaa muokatessa. (Lougheed 2024a.)

6.4 Taustajärjestelmän testausmenetelmät

Koska taustajärjestelmää ajetaan NodeJs-ympäristössä, on suoraviivaisinta hyödyntää testitarkoitukseen luotua NodeJs-moduulia. Yksi moduulivaihtoehto on test-moduuli. Tuo moduuli pitää sisällään funktioita, jotka tarkastelevat, jos funktio aiheuttaa poikkeuksen, annettu lupaus evätään tai jos done-funktion palautekutsu ajetaan tosien parametrien kanssa. Kaikissa näissä tapauksissa testi katsotaan epäonnistuneeksi.

Varsinainen testi ajetaan test-funktion sisällä. Funktio ottaa parametrikseen merkkijonon, jolla testiä on tarkoitus kuvata. Tämä merkkijono toimii ikään kuin testin nimenä. Toinen parametri on funktion, jossa varsinainen testilogiikka on. Yksinkertainen testifunktio on

esitetty kuvassa (Kuva 23). Tämä funktio voidaan asettaa asynkroniseksi, jos se on tarpeellista.

```
test('synchronous passing test', (t) => {
  // This test passes because it does not throw an exception.
  assert.strictEqual(1, 1);
});
```

Kuva 23. Testifunktio (Atlow ym. 2024)

Testien välille voi rakentaa omaa logiikkaansa testien ohittamisen ja testien sisäisten testien muodossa. Testin ohittaminen on verrattavissa ehtolauseeseen. Jos jokin väittämä on totta, niin testiä ei suoriteta. Testin sisälle voidaan kirjoittaa alatestejä. Jos testin sisällä on testejä nuo testit on läpäistävä jotta niiden ympärillä oleva suurempi testi läpäistäisiin. Test-moduulin testit aktivoidaan manuaalisesti komentopaneelin kautta. Komento on node –test. Tämä komento ajaa kaikki testitunnisteella varustetut tiedostot, mutta tätä aluetta voidaan rajata tarvittaessa lisäämällä komentoon muuttujia. Esimerkki tällaisesta muuttujasta on merkkijono, joka rajaa minkä nimisiä tiedostoja komento ajaa. Muuten komento ajaa käytännössä kaikki tiedostot, joiden nimessä on .test. (Atlow ym. 2024.)

Toinen esimerkki komennon täsmentämiselle on only-komento. Tämä komento ajaa edelleen kaikki päätestit, mutta erottaa niiden sisäiset testit ajettaviin ja ajamatta jätettäviin. Alatesti merkataan ajettavaksi only-komennolla, jos siihen merkitään tosi only-parametri. Tällä tavalla voidaan tarkemmin erotella, mitä osia testistä halutaan ajaa, jos esimerkiksi suuremman olion yhtä osaa halutaan testata, mutta sille ei haluta kirjoittaa kokonaan omaa testiä. Yksittäisiä testifunktioita voidaan myös kutsua niiden nimiparametrin avulla. Testi ajetaan, jos sen nimiparametrin merkkijono vastaa komentoon kirjoitettua kaavaa. (Atlow ym. 2024.)

7 Uuden ominaisuuden toteuttaminen

7.1 Ominaisuuden kuvaus

Yksi suurimmista päivityksistä työprojektiin oli niin sanottu kuvanlataajakomponentti ja sen ympärille tehdyt funktiot niin käyttöliittymässä kuin taustajärjestelmässä. Tavoitteena oli luoda komponentti, jonka avulla käyttäjä voi ladata koneeltaan kuvan sovelluksen käyttöön ja esittää tämä ladattu kuva käyttöliittymässä. Ensisijainen tarve tällaiselle komponentille oli käyttäjien profiilikuvat ja yritysten logot. Tällaiselle komponentille on myös muita käyttötarkoituksia, kuten kuvien lisääminen resursseihin tai huoltojen kommentteihin.

Lopputuloksena tulisi siis olla komponentti, joka esittää käyttöliittymässä tyhjän kuvan kehyksen, jossa on mekanismi kuvan lataamiseen. Ladattu kuva näkyisi kehyksessä. Kehys täytyy olla muokattavissa kontekstiin sopivaksi niin koon kuin muodon osalta. Komponentista tulee tehdä versio, jolla käyttäjä voi ladata useita kuvia samaan kohteeseen. Tällaisissa tilanteissa tulee myös ottaa huomioon ruudun koon tuomat rajoitteet kuvia esittäessä.

Ladatun kuvan tiedostokokoa pitää olla mahdollista alentaa tallennustilan säästämiseksi ja suurempien kuvien tallentamisen mahdollistamiseksi. Kuvat pitää myös pystyä konvertoimaan webp-muotoon.

Ladatuille kuville on kehitettävä kuvapankki taustajärjestelmään. Kuvia ei haluta tallentaa tietokantaan, koska potentiaalisesti tuhannet kuvat veisivät tietokannassa liikaa tilaa. Kuvat tallennetaan siis palvelimelle. Järjestelmä tarvitsee tavan luoda tiedostoja ja kansioita sekä muokata molempien sisältöä. Tämän lisäksi tavoitteena oli luoda tallennusjärjestelmä siten, että tietokantaan ei tarvitsisi tallentaa mitään. Tiedostorakenteen tulee jakaa käyttäjien ja yritysten kuvat kansioihin yrityksen tai käyttäjän nimen ensimmäisen kirjaimen mukaan. Kuvia on pystyttävä poistamaan ja lataamaan tästä kuvapankista.

7.2 Kuvanlataajan toteutus

Kuvanlataajan ensimmäinen implementointi kohdistui käyttäjän profiilikuvaan. Latausmekanismiksi valittiin kuvan tai tyhjien kehysten painaminen. Tästä tuli standardi, jonka pohjalta kuvanlataaja luotaisiin.

Ensimmäinen tavoite oli löytää tapa ladata kuva käyttäjän koneelta. Flutteriin on luotu erilaisia kirjastoja tätä tarkoitusta varten. Toteutuksessa hyödynnettiin image_picker-kirjastoa, joka antaa työkalut kuvien lataamiseen käyttäjän koneelta ja kuvien ottamiseen mobiililaitteen kameralla ja niiden saattamiseen sovelluksen käyttöön. (Baker ym. 2024). Jälkimmäistä ominaisuutta ei käytetty.

Image_picker-kirjasto tekee mahdolliseksi luoda funktion, jota kutsuttaessa käyttäjä voi valita kuvakansiostaan haluamansa kuvan ja ladata sen sovelluksen käyttöön. Työprojektin kuvanlataajafunktio on esitetty kuvassa (Kuva 24). Ladattavalle kuvalle on myös mahdollista valita kuvanlaatu asteikossa 0-1. Mitä lähempänä nollaa tämä luku on, sitä alhaisempi kuvan laatu tulee olemaan. (Baker ym. 2024.) Tämä mahdollisti kuvien pakkaamisen latausvaiheessa, mutta kuvan kokoa ei tällä tavalla saatu tarpeeksi pieneksi kuvan laadun kärsimättä liikaa. Tallennustilaa on kuitenkin säästettävä aina kun mahdollista, joten kaikki kuvat ladataan lähes poikkeuksetta lähtötasoa alemmalla laadulla. Varsinkin pienemmissä profiilikuvissa tämä ei vaikuttanut kuvan ulkonäköön merkittävästi.

Image_picker-kirjaston funktio palauttaa ladatun kuvan XFile-oliona (Baker ym. 2024). Koska käytössä oleva komponentti, joka visualisoi kuvat, ottaa vastaan Uint8List-tyypin muuttujia, piti XFile-muuttuja muuttaa muotoon Uint8List. XFile-luokalla on funktio, joka palauttaa XFile-olion Uint8List-muodossa (pub.dev). Tätä funktiota hyödyntämällä saatettiin luoda funktio, joka lataa kuvan käyttäjän koneelta ja palauttaa tuon kuvan suoraan Uint8List-muodossa.

```
Future<Uint8List?> fetchImageFromUser(int quality) async {
  final XFile? pickedFile =
    await ImagePicker().pickImage(source: ImageSource.gallery, imageQuality: quality);

    Uint8List webImage = Uint8List(8);

    if(pickedFile != null) {
      var newImage = await pickedFile.readAsBytes();
      webImage = newImage;
      logger.d("${pickedFile.name}: ${webImage.lengthInBytes}");
    }

    return pickedFile != null ? webImage : null;
}
```

Kuva 24. Kuvanlataajafunktio

Latausmekanismin toteuttamisessa käytettiin GestureDetector-komponenttia. Tämä komponentti havaitsee, jos sen lapsikomponentin kanssa vuorovaikutetaan määrätyllä tavalla. Vuorovaikutus määritettiin painamiseksi. Kun tämä interaktio tapahtuu GestureDetector kutsuu kuvanlataajafunktiota, kuten esitetty kuvassa (Kuva 25). (Windmill 2020. 130.)

```
class _ImageUploaderState extends State<ImageUploader> {
  bool loading = false;

  @override
  Widget build(BuildContext context) {
    return GestureDetector(
      onTap: () async {
        try {
          bool? temp = await imageEditDialog(context);
          //Downloading the image
          if (temp != null && temp) {
            setState(() {
              loading = true;
            });
            Uint8List? fetchedImage = await fetchImageFromUser(60);
```

Kuva 25. Kuvanlataaja osa 1

Kuvan lataamisen jälkeen ajetaan kuvan pakkaamiseen liittyvää logiikkaa. Jos kuva pakataan onnistuneesti, kuva esitetään käyttöliittymässä. Tämän implementointi esitetään kuvassa (Kuva 26).

```
          if (fetchedImage != null) {
            Uint8List? downGradedImage = await fixImageSize(fetchedImage);

            if(downGradedImage != null) {
              bool tooLargeImage = checkUploadedImage(context, downGradedImage, 1000000);
              if(tooLargeImage) {
                setState(() {loading = false;});
                widget.returnImageString(base64Encode(downGradedImage));
              } else {
                setState(() {loading = false;});
              }
            } else {
              //Error message
              temporaryMessage(context, AppLocalizations.of(context)!.tooBigImage, 2);
              setState(() {loading = false;});
            }
          } else {
            //Error message
            temporaryMessage(context, AppLocalizations.of(context)!.tooBigImage, 2);
            setState(() {loading = false;});
          }
        } else if (temp != null && !temp) {
          widget.returnImageString("");
          setState(() {loading = false;});
        }
      } catch (error) {
        setState(() {
          loading = false;
        });
        logger.d("Error while uploading image $error");
      }
```

Kuva 26. Kuvanlataaja osa 2

Kuvan esittämiseen käyttöliittymässä hyödynnettiin Container-komponenttia. Tämän komponentin muuttujiin kuuluu decoration-muuttuja. Tuolla muuttujalla voidaan muokata Container-komponentin ulkonäköä. Tämän muuttujan arvoksi annettiin BoxDecoration-komponentti, joka mahdollistaa kuvan lisäämisen Containerille image-muuttujan avulla. BoxDecoration-komponentin shape-muuttujalla voidaan myös määrittää Containerille uusi muoto (Hickson & Jhonson. 2024). Kuvattu implementointi on esitetty kuvassa (Kuva 27). Image-muuttujaan asetettiin MemoryImage-komponentti, joka esittää kuvan käyttöliittymässä Containerinsa ulottuvuuksien puitteissa. MemoryImage ottaa kuvan vastaan Uint8List-muodossa. (Hickson & Williams 2024.)

```
Widget imageDisplay(double size, bool sircle, String image, {Icon icon = const Icon(Icons.add)}) {
  return sircle
      ? (image != ""
          ? Container(
              width: size,
              height: size,
              decoration: BoxDecoration(
                shape: BoxShape.circle,
                image: DecorationImage(
                  image: MemoryImage(base64Decode(image)),
                  fit: BoxFit.cover,
                ), // DecorationImage
              )) // BoxDecoration // Container
          : CircleAvatar(
              radius: size / 2,
              child: icon,
            ))
      : (image != ""
```

Kuva 27. Kuvan esittävä komponentti

Jos kuvaa ei ole, kuvan kehykset näytetään tyhjinä ja Containerin lapseksi asetetaan Icon-komponentti. Tämän avulla kehysten keskelle voidaan asettaa haluttu ikoni. Lopullisessa funktiossa ikoni on vapaaehtoinen parametri. Jos ikonia ei erikseen syötetä, ikoni on plus-merkki. Tällä tavalla yritetään viestiä, että tyhjä kuvakehys on jotain, jonka kanssa käyttäjä voi vuorovaikuttaa. Profiilikuvan kontekstissa ikoni vaihdetaan ihmisen siluettiin. Kuvattu logiikka on nähtävissä kuvassa (Kuva 28).

```
      //Show loading indicator if loading
      child: loading ?
      SizedBox(
        width: widget.size,
        height: widget.size,
        child: Center(
          child: CircularProgressIndicator(
            color: Theme.of(context).colorScheme.primary
          ) // CircularProgressIndicator
        ) // Center
      ) :  // SizedBox
      //Display image
      imageDisplay(
        widget.size,
        widget.circle,
        widget.imageString,
        icon: widget.icon ?? const Icon(Icons.add)
      ),
    ); // GestureDetector
  }
}
```

Kuva 28. Kuvanlataaja osa 3

Useamman kuvan lataaminen samaan kohteeseen toteutettiin luomalla funktio, joka palauttaa kuvanlataajalistan sille asetetun kuvalistan mukaan. Tämä funktio lisättiin komponenttiin, joka esittää tuon listan joko rivinä tai jonona riippuen ruudun leveydestä. Rivin tai jonon loppuun asetetaan painike, jota painamalla käyttäjä voi lisätä uuden tyhjän kuvanlataajan joukon jatkoksi. Kuvanlataajajono on esitetty kuvassa (Kuva 29).

Kuva 29. Kuvanlataaja käyttöliittymässä kuvan kanssa ja ilman kuvaa

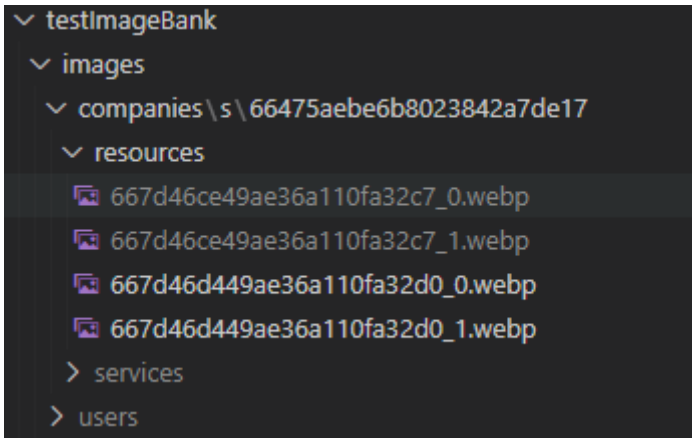
Joissain tapauksissa tarvittiin mahdollisuus esittää paljon kuvia pienessä tilassa. Esimerkki tälle on huoltojen kommenttikenttä. Yhdellä kommentilla voi olla useita kuvia ja ne on näytettävä kentässä järkevästi. Jos kuvia on useampia, ne asetetaan eräänlaiseen kuvakollaasiin. Kuvankaappaus kuvakollaasista on kuvassa (Kuva 30). Tämä näyttää kommentin neljä ensimmäistä kuvaa kehyksen sisällä. Kaikki kuvat saadaan näkyviin listana painamalla kuvakollaasia.

Kuva 30. Kommentin kuvakollaasi

7.3 Kuvapankin toteutus

Tiedostojen käsittelyssä hyödynnettiin NodeJssän File-system-moduulia. Moduuli antaa työkalut tiedostojen luomiseen, poistamiseen, kopiointiin, tiedostoihin kirjoittamiseen, tiedostojen erottamiseen kansioista ynnä muita tiedostojen manipuloimiseen liittyviä työkaluja. (Hamel ym. 2024.)

Suurin ongelma kuvapankin luonnissa oli suunnitella tallennuslogiikka, joka luo yllä kuvatun tiedostorakenteen. Tämä ongelma ratkaistiin hyödyntämällä tietokannan käyttämiä tunnisteita tiedostorakenteen luomisessa. Tunnisteet ovat uniikkeja jokaiselle oliolle ja eivät muutu, ellei oliota tuhota. Niiden avulla voidaan luoda tiedostorakenne, jota voidaan navigoida ilman polkujen tallentamista. Koska jokaisella tietokantaan tallennettavalla oliolla on taustajärjestelmässä omat tallennusfunktionsa, saatettiin niiden lomaan lisätä logiikka, joka luo kuvapankkiin tarvittavat polut. Esimerkiksi jos halutaan tallentaa resurssien kuvia, niiden lopullinen tiedostorakenne näyttäisi seuraavalta (Kuva 31):



Kuva 31. Kuvapankin tiedostorakenne

Polun pohjan tälle tiedostorakenteelle luo funktio, joka ottaa vastaan totuusarvomuuttujan user, merkkijonon id ja toisen merkkijonon name. Funktion toiminnan kannalta pakolliset parametrit ovat user ja id. User-muuttuja kertoo, jos polkuun tulee lisätä images-sanan jälkeen companies vai users. Jos user-muuttuja on totta, lisätään users ja jos ei, niin lisätään companies. Nimen käsittelylogiikka on nähtävissä kuvassa (32).

```
async function makeBasePath(user, id, name) {
    var path = ""
    var newName = ""
    console.log(name)
    if(user) {
        path += "users"
        if(name === null || name == undefined) {
            newName = await getUserName(id)
        } else {
            newName = name
        }
    } else {
        path += "companies"
        if(name === null || name == undefined) {
            newName = await getCompanyName(id)
        } else {
            newName = name
        }
    }
    var alphabetPath = makeAlphabetPath(newName)
    if(alphabetPath != null) {
        path += `/${alphabetPath}`
    }
    return path += `/${id}`
}
```

Kuva 32. Polun alun luontifunktio

Tämän jälkeen tuleva kirjain määritellään käyttäjän tai yrityksen nimen ensimmäisen alkukirjaimen mukaan. Tarvittavan olion tiedot haetaan tietokannasta annetun id-tunnisteen avulla, jos nimiparametria ei ole funktioon määritelty. Kirjainta seuraava merkkijono on yrityksen tai käyttäjän tunniste. Tämä saadaan niin ikään id-parametristä. Tämän jälkeinen sana lisätään sen mukaan, minkä olion tallennusfunktio on kyseessä. Sana lisätään sen jälkeen, kun pohjapolku on valmistunut.

Resources kansion tiedostot ovat tallennetut kuvat. Kuvien nimet määrittyvät sen resurssin tunnisteesta, jonka kuvia ollaan tallentamassa. Koska yksi resurssi voi pitää sisällään monia kuvia ja samassa kansiossa ei voi olla montaa saman nimistä tiedostoa, pitää kuvien nimet erotella jotenkin. Tässä tapauksessa ne erotetaan toisistaan alaviivalla ja järjestysnumerolla. Järjestysnumero määräytyy tallennettavien kuvien määrästä (Kuva 29).

7.4 Kuvapankin manipuloiminen

Edellisessä osassa kuvattua logiikkaa käytetään kaikessa kuvien manipulaatiossa. Kuvapankin toiminnassa oli vielä vaatimuksia ja ongelmakohtia, joihin piti löytää vastaus. Ensimmäinen näistä on kytköksissä polkua tekevään funktioon. Jos polussa näkyviä kansioita ei ole olemassa, ei tuon polun kansioihin voi tallentaa mitään. Tämä ongelma ratkaistiin lisäämällä tallennuslogiikkaan funktio, joka käy kasatun polun läpi ja tarkistaa, ovatko kaikki siihen merkityt kansiot olemassa. Jos jotain kansiota ei löydy, se luodaan funktiolla, joka esitetään kuvassa (Kuva 33).

```
//Takes path as a parameter and creates that path as empty folders
async function goThroughFolderPath(path) {
    var splitPath = path.split("/");
    var currentPath = ""
    for(var i = 0; i < splitPath.length; i++) {
        currentPath += `/${splitPath[i]}`;
        await makeMissingFolder(currentPath);
    }
}

//Makes folder to given location
async function makeMissingFolder(folder) {
    var exists = await checkIfFolderExists(folder);
    if(!exists) {
        await fs.mkdir(`${baseUrl}${folder}`);
    }
}
```

Kuva 33. Polun rakennusfunktiot

Toinen ongelma liittyi kuvien poistamiseen. Koska kuvat ovat osa oliota, kuvien muokkaaminen tapahtuu olion tietojen päivittämisen yhteydessä. Olion päivittäminen toimii siten, että uusi data kirjoitetaan vanhan datan päälle. Jos jostain olion listasta on poistettu kohteita, näitä kohteita ei poisteta tietokannasta vaan muokattu lista kirjoitetaan vanhan listan päälle. Tämä ei tapahdu samoin kuvapankin kohdalla. Jos kuvapankkiin tallennetaan kuva, jonka nimi on sama kuin jo olemassa oleva kuva, uusi kuva tallennetaan vanhan päälle. Jos taas kuva on poistettu ei kuvapankki tunnista tätä vaan jättää poistetun kuvan kuvapankkiin. Tämä ongelma ratkaistiin siten, että kuvia sisältävän olion päivittämisen yhteydessä, kaikki vanhat kuvat, joissa on tuon olion tunniste, poistetaan ja uudet kuvat tallennetaan tavalliseen tapaan.

Viimeinen ongelma oli mahdollisuus sille, että käyttäjä voi muuttaa nimeään tai yrityksensä nimeä. Koska vaaditussa tiedostorakenteessa yritykset ja käyttäjät tuli jaotella kansioihin nimen perusteella, piti kehittää keino, jolla muuttuvan nimen mahdollisuus otetaan huomioon. Tämä ratkaistiin lisäämällä funktio, joka vertaa yrityksen tai käyttäjän nimeä vanhaan nimeen käyttäjän tai yrityksen tietojen päivityksen aikana. Jos nimen ensimmäinen kirjain ei ole muuttunut, mitään ei tarvitse tehdä. Jos taas kirjain on muuttunut, niin tavallisten päivitystoimenpiteiden lisäksi käyttäjän tai yrityksen kansio kopioidaan uutta nimeä vastaavaan kansioon ja vanha kansio poistetaan. Koska kansiota, joka pitää sisällään tiedostoja ei voi poistaa suoraan, niin poistologiikkaan piti lisätä funktio, joka käy läpi kansion sisällön ja poistaa kaikki tiedostot yksi kerrallaan. Kansion kopiointifunktio on esitetty kuvassa (Kuva 34).

```
//Copies folder into another folder and deletes the old folder
async function copyFolderToOtherFolder(folderToCopy, destinationFolder) {
    console.log(`old folder: ${folderToCopy}\nnew folder: ${destinationFolder}`)

    if(await checkIfFolderExists(folderToCopy)) {
        //If no destination folder exists, it is made
        if(await checkIfFolderExists(destinationFolder)) {
            await fs.cp(`${baseUrl}/${folderToCopy}`, `${baseUrl}/${destinationFolder}`, {recursive: true},
                (err) => {
                    console.log("Error while copying folder: " + err)
                })
            await deleteFolder(folderToCopy)
        } else {
            await goThroughFolderPath(destinationFolder)
            await fs.cp(`${baseUrl}/${folderToCopy}`, `${baseUrl}/${destinationFolder}`, {recursive: true},
                (err) => {
                    console.log("Error while copying folder: " + err)
                })
            await deleteFolder(folderToCopy)
        }
    }
}
```

Kuva 34. Kansion kopiointifunktio

7.5 Kuvan pakkaamisen toteutus

Koska kuvien tallentaminen tietokannan ulkopuolelle saatiin toimimaan, ei kuvien pakkaamisella ollut enää suurin prioriteetti. Tälle kuitenkin yritettiin löytää vaihtoehtoja, koska suurien kuvien lähettäminen taustajärjestelmään JSON:in avulla ei ollut mahdollista. Pakkaamiselle yritettiin löytää erilaisia tapoja tai kirjastoja, mutta löydettyjä kirjastoja ei saatu toimimaan. Tarvittavat kirjastot olivat vanhentuneita.

Tämä jätti ainoaksi vaihtoehdoksi kuvan laadun alentamisen. Tavoitteena oli luoda funktio, joka pienentää kuvan laatua niin kauan, että sen voi lähettää taustajärjestelmään. Kuvien pakkaamiseen käytettiin image-paketin encodeJpg-funktiota**.** Tämä funktio muuttaa kuvan jpg-muotoon. Sillä voi myös alentaa kuvan laatua, täten pienentäen sen kokoa. Tulokset olivat hyvin epätasaisia kuvien välillä. Vaikka kuvan laadun laskemiseen käytettiin näennäisesti prosentuaalista laskutapaa, niin kuvan koko ei muuttunut samassa suhteessa. Tästä syystä ei voitu luoda luotettavaa laskuria sille, kuinka paljon kunkin kuvan laatua tulisi laskea.

Ongelma ratkaistiin osittain asettamalla ladattu kuva silmukan sisään, joka ajaa pakkausfunktion uudelleen ja uudelleen, kunnes tavoiteltu koko on saavutettu. Tämä toimi, mutta prosessi oli hidas. Suuremmissa kuvissa saattoi kestää useita kymmeniä sekunteja ennen kuin kuva saatiin oikean kokoiseksi.

Ajankäytön ongelmaa yritettiin lieventää jättämällä silmukan ajaminen käyttäjän päätökseksi. Kun kuva pakattaisiin ensimmäisen kerran ja se silti olisi liian suuri, ilmestyisi ponnahdusikkuna, joka kysyisi käyttäjältä haluaako hän pakata kuvan uudelleen. Tätä jatkettaisiin, kunnes kuva pienenisi tarpeeksi tai käyttäjä antaisi kieltävän vastauksen. Tämä ei ratkaise ajankäytön ongelmaa mutta antaa käyttäjälle valinnan tuon ajan käyttämisestä.

Tämä toteutus saatiin toimimaan osittain, mutta kun kuvanpakkausfunktio kutsuttiin, käyttöliittymä jäätyi. Tämä johti siihen, että ponnahdusikkuna ei kadonnut käyttäjän valinnan jälkeen. Tätä yritettiin kiertää siten, että kuvan pakkaus siirrettäisiin ohjelmassa toiseen säikeeseen, jolloin käyttöliittymä voisi päivittyä samalla kun kuvaa pakataan. Tämä ei ollut kuitenkaan mahdollista sovelluksen verkkoversiossa.

Lopulta päädyttiin vaihtoehtoon, jossa kuva ajettaisiin pakkaussilmukan läpi korkeintaan kahdesti. Ensimmäisellä kerralla laatua pienennettäisiin melko vähän, mutta jos se ei riitä niin pakkaus suoritettaisiin uudelleen pienemmällä kertoimella. Tällä tavalla saatiin suurin osa kuvista tallennettavaan kokoon. Jos kuva ei käsittelyn jälkeen tavoittanut vaadittua kokoa, niin käyttöliittymään näytetään viesti, jossa kerrotaan ladattavan kuvan olevan liian suuri. Pakkausfunktio on nähtävissä kuvassa (Kuva 35).

```
Future<Uint8List?> fixImageSize(Uint8List webImage) async {
  try {
    //Check image resolution
    if(highImageResolution(webImage, 1000000)) {
      img.Image? image = img.decodeImage(webImage);
      webImage = await changeImageResolution(
        webImage,
        (image!.width/2).round(),
        (image.height/2).round()
      );
    }
    //Check image size
    if(bigImage(webImage)) {
      logger.d("Too big image: ${webImage.lengthInBytes}");

      Uint8List smaller = await downgradeImageQuality(webImage, 50);
      logger.d("Bit smaller: ${smaller.lengthInBytes} bytes");
      //If not too big, return image
      if(!bigImage(smaller)) {
        return smaller;
      }
      //If too big, downgrade again
      if(bigImage(smaller)) {
        Uint8List smaller = await downgradeImageQuality(webImage, 25);
        if(!bigImage(smaller)) {
          return smaller;
        }
        logger.d("Too big: ${webImage.lengthInBytes} bytes");
      }
      return null;
    } else {
      logger.d("Returned image size: ${webImage.lengthInBytes}");
      return webImage;
    }
  } catch(error) {
    logger.d("Error while fixing image size: $error");
    return null;
  }
}
```

Kuva 35. Kuvan pakkausfunktio

8 Ominaisuuden muokkaaminen

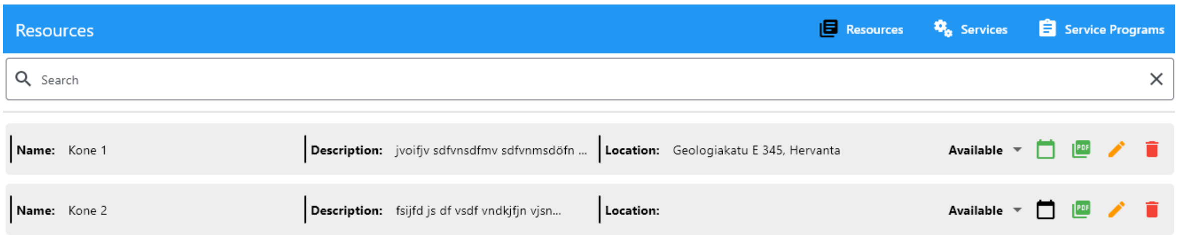
8.1 Ominaisuuden lisääminen

Useimmissa tapauksissa ominaisuuspäivitykset työprojektissa olivat uuden muuttujan lisääminen olioon ja sen ympärille suunniteltujen toiminnallisuuksien toteuttaminen. Tähän sisältyi usein jonkinlainen muuttujan visualisointi käyttöliittymässä ja muuttujan yhdistäminen sovelluksen taustajärjestelmään. Ensimmäiseksi käyttöliittymän järjestelmään lisätään uusi muuttuja ja sille tarvittava logiikka, jotta se toimisi jo rakennetussa logiikassa. Tähän kuuluu välittäjäohjelma ja uudet funktiot API-palvelusta vastaavaan olioon. Tämän jälkeen siirrytään taustajärjestelmään ja luodaan sinne tarvittavat osat kuten polut, CRUD-pyyntöjä käsittelevät funktiot ja olio, jonka pohjalta varsinainen data luodaan tietokantaan. Näiden jälkeen luodaan yksinkertainen käyttöliittymä, yhdistetään se välittäjään ja testataan, jos yhteys taustajärjestelmään on toimiva. Kun tämä on varmistettu, voidaan alkaa parantelemaan käyttöliittymän ulkoasua ja lisäämään siihen toimintoja.

Tällä tavalla voidaan saada yksinkertainen versio ominaisuudesta toimimaan suhteellisen nopeasti. Jos ominaisuus ei poikkea paljoa edellisistä ominaisuuksista, voidaan edellisten ominaisuuksien funktiot ottaa pohjaksi ja muokata niistä toimintalogiikka uudelle ominaisuudelle. Kuitenkin jos ei ole huolellinen, saattaa uuden ominaisuuden funktioihin jäädä pohjana käytetyn funktion konsoliviestejä tai kommentteja. Nämä voivat myöhemmin aiheuttaa hämmennystä. Tapauksissa, joissa samalaista koodia toistuu usein, on hyödyllistä yrittää korvata samanlaisen koodin toistaminen yleispätevämmällä koodilla.

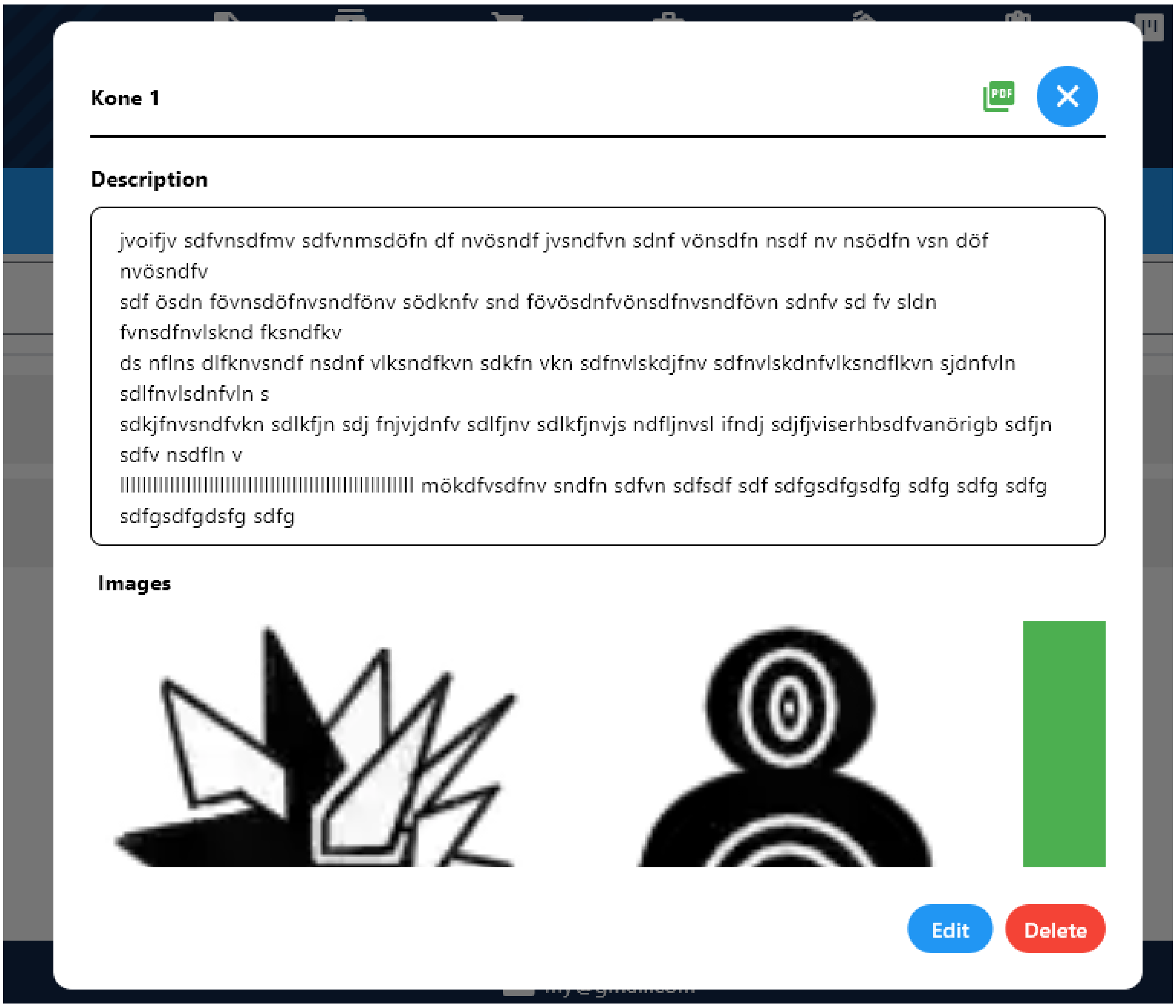
Kun uutta ominaisuutta aletaan työstämään eteenpäin, on hyvä käyttää aikaa suunnittelemaan ominaisuuden tiedostorakennetta. Jos ominaisuus on pieni, esimerkiksi uusi ponnahdusikkuna tai uusi suodatin hakujärjestelmään, voidaan se lisätä jo olemassa oleviin tiedostoihin. Jos kyseessä on suurempi kokonaisuus, kuten monimutkainen ponnahdusikkuna, uusi alasivu tai kokonaan uusi sivu, niin tälle ominaisuudelle on hyvä luoda oma tiedostonsa. Tällä tavalla voidaan välttää yksittäisten tiedostojen kasvaminen liian suuriksi. Yleisesti ottaen yksi tiedosto pitää sisällään ominaisuuden yhden osan. (Salvatierra 2022.) Työprojektissa esimerkki tällaisesta tiedostojaottelusta on jo edellä mainitut resurssit. Resurssien käyttöliittymä kasautuu pääosin kolmesta osasta:

- Resurssien lista. joka on kuvassa (Kuva 36).



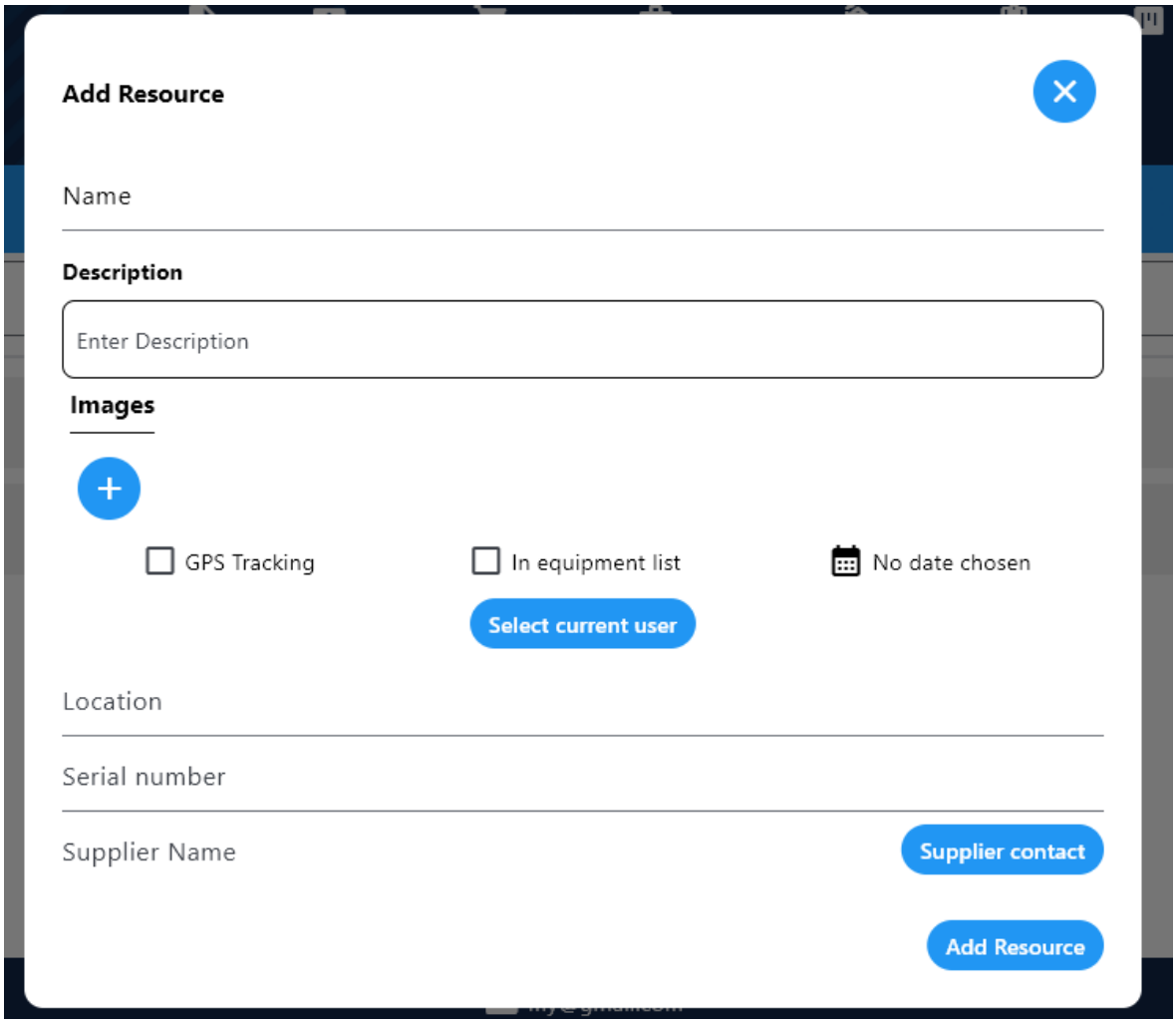
Kuva 36. Resurssilista

- Resurssin tietolomake, joka näkyy kuvassa (Kuva 37).



Kuva 37. Resurssin tietolomake

- Resurssin lisäyslomake, joka on kuvassa (Kuva 38).



Kuva 38. Resurssin lisäyslomake

Vaikka kukin näistä tarvitsee toistaan luodakseen toimivan resurssien käyttöliittymän, ovat ne tarpeeksi erossa toisistaan ja tarpeeksi laajoja, että niiden asettaminen omiin tiedostoihinsa on perusteltua. Tällaista arkkitehtuuria käytetään työprojektissa usein, sillä moni toiminto seuraa resurssien lista-, luonti- ja infotarpeita.

Pienempien kokonaisuuksien kanssa voi olla parempi pitää kokonaisuuden eri osat samassa tiedostossa. Lukuisat pienet tiedostot luovat ennen pitkää saman ongelman kuin yksi suuri tiedosto. Esimerkiksi työprojektiin luotu commonWidgets-tiedosto pitää sisällään useita eri komponentteja, jotka eivät varsinaisesti liity toisiinsa. Niiden yhdistävä tekijä on se, että niitä käytetään monessa osassa ohjelmaa. Näiden komponenttien kohdalla yksi suuri tiedosto koetaan paremmaksi verrattuna moneen pieneen, sillä moni tiedoston komponentti on rivikooltaan melko pieni. Kaikkien yleispätevien komponenttien kasaaminen yhteen tiedostoon helpottaa myös niiden jakamista muun ohjelman käyttöön.

8.2 Virheen korjaaminen

Työprojektissa korjattiin monenlaisia virheitä. Virhe saattoi olla käyttöliittymän virheellinen päivittyminen tai päivittämättä jättäminen, käyttöliittymän osan näkyminen ei toivotulla tavalla, tarpeeton osa käyttöliittymässä tai toimintavirhe taustajärjestelmässä. Useimmat virheet, joita työprojektissa korjattiin, olivat niin sanotusti itse aiheutettuja.

Jotta virhe voitaisiin korjata se pitää pystyä tunnistamaan. Tämä on tärkein osa virheen korjaamisen prosessia. Kun virhekohta löydetään ja virheen aiheuttaja tunnistetaan, voidaan siirtyä suunnittelemaan tapaa korjata virhe.

Useimpien ohjelmointikielien kääntäjissä on sisään rakennettuna joitain työkaluja, jotka auttavat virheiden löytämisessä. Tällaisissa tapauksissa virhekohdan löytäminen ja virheen tunnistaminen on suhteellisen suoraviivaista, kun koodieditorin konsolista voi löytää ongelman aiheuttavan tiedoston ja rivin, josta virhe sai alkunsa. Tällaisia virheviestejä ei kuitenkaan ilmene, jos ohjelman toiminnassa ei ole ongelmaa teknisesti. Tällöin koodi toimii, mutta sen suorittama toiminto ei johda haluttuun lopputulokseen.

Tällaisessa tilanteessa ensimmäinen vaihe virheen korjaamisessa on sen toistaminen. Tämä tarkoittaa ohjelman käyttämistä siten, että etsitty virhe tapahtuu. Tämä auttaa tunnistamaan tarkemmin, mikä ohjelmassa ei tarkalleen toimi ja auttaa keskittämään huomion oikeaan osaan ohjelmaa.

Esimerkkinä tällaisesta tilanteesta on, kun käyttöliittymässä tulisi olla lista olioita, mutta tätä listaa ei jostain syystä näy. Tällainen ongelma toistetaan varmistamalla, että tietokannassa on listan olioita, vaikka ne eivät listassa näkyisi. Useimmissa tapauksissa olioiden puute johtui siitä, että oliolistaa ei saatu tuotua oikein ja täten käyttöliittymän lista on tyhjä. Tällaisessa tilanteessa virheen aiheuttamaa aluetta voi rajata asettamalla konsoliviestejä ominaisuutta käsitteleviin funktioihin. Näillä pyritään rajaamaan aluetta, jossa virhe voisi olla. Alue pyritään kaventamaan yhteen funktioon ja siitä yhteen osaan tuota funktiota. Tätä prosessia voisi nopeuttaa, jos projektiin implementoisi automatisoituja testejä merkittävimpiin kohtiin. Esimerkiksi edellä kuvatun konsoliviestirajauksen voisi jättää useimmissa tapauksissa tekemättä, jos listaa käsitteleville funktioille kirjoitettaisiin testifunktiot.

Useimmissa tapauksissa kun virhettä korjataan, niin kyse on korjaajan itsensä kirjoittamasta koodista. Joskus kuitenkin päädytään korjaamaan virhettä ohjelmassa, jota ei olla itse kirjoitettu. Saattaa myös olla mahdollista, että virhe johtuu korjaajan ja jonkun muun kirjoittaman koodin välisestä ristiriidasta. Tällaisissa tilanteissa on tärkeää, että korjaajan ja muiden ohjelmoijien kirjoittama koodi on mahdollisimman selkeää. Yksi helpoimmista tavoista

edistää tätä on nimetä funktiot, luokat ja muuttujat kuvaavasti. Joissain tapauksissa tämä ei riitä ja silloin on hyvä lisätä koodiin kommentti.

Jokaiselle funktiolle, jonka toiminta ei ole täysin ilmiselvää kannattaa kirjoittaa kuvaus kommenttina. Tämä auttaa niin ikään muita ohjelmoijia ymmärtämään kirjoitettua ohjelmaa, kuten myös alkuperäisen koodin kirjoittajaa, jos hän palaa työskentelemään koodin kanssa myöhemmin. Kommentoinnista voi olla myös apua koodin jaottelemisessa helpommin luettaviin osiin. (McKee 2023.) Yksinkertaisistakin Flutterin komponenteista kasvaa helposti kymmenien rivien mittaisia. Monimutkaiset käyttöliittymän osat saattavat nousta yli tuhanteen riviin. Tällaisten kokonaisuuksien lukemista ja muokkaamista voidaan helpottaa, jos sitä jaetaan eri osiin kommenttien avulla. Esimerkiksi kaksi tekstikenttää näyttävät tekstitiedostossa hyvin samanlaisista, mutta voivat olla yhteydessä täysin eri toiminnallisuuksiin ohjelmassa. Tässä tapauksessa kuvaava kommentti molempien yläpuolella auttaa erottamaan ne nopeasti toisistaan. Myös rivinvaihto voi helpottaa erottamaan käyttöliittymän eri osat koodissa (McKee 2023).

9 Yhteenveto ja pohdinta

Opinnäytetyössä tutustuttiin käytössä olleiden työkalujen ominaisuuksiin teoriassa ja käytännössä. Työn aikana tutkittiin myös mahdollisuuksia parantaa työn tulosta tai helpottaa työprosessia tulevaisuudessa. Esimerkkinä tästä ovat testijärjestelmät. Ennen testifunktioita ei pidetty tärkeinä. Nyt suuremman projektin kanssa työskentelemisen jälkeen, ymmärretään niiden potentiaali säästää aikaa ja yhdenmukaistaa kehitysprosessia.

Osa päiväkirjassa- ja opinnäytetyössä kuvatuista ominaisuuksista jäivät osiksi projektia. Toiset osoittautuivat tarpeettomiksi tai korvattiin paremmalla järjestelmällä. Esimerkiksi kuvanlataajaa käytetään kaikissa projektin kuvaimplementoinneissa ja universaalia listakomponenttia käytetään monissa paikoin. Useimmiten kun johonkin olioon halutaan lisätä yhteystietoja tai muita olioita.

Kaikissa implementoinneissa huomattiin ennemmin tai myöhemmin puutteita, joita ovat myös muut ohjelmoijan korjanneet. Esimerkiksi listakomponentti näyttää olioiden tiedoista vain nimen ja tämä on joissain tapauksissa liian vähän informaatiota. Tulevaisuudessa tätä voisi laajentaa siten, että kehittäjä voisi halutessaan luoda oman komponenttinsa, jonka pohjalta lista loisi olioille komponentit.

Kuvanlataaja vaatii pakkaukselle paremman järjestelmän. Kuvapankki on myös melko alkeellisella tasolla ja esimerkiksi tallennusjärjestelmä, joka hyödyntää tietokannan tunnisteita, saattaa aiheuttaa turvallisuusriskejä. Käyttäjän tunnisteella voidaan päästä käsiksi käyttäjän tietoihin ja tuo tunniste on nähtävissä kuvapankin tiedostorakenteessa. Tämän voisi estää tekemällä tunnisteet tunnistamattomiksi.

Päiväkirjan ylläpitäminen auttoi pohtimaan oman työn tulosta kriittisesti ja mahdollisuuksia parantaa työtä tulevaisuudessa. Se konkretisoi työn aikana tapahtunutta kehitystä. Alkuvaiheessa tehtävän päivityksen valmistumisessa saattoi kulua suurin osa päivästä, mutta työjakson lopussa samanlainen päivitys oli vain yksi monista päivän töistä. Päiväkirja toimi myös muistikirjana. Jos yhtenä päivänä oli kehitetty toimiva ratkaisu jollekin ongelmalle, voitiin tämä ratkaisu hakea päiväkirjasta, kun sama ongelma uudistui.

Tämä tietyllä tavalla osoitti dokumentoinnin tärkeyden. Vaikka kirjoitus ohjelman toiminnot ja rakenne olisivatkin selviä, tämä ei välttämättä ole totta muiden ohjelmoijien tai tulevaisuuden itsensä näkökulmasta. Tämän takia on hyvä pitää kirjaa tekemistään muutoksista, joko kommenttien tai muun dokumentoinnin muodossa.

Opinnäytetyön prosessi toi itsevarmuutta omaan osaamiseen niin käyttöliittymän kuin taustajärjestelmän kanssa työskentelystä. Sisäistettiin myös laajempialaisen

ohjelmistokehityksen realiteetteja. Saatiin myös varmuus siitä, että vaikka jokin asia ei olisikaan entuudestaan tuttu, niin kyetään tästä etsimään tietoa ja soveltamaan sitä tehtävän tarpeisiin. Tämä taito osoittautui mahdollisesti tärkeimmäksi, koska alati muuttuvat työtehtävät uudessa kehyksessä vaativat nopeaa ymmärrystä kulloisestakin aiheesta. En usko, että tämä tulee tulevaisuudessa muuttumaan, joten tätä oppimiskykyä tulee olemaan aina tarpeen harjoittaa.

Työprojekti on ensimmäinen suuremman skaalan projekti, jonka kanssa on työskennelty. Tästä huolimatta prosessin loputtua oli työskennelty lähes jokaisen projektin osan kanssa ja luotu siihen päivityksiä. Koen, että opinnäytetyöprosessi on muuttanut minut web-kehityksen opiskelijasta työkykyiseksi web-kehittäjäksi.

Lähteet

Abazhenov. 2021. Express-async-handler. Viitattu 05.09.2024. Saatavissa https//www.npmjs.com/package/express-async-handler

Ackermann, T., Ahmadi, R., Avila, J., Bachhuber, C., Bently, S., Bernard, B., Blain, P., Bodnar, P., Bobryck, S., Bonaventura, X., Buehler, E., Byster, J., Cannon, B., Chacon, S., Chen, R., Cooper, M., Day, R., Dewender, J., Donnelly, C., Dopplinger, P., D'Souza, R., Drejhammar, F., Eyherabide, H., Fiers, T., Forrest, J., Gabrielssen, A., Garnier, A., Gathoye, W., Goncalves, W., Griffin, B., Hietala, S., Hochbaum, E., Ho, Y., Holey, R., Jacobs, S., Jakubowski, M., Javid, M., Joseph, S., Karg, M., Kerr, R., Kuznetsov, S., Kuznetsov, V., Leigh, N., Leinweber, K., Ljubenovic, L., Loiseau, A., Lum, K., Luo, Y., Meng, S., Mendez, A., Menshikov, M., Miossec, P., Nadagouda, P., Nakayama, H., Ollier, A., Oortwijn, J., Pipe, A., Radchenko, M., Samoylov, O., Sanjaya, B., Smirnov, D., Szumny, K., Turek, B., Varma, A., Velkovski, K., Vries, H., Wilson, C., Yatsenko, K., Zhang, W., Zelle, R. & Ziller, D. 2024a. Getting Started – About Version Control. Viitattu 08.07.2024. Saatavissa https//git-scm.com/book/en/v2/Getting-Started-About-Version-Control

Ackermann, T., Ahmadi, R., Avila, J., Bachhuber, C., Bently, S., Bernard, B., Blain, P., Bodnar, P., Bobryck, S., Bonaventura, X., Buehler, E., Byster, J., Cannon, B., Chacon, S., Chen, R., Cooper, M., Day, R., Dewender, J., Donnelly, C., Dopplinger, P., D'Souza, R., Drejhammar, F., Eyherabide, H., Fiers, T., Forrest, J., Gabrielssen, A., Garnier, A., Gathoye, W., Goncalves, W., Griffin, B., Hietala, S., Hochbaum, E., Ho, Y., Holey, R., Jacobs, S., Jakubowski, M., Javid, M., Joseph, S., Karg, M., Kerr, R., Kuznetsov, S., Kuznetsov, V., Leigh, N., Leinweber, K., Ljubenovic, L., Loiseau, A., Lum, K., Luo, Y., Meng, S., Mendez, A., Menshikov, M., Miossec, P., Nadagouda, P., Nakayama, H., Ollier, A., Oortwijn, J., Pipe, A., Radchenko, M., Samoylov, O., Sanjaya, B., Smirnov, D., Szumny, K., Turek, B., Varma, A., Velkovski, K., Vries, H., Wilson, C., Yatsenko, K., Zhang, W., Zelle, R. & Ziller, D. 2024b. About. Viitattu 29.06.2024. Saatavissa https//www.git-scm.com/about

Atlow, M., Hamel, A., Ihrig, C. & Wendel, E. 2024. Test runner. Viitattu 17.08.2024. Saatavissa https//nodejs.org/api/test.html#test-runner

Austin, D., Hickson, I., Patil, A. & Porcaro, L. 2024a. Dialog class. Viitattu 12.09.2024. Saatavissa https//api.flutter.dev/flutter/material/Dialog-class.html

Austin, D., Hickson, I., Patil, A. & Porcaro, L. 2024b. AlertDialog class. Viitattu 12.09.2024. Saatavissa https//api.flutter.dev/flutter/material/AlertDialog-class.html

Baker, R. & Beusekom, B. & Mackall, G. 2024. Image Picker plugin for Flutter. Viitattu 13.09.2024. Saatavissa https//pub.dev/packages/image_picker

Belanger, M., Eronmosele, V. & Lougheed, P. 2024. Dart overview. Viitattu 05.07.2024. Saatavissa https//dart.dev/overview

Bizzotto, A. 2022. Flutter Project Structure Feature-first or Layer-first? Viitattu 10.09.2024. Saatavissa https//codewithandrea.com/articles/flutter-project-structure/

Chacon, S. & Straub, B. 2024. Pro Git. 2.1.430. uudistettu painos. New York Appress

Feizollah, A. 2023. Understanding Flutter Project Structure Explorin Each Folder and its Purpose. Viitattu 05.08.2024. Saatavissa https//www.youtube.com/watch?v=xP0rt64dT6Q

Gascon, U. & Seip, J. 2024. Hello world example. Viitattu 28.09.2024. Saatavissa https//expressjs.com/en/starter/hello-world.html

GitHub Inc. a. About SSH. Viitattu 12.08.2024. Saatavissa https//docs.github.com/en/authentication/connecting-to-github-with-ssh/about-ssh

GitHub Inc. b. About GitHub and Git. Viitattu 06.07.2024 Saatavissa https//docs.github.com/en/get-started/start-your-journey/about-github-and-git

GitHub Inc. c. About commits. Viitattu 20.07.2024. Saatavissa https//docs.github.com/en/pull-requests/committing-changes-to-your-project/creating-and-editing-commits/about-commits

GitHub Inc. d. About pull requests. Viitattu 16.10.2024. Saatavissa: https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/about-pull-requests

Gills, A. 2023. MongoDB. Viitattu 08.07.2024. Saatavissa https//www.techtarget.com/searchdatamanagement/definition/MongoDB

Hamel, A., Keller, A. & Medeiros, L. 2024. File system. Viitattu 14.09.2024. Saatavissa https//nodejs.org/api/fs.html#file-system

Haverbeke, M. 2024a. Eloquent JavaScript. 4. E-kirja. San Francisco No Scratch press. Viitattu 03.07.2024. Saatavissa https://eloquentjavascript.net/00_intro.html

Haverbeke, M. 2024b. Eloquent JavaScript. 4. E-kirja. San Francisco No Scratch press. Viitattu 03.07.2024. Saatavissa https://eloquentjavascript.net/01_values.html

Hrácek, F. 2023. Building your first Flutter App – with Codelab! Video. Viitattu 05.08.2024. Saatavissa https//docs.flutter.dev/get-started/codelab

Hickson, I. 2024. SingleChildScrollView class. Viitattu 12.09.2024. Saatavissa https//api.flutter.dev/flutter/widgets/SingleChildScrollView-class.html

Hickson, I. & Jhonson, M. 2024. BoxDecoration class. Viitattu 13.09.2024. Saatavissa https//api.flutter.dev/flutter/painting/BoxDecoration-class.html

Hickson, I. & Williams, J. 2024. MemoryImage class. Viitattu 13.09.2024. Saatavissa https//api.flutter.dev/flutter/painting/MemoryImage-class.html

Javapoint. 2024a. How does JavaScript Work? Viitattu 03.10.2024. Saatavissa https//www.javatpoint.com/how-does-javascript-work

Javapoint 2024b. V-Model. Viitattu 11.10.2024. Saatavissa https//www.javatpoint.com/software-engineering-v-model

Javapoint 2024c. JavaScript Strict Mode. Viitattu 16.10.2024. Saatavissa https://www.javatpoint.com/javascript-strict-mode

Juhantalo, K. 2024. Osaamista monella sektorilla. Viitattu 18.09.2024. Saatavissa https//casamedia.fi/yritys/tarinamme

Lougheed, P. 2024a. Testing Flutter apps. Viitattu 15.07.2024. Saatavissa https//docs.flutter.dev/testing/overview

Lougheed, P. 2024b. Simple app state management. Viitattu 11.09.2024. Saatavissa https//docs.flutter.dev/data-and-backend/state-mgmt/simple

Lougheed, P. 2024c. An introduction to widget testing. Viitattu 11.10.2024. Saatavissa https//docs.flutter.dev/cookbook/testing/widget/introduction

McKee, A. 2023. Coding Best Practices and Guidelines for Better Code. Viitattu 16.09.2024. Saatavissa https//www.datacamp.com/tutorial/coding-best-practices-and-guidelines?dc_referrer=https%3A%2F%2Fwww.google.com%2F

pub.dev. readAsBytes method. Viitattu 13.09.2024. Saatavissa https//pub.dev/documentation/cross_file/latest/cross_file/XFile/readAsBytes.html

Quinlan, B. 2024. Http 1.2.2. Viitattu 10.09.2024. Saatavissa https//pub.dev/packages/http

Salvatierra, J. Day 21 Splitting Code Into Multiple Files. Viitattu 17.09.2024. Saatavissa https//teclado.com/30-days-of-python/python-30-day-21-multiple-files/

Shew, A. 2024. Difference between Node.js and the Browser. Viitattu 30.09.2024. Saatavissa https//nodejs.org/en/learn/getting-started/differences-between-nodejs-and-the-browser

Sufiyan T. 2024. What Is Node.js? A Complete Guide for Developers. Viitattu 09.07.2024. Saatavissa https//www.simplilearn.com/tutorials/nodejs-tutorial/what-is-nodejs

Swain, D. 2024. V Model in Software Testing | What, Why, Advantages and Disadvantages. Viitattu 01.10.2024. Saatavissa https//testsigma.com/blog/v-model-in-software-development-life-cycle/

Testsigma Technologies Inc. System Testing What is it, Why, Types & How to Perform. Viitattu 18.07.2024. Saatavissa https//testsigma.com/guides/system-testing/

University of Helsinki. 2022. Full Stack open. Viitattu 05.09.2024. Saatavissa https//fullstackopen.com/osa3/node_js_ja_express#express

Windmill, E. 2020. Flutter in Action. Shelter Island Manning.

Zakhour, S. & Lougheed, P. Visual Studio Code. Viitattu 07.06.2024. Saatavissa https//docs.flutter.dev/tools/vs-code

Liite 1. Oppimispäiväkirja

1.5.

Tavoite:

Tavoitteena on tutustua tarkemmin työprojektiin ja ottaa selvää sen tiedostorakenteesta. Projekti on jo aikaisemmin tuotu paikalliseen kansioon. Tavoitteena on myös totutella työskentelemään Flutter-kehyksen kanssa.

Tulos:

Flutter:in kohdalla keskityttiin lähinnä käyttöliittymän muokkaamiseen välittäjien avulla. Flutter-osaamisessa on vielä aukkoja, kuten käyttöliittymän visuaalinen asettelu.

Projektin yleinen rakenne on nyt tuttu ja se seuraa Flutter-projekteille tyypillistä rakennetta.

2.5.

Tavoite:

Tavoitteena on viedä orientaatiota eteenpäin. Keskitytään yksinomaan Flutter:in perusteisiin ja selkeän kokonaiskuvan luomiseen tällä kehyksellä työskentelystä. Pääpainona ovat käyttöliittymän visuaalinen asettelu ja muokkaus käyttäjän toimien perusteella.

Tulos:

Flutter:in toiminnallisuuksista on omaksuttu pääasiat. Varsinaiseen työskentelyyn vaadittava taitotaso on saavutettu.

3.5.

Tavoite:

Tehtävänä on selvittää miten Flutter:in kautta voisi lähettää viestejä WhatsApp-sovellukseen ja ottaa viestejä vastaan tämän sovelluksen kautta. Koska viikko on lopuillaan, on tavoitteena tutustua ongelmaan vain teoriassa.

Tulos:

Summittainen kuva mahdollisuuksista on luotu. Mahdollisuuksia on käytännössä kaksi. Toinen on helpompi toteuttaa, mutta sen hyödyntäminen saattaa rikkoa WhatsApp:in käyttäjäehtoja. Toinen taas vaatii enemmän työstämistä, mutta on ainakin teoriassa toteutettavissa sääntöjen mukaan. Jälkimmäiseen toteutukseen tarvitaan WhatsApp Business-käyttäjä. Tämä saattaa tuottaa ongelmia siinä suhteessa, että projektin sovelluksen on tarkoitus olla kaupallinen. Jos sen toiminta perustuu kolmannen osapuolen kaupalliseen ohjelmaan voi sen implementointi tuottaa juridisia ongelmia.

6.5.

Tavoite:

Lähettää viesti Flutter:in käyttöliittymästä WhatsApp-sovellukseen.

Tulos:

Viesti onnistuttiin lähettämään hyödyntämällä WhatsApp Cloud API:ta. Vaikka onkin täysin toimiva vaihtoehto, tämä toteutus vaatii toimiakseen erillisen WhatsApp Business-käyttäjän, jonka liittäminen toiseen sovellukseen voi tuottaa haasteita. Esimerkiksi, se miten viestinnässä tarvittavat tunnisteet siirretään business-käyttäjästä Flutter-sovellukseen, on vielä epävarmaa.

7.5.

Tavoite:

Ottaa WhatsApp-sovelluksesta lähetetty viesti vastaan Flutter-sovelluksella.

Päivän aikana tuli toinen tavoite muokata varsinaisen projektin kalenterisovellusta siten, että sen käyttöliittymä säilyy johdonmukaisena ruudun leveydestä huolimatta.

Tulos:

WhatsApp-viestittelyn toteutus on toistaiseksi pysähtynyt. Toteutuksen jatkamiseen tarvitaan erillinen käyttäjä, jonka voi luoda vain yrityksille, eikä se sovi haluttuun toteutukseen.

Projektin toiminnan kanssa tuli myös ongelmia. Päivän aikana ei onnistuttu kirjautumaan projektin sovellukseen testaamista varten. Tämä johtui siitä, että paikallinen .NET-SDK-tiedosto, jota projekti tarvitsee toimiakseen, oli vanhentunut. Tämän päivittämisen jälkeen saatettiin jatkaa töitä.

Kalenterin käyttöliittymä saatiin muuttumaan sen mukaan, kuinka leveä ruutu on käytössä. Tämä oli melko vaivatonta, sillä Flutter:issa on sisään rakennettu toiminto, joka voi palauttaa ruudun leveyden tai korkeuden muuttujana.

8.5.-13.5.

Tavoite:

Tavoite yllä mainituille viidelle päivälle oli korjata joitakin sovelluksen kalenterissa ilmeneviä virheitä ja tehdä asetuspäivityksiä. Esimerkiksi muuttaa päivämäärän rakennetta käyttöliittymässä käyttäjän toiveiden mukaan.

Tulos:

Kalenteri päivitettiin ja korjattiin onnistuneesti.

Dart-ohjelmointikieli yhdistettynä Flutter:iin on nyt tullut paremmin tutuksi. Nyt osataan lukea projektia tehokkaammin ja navigoida sen tiedostoja, kun etsitään jotain tiettyä koodista. Työskentelyn tahti on nopeutunut.

Myös ymmärrys siitä mitä käyttöliittymään on syytä laittaa, on kasvanut. Yleisesti ottaen on huomattu, että tekstiä kannattaa välttää. Jos viestintään voi käyttää symbolia tai väriä, niin se on yleensä parempi vaihtoehto. Tällä tavalla käyttöliittymä pysyy avarampana.

Suurin aikaa vievä asia työssä on jatkuva tarve etsiä esimerkkejä siitä, miten jokin Flutter:in valmis komponentti toimii. Niitä on kehyksessä lukuisia ja jokaista tarvitaan satunnaisesti. Näiden ohjelmien käyttämistä on yritetty saada jäämään muistiin, mutta niiden määrä tekee tämän vaikeaksi.

14.5.

Tavoite:

Luoda kalenterille ikkuna, jossa näkyy kaikki kalenteriin merkityt tapahtumat tapahtumisjärjestyksessä. Tähän ikkunaan tulee myös luoda hakukenttä ja sen rinnalle jonkinlaiset suodattimet, jotka rajaavat, mitä tapahtumia käyttäjälle näytetään.

Tulos:

Suurin osa työstä oli tehty jo aiemmin toisen ohjelmoijan toimesta. Projektissa oli käytössä valmis komponentti, joka loi listan tapahtumista. Tämä lista piti vain lisätä ikkunaan ja lisätä siihen hakukentät. Suurin ongelma tuli vastaan käyttöliittymän visuaalisen asettelun kanssa. Flutter on melko tarkka työkalu siitä, onko jokin osa käyttöliittymää liian leveä tai asetettu väärin. Tämän lisäksi sen esittämät virheviestit voivat olla melko epäselviä. Useimmissa tapauksissa täytyikin etsiä vastauksia internetistä, jotta voisi ymmärtää mitä mikin virheviesti tarkoittaa tai mihin se edes liittyy. Ei myöskään tiedetä voisiko virheviestejä suodattaa jotenkin. Yhteen virheviestiin liittyy liuta eri epäkohtia useissa eri tiedostoissa, jotka ovat useimmiten osa jotain valmista kirjastoa. Nämä ilmoitukset ovat lähes poikkeuksetta hyödyttömiä, sillä valmiiden kirjastojen koodia ei tulla koskaan muokkaamaan. Silti ilmoituskenttä täyttyy viesteistä osoittaen kymmeniin tiedostoihin, joihin ei tulla koskaan koskemaan.

15.5.

Tavoite:

Ei merkintää

Tulos:

On opittu käyttämään välittäjiä paremmin käytännössä. Tuli ilmi, että Flutter:issa välittäjiä voi hyödyntää myös kuluttajien ulkopuolella. Tämä tekee välittäjien hyödyntämisestä huomattavasti helpompaa, etenkin pienemmissä komponenteissa. Useimmissa tapauksissa komponenttien käyttöliittymää käsittelevän koodin sisälle ei voi kirjoittaa suoraan muita kuin muita komponentteja tai niiden sisäisiä parametrejä. Tämä tuottaisi paljonkin ongelmia, jos kuluttaja ohjelma olisi ainoa tapa päästä käsiksi välittäjän tarjoamaan dataan. Kuluttaja kun pitää kirjoittaa osaksi käyttöliittymäkoodia. Mutta nyt kun osataan kirjoittaa välittäjäkoodia käyttöliittymäkoodin ulkopuolella, niin välittäjien käyttö ja siten uusien komponenttien käyttö helpottuu huomattavasti.

16.5.

Tavoite:

Yhdistää kalenterin tapahtumien kategoriat tietokantaan ja korjata kategorioiden poistamisesta johtuva virhe. Kun kategorian poistaa, tämä muutos ei näyt reaaliaikaisesti kaikissa käyttöliittymän osissa.

Tulos:

Koska projektin tietokannan käsittely oli vielä tuntematon, tämä tehtävä vei jokseenkin kauan. Tietokanta hyödyntää MongoDB-palvelua, jossa käyttäjä voi luoda itselleen oman tietokantansa ja yhdistää siihen erillisillä tunnuksilla. Tähän projekti hyödyntää NodeJs-ajoympäristöä ja niin ikään express-kehystä. Molemmat näistä ovat entuudestaan tuttuja, joten tietokannan käsittelyohjelmiston rakenne on jokseenkin tuttu. Jotta tietokantaan tehtäviä muutoksia saatettiin testata, piti tehdä henkilökohtainen käyttäjä MongoDB-palveluun. Tästä oli annettu erilliset ohjeet projektin tiedoissa, joten tämä ei ollut muuta kuin aikaa vievää.

Kategorioiden poistamiseen liittyvää ongelmaa ei saatu ratkaistua. Ongelma todennäköisesti on se, että kategorioiden välittäjä ei ole yhdistetty oikein, joten kun kategoria poistetaan, niin tämä muutos ei välity käyttöliittymän muihin osiin. Vielä ei olla varmoja, miten tämä korjattaisiin.

22.5.

Tavoite:

Ei merkintää

Tulos:

Projektin aikana on kohdattu pääosin kaksi tapaa kirjoittaa Flutter-pohjaista ohjelmaa. Joko kirjoittaa niin sanotusti puhtaita komponentti tai sitten kirjoittaa funktioita, jotka palauttavat komponentin. Kummassakin tyylissä on hyvät ja huonot puolensa. Jälkimmäisessä eli niin sanotusti funktiovetoisessa tyylissä on se etu, että käyttöliittymän koodia ja muuta ohjelmistologiikkaa voi melko vapaasti kirjoittaa sekaisin, kun taas puhtaissa komponenteissa tämä ei ole yhtä helppoa. Puhtaissa komponenteissa a taas tilojen hallinta on huomattavasti helpompaa.

On huomattu, että ohjelmistoarkkitehtuuri kaipaisi harjoittelua. Monessa tapauksessa tiedostojen rivimäärä kasvaa liian suureksi ja tiedoston navigointi käy työlääksi. Pitäisi tutustua tähän

tarkemmin ja yrittää löytää jonkinlainen ohje sille, miten ohjelmiston eri osat kannattaa jakaa eri tiedostoihin.

24.5.

Tavoite:

Luoda sovelluksen käyttäjille tiimimuuttuja, johon voi asettaa haluamansa määrän käyttäjiä. Tälle tulee luoda myös käyttöliittymä ja yhdistää se tietokantaan. Tiimillä tulee myös olla tiettyjä oikeusmuuttujia, jotka välittyvät tiimin jäsenille.

Tulos:

Tavoitteen saavuttaminen ei vaatinut minkään uuden tiedon löytämistä. Ainoa mikä löydettiin, oli vastaus siihen, miksi joitakin välittäjiä ei käytetä kuluttajien kautta. Jos jokin komponentti ei tarvitse jatkuvaa päivittämistä välittäjän välittämästä datasta, niin siinä tapauksessa voi olla perusteltua käsitellä välittäjää vain muuttujana eikä kuluttajan kautta.

Tänään kohdattiin myös uusi ongelma tietokannassa. Vielä ei olla varma, miten useamman olion päivittäminen tai lisääminen tulisi hoitaa yhdellä API-kutsulla. Yksittäisen olion lähettäminen ei ole ongelma, mutta useamman olion lähettäminen ja ennen kaikkea purkaminen toisessa päässä tuottaa ongelmia. Jostakin syystä objektin ID-muuttujan purkaminen tuottaa eniten vaikeuksia.

4.6.

Tavoite:

Muokata eilen luodusta käyttöliittymästä käyttäjäystävällisempi. Käyttöliittymän listoissa olevat objektit esittävät liikaa tietoa käyttäjälle kerralla, joka vie likaa tilaa ruudulla. Tarkoitus on pienentää listoissa näkyviä objekteja ja lisätä niihin laajennusominaisuus, jos käyttäjä haluaa tarkastella niitä tarkemmin. Tarkoituksena on myös luoda huoltotoimenpiteitä kuvaaville objekteille jonkinlainen kommenttikenttä, johon huoltoja tekevät henkilöt voivat lisätä huomioita.

Tulos:

Huomioiden lisääminen onnistui. Ongelmana oli, että yhden huollon huomiot näkyivät kaikissa muissa samanlaisissa huolloissa. Tämä ongelma ratkaistiin asettamalla huoltojen listat Map-olion sisään. Avaimena toimi huolto-ohjelman id-muuttuja. Tällä tavalla huolto saattoi tunnistaa huolto-ohjelman ja antaa huomioita oikeasta huolto-ohjelmasta. Ongelmana on vielä käyttöliittymän yhdistäminen tietokantaan. Nykyisellä tyylillä Map-olioiden ajaminen JSON:in läpi tuottaa haasteita. Varsinkin kun Map-olion arvot ovat toisia olioita. Seuraavan päivän tavoitteena olisikin alustavasti saada vastaus tähän ongelmaan, jotta kommentit voitaisiin tallentaa.

5.6.

Tavoite:

Yhdistää huoltoihin liitetyt kommentit tietokantaan. Tavoitteena on myös muuttaa huoltojen "tehty"-tila sellaiseksi, että jos sama huolto on useassa huolto-ohjelmassa, niillä kaikilla on oma

"tehty"-tilansa. "Tehty"-tilan on tarkoitus kertoa, jos mikäkin huolto on tehty vai ei. Tavoitteena on myös yhdistää tämä tietokantaan.

Tulos:

Huoltojen kommentit saatiin yhdistettyä tietokantaan. Tämä paljastui melko haasteelliseksi aluksi, koska ei tiedetty, miten Map-olioita siirretään JSON:in kautta. Tähän kuitenkin löydettiin ratkaisu.

Huoltojen "tehty"-tilan päivityksiä ei saatu valmiiksi. Ensimmäinen sovellutus epäonnistui siksi, koska "tehty"-tila vaatii toimiakseen huolto-ohjelman sarjanumeron. Tätä sarjanumeroa huolto-ohjelmalla ei kuitenkaan ole, ennen kuin se on tallennettu tietokantaan. Tämä johtuu siitä, koska MongoDB luo tietokannan olioiden sarjanumerot automaattisesti. Siitä huolimatta "tehty"-tilan kanssa tulee olla mahdollista vuoro vaikuttaa ennen kuin huolto-ohjelma tallennetaan ensimmäistä kertaa tietokantaan. Tämän ongelman ratkaiseminen tulee olemaan seuraavan päivän ensimmäinen tavoite.

6.6.

Tavoite:

Saada "tehty"-tila eriäväksi jokaisen huolto-ohjelman huollossa. Sama huolto voi olla useammassa huolto-ohjelmassa, joten huollon on tunnistettava eri huolto-ohjelmat ja ilmaistava "tehty"-tilansa huolto-ohjelmansa mukaan. Toinen tavoite on lisätä resurssioliolle uusia muuttujia ja yhdistää ne käyttöliittymään ja myöhemmin tietokantaan.

Tulos:

"Tehty"-tila saatiin toimivaksi ja yhdistettyä tietokantaan. Tämä onnistui siten, että "tehty"-tila yhdistetään tietokantaan vasta sitten, kun käyttäjä vaihtaa sen ensimmäisen kerran tai kun käyttäjä tallentaa huolto-ohjelman ensimmäisen kerran. Resurssien muuttujapäivitys jäi suppeaksi. Pääpuoli päivästä kului resurssien käyttöliittymän päivittämiseen päivitystä varten. Entinen käyttöliittymä ei antanut suuria mahdollisuuksia laajentaa resurssien ominaisuuksia tekemättä käyttöliittymästä sotkuista. Entisessä käyttöliittymässä saattoi lisätä useita resursseja kerralla ja päivittää niitä suoraan listasta, mutta tämä lähestymistapa vaatii sitä, että kaikki resurssien tiedot ovat listassa näkyvillä. Tämä taas täyttää ruutua liikaa resurssien ominaisuuksien kasvaessa. Tämän takia katsottiin hyväksi luopua vanhasta käyttöliittymästä. Resurssille luotiin lomake, jota täyttämällä käyttäjä voi syöttää haluamansa tiedot resurssiin sitä luodessaan. Tämän lisäksi suurin osa resurssien tiedoista siirrettiin listasta erilliseen ponnahdusikkunaan, joka ilmestyy, kun listan resurssia klikkaa. Tämä antaa enemmän tilaa muuttujille luontivaiheessa ja enemmän joustovaraa tulevia päivityksiä ajatellen.

Parannettavaa:

Työntekoa hidastaa nykyinen tyyli tehdä jokainen päivitys kauttaaltaan loppuun. Ensin tehdään yksinkertainen käyttöliittymä ja sitten sen tuoma lisäys yhdistetään tietokantaan. Voisi olla tehokkaampaa ensin tehdä kaikki muutokset käyttöliittymään, jonkinlaisen nukkemuuttujan turvin, joka säilöö käyttöliittymässä käytetyt muuttujat. Vasta kun käyttöliittymään on tehty koko päivitys, niin sen jälkeen se yhdistettäisiin tietokantaan. Tämä voisi suoraviivaistaa työprosessia, kun ei tarvitse keskittyä kuin yhteen asiaan kerrallaan, eikä vaihtaa nopeaan tahtiin käyttöliittymän ja tietokannan käsittelyn välillä.

On myös huomattu, että samanlaista koodia tarvitaan usein monessa paikassa. Pitäisi painottaa enemmän valmiiden funktioiden tai luokkien uudelleen käyttämiseen kuin samantapaisen koodin kirjoittamiseen uudelleen ja uudelleen. Esimerkkinä, monessa ikkunassa käytetty yläkulman sulkemispainike on nyt oma erillinen komponenttinsa, jota voi kutsua, kun tuon painikkeen haluaa. Tämän kaltaisia yleisiä käyttöliittymän palasia tulisi standardoida enemmän.

10.6.

Tavoite:

Ensimmäisenä on tarkoitus viedä resurssien käyttöliittymän päivitys loppuun. Kun resurssia klikkaa, on tarkoitus avautua uusi ikkuna, jossa näkyy resurssin tarkemmat tiedot. Toinen tavoite on päivittää huoltojen käyttöliittymää. Huoltajia tulee olla mahdollista lisätä joko vapaana tekstinä tai valita erillisestä valikosta.

Tulos:

Resurssien käyttöliittymä saatiin toimimaan tarkoitusten mukaisesti. Visuaalisessa puolessa voi vielä olla hiomista, mutta se näyttää sen informaation mikä on tarkoitus.

Huoltajien lisääminen vaati suuremman päivityksen kuin vain käyttöliittymämuutoksen. Huollon tekijöitä voi tällä hetkellä valita joko kontaktilistasta tai kirjoittaa tekstikenttään. Jotta valikosta valitut huoltajat eivät menisi sekaisin kirjoitettujen kanssa, oli ne eriteltävä uuteen listaan. Tämä vaati päivityksen myös tietokannan toimintaan. Myös käyttöliittymä tuotti ongelmia. Kahden eri listan perusteella on vaikeaa luoda saumatonta listaa käyttöliittymään. Tämä ongelma ratkesi niin sanotusti sitomalla kirjoitettujen ja valittujen huoltajien listat toisiinsa. Siinä missä on kirjoitettu huoltaja yhdessä listassa, on valitussa listassa vain tyhjä kohta. Tällä tavalla voidaan käyttöliittymää luodessa käyttää hyödyksi vain yhden listan tietoja ja sen perusteella luoda todellisuutta vastaava lista käyttöliittymään.

Tämä tosin luo listaan turhaa dataa ja vaatii toimiakseen sen, että käyttäjä antaa aina jonkin nimen huoltajalle tai valitsee huoltajan, jos mahdollista. Muussa tapauksessa listaan voi ilmestyä täysin tyhjiä kohtia myös käyttöliittymään ja siten aiheuttaa hämmennystä. Tämä ei kuitenkaan ole ratkaisematon ongelma. Käyttäjää pitää vain estää tallentamasta huoltoa, jos syötteessä on tyhjä kohta.

11.6.

Tavoite:

Tavoitteena on luoda huollolle muuttuja, mikä osoittaa, jos huollon on tehnyt alihankkija tai ei. Tämä on muuttuja, jonka on tarkoitus näkyä vain, kun huolto on huolto-ohjelmassa. Toinen tavoite on luoda huolto-ohjelmille jonkinlainen hakukenttä tai useita hakukenttiä erilaisille hauille. Esimerkkinä, tavanomainen tekstikenttä, joka hakee nimen tai kuvauksen perusteella tai valikko, josta voi valita huolto-ohjelman tilan, jolloin näkyy vain tuon tilan huolto-ohjelmat. Tarkoituksena on myös hyödyntää ulkoisen ohjelmakirjaston taulukkoa huolto-ohjelmien visualisoimiseen.

Tulos:

Huollon uusi muuttuja ei tuottanut ongelmia. Samantapainen sovellutus oli tehty aiemmin, joten samaa koodia pystyttiin käyttämään uudelleen. Hakukenttäkin oli tuttua työtä. Ainoa mikä tuotti hieman vaikeuksia, oli listamuotoinen hakukenttä, mutta lähinnä siinä suhteessa, että Flutter:in pudotuslistat ovat hieman vieraita ja melko tarkkoja siitä, miten niihin syötetään muuttujia.

Huolto-ohjelmille tarkoitettu taulukko oli myös melko vaivatonta asentaa ohjelmaan. Koska suurin osa taulukosta löytyi valmiina Flutter:in kirjastosta, ainoaksi ongelmaksi jäi taulukon yhdistäminen muun ohjelman logiikkaan. Taulukossa näytettävillä olioilla toisaalta oli yhteensopivat muuttujat taulukon kanssa, joten kyse oli vain olioiden lataamisesta oikeaan paikkaan. Taulukosta saatiin valmiiksi raakaversio.

Parannettavaa:

Joissain paikoissa on huomattu, että projektia vaivaa melko suuret tiedostojen rivimäärät. Tämä hidastaa tiedostojen muokkausta, kun haluamiaan funktioita on vaikea löytää joskus yli kahdentuhannen rivin tiedostoista. On yritetty korjata tätä ongelmaa pilkkomalla joitain tiedostoja useammaksi tiedostoksi, mutta tämä on tuonut usein uusia ongelmia. Jotkin toteutukset ovat olleet hyvin hankalia jakaa toisiin tiedostoihin.

Tänä päivänä yritettiin jakaa yksi melko suuri tiedosto kahteen pienempään tiedostoon. Yritys onnistui mutta koska jaettava osa tiedostosta oli ohjelmoitu kiinteäksi osaksi suurempaa tiedostoa, niin sen siirtäminen vaati paljon ylimääräistä ohjelmoimista. Tältä olisi voitu välttyä, jos olisi tutustuttu tarkemmin ohjelman rakenteeseen. Tulevaisuudessa tuleekin ottaa huomioon työympäristön tuomat rajoitukset ja punnita tehtävät muutokset tarkemmin, jos niitä ei ole erikseen pyydetty. Tulevaisuudessa tulee kuitenkin kirjoittaa koodi siten, että se on jo jaettu tai ainakin jaettavissa pienemmiksi tiedostoiksi, vaikkakin tiedostojen määrä täten kasvaisi.

12.6.

Tavoite:

Tavoite on viedä edellisenä päivänä luotua taulukkoa eteenpäin. Aikajanaa on oltava mahdollisuus muuttaa tarkkuudessa, joko viikkojen tai kuukausien tarkkuuteen, sekä aloitus ja lopetuspäivämäärä tulee olla muokattavissa. Taulukko kaipaa myös ulkoasun päivitystä. Edellisen päivän huolto-ohjelmien hakusysteemi sai myös uusia vaatimuksia. Se täytyy muokata hakemaan huolto-ohjelmia myös tehtyjen kommenttien ja yksittäisten huoltojen- tai huoltojen välivaiheiden perusteella.

Tulos:

Tavoitteisiin päästiin. Toisaalta kummankin toteutuksessa on parantamisen varaa. Hakusysteemi kirjoitettiin hankalalukuisesti ja siinä on paljon for-silmukoita sisäkäin, joka voi olla koneelle raskasta. Varsinkin jos hakusysteemi seuloo suurempaa määrää tietoa.

Huolto-ohjelmien taulukon päivitys saatiin tehtyä. Taulukon tarkkuuden vaihtaminen päivien, viikkojen ja kuukausien välillä tuotti hieman hankaluuksia. Niille ei ollut taulukon kirjastossa valmista pohjaa vaan taulukkoa piti manuaalisesti muokata. Tämä hajotti taulukon ulkoasua ja rakennetta jonkin verran. Vielä ei olla saatu selville, voiko tätä toiminnallisuutta toteuttaa ilman taulukon hajoamista.

13.6.

Tavoite:

Tehdä rästiin kertyneitä tehtäviä. Tarkoituksena on luoda tapa, jolla saadaan muutettua huolto- tai huolto-ohjelma PDF-tiedostoksi. Tämän lisäksi huolto-ohjelmat on voitava lisätä kaleenteriin ja resurssien luontilomakkeeseen pitää lisätä otsikko ja virheviesti laittaa näkyviin vasta, jos käyttäjä yrittää tallentaa virheellisen resurssin. Tällä hetkellä virheviesti näkyy niin kauan, kun vaadittavat kentät ovat tyhjiä.

Tulos:

Resurssien lisäämislomakkeen päivitys onnistui ilman ongelmia. Huolto-ohjelmien ja myöhemmin resurssien lisääminen kalenteriin olisi voinut olla suuri urakka, ellei olisi keksitty muuntaa huolto-ohjelmat ja resurssit hetkellisesti kalenteritapahtumiksi. Kaikilla näillä on yhteisiä muuttujia, kuten nimi ja päivämäärät. Tällä tavalla saatettiin käyttää hyväksi jo olemassa olevaa koodia, ja yhdistää huolto-ohjelmat ja resurssit kalenteriin jokseenkin saumattomasti. Kuitenkin on huomioitava, miten kalenteritapahtumien tallentaminen hoidetaan, jos siihen tehdään muutoksia. Mahdollisuus sille, että huolto-ohjelma tai resurssi tallentuisi tietokantaan tapahtumana pitää yrittää minimoida.

Huolto-ohjelmien PDF-konversio jäi kesken. Lähinnä saatiin aikaiseksi konversiopainikkeiden lisääminen käyttöliittymään ja konversioon tarvittavien luokkien ja funktioiden raakaversiot. Nämä funktiot eivät vielä tee muuta, kuin lataavat tyhjän pdf-tiedoston huolto-ohjelman nimellä.

Huomattiin, että tietokantaa päivittäessä on otettava huomioon jo tietokannassa oleva data. Tämä data ei päivity, kun tietokannan toimintaa päivitetään ja tämä aiheuttaa herkästi virheviestejä. Näin kehitysvaiheessa sillä ei usein ole suurtakaan väliä, mutta tulevaisuutta ajatellen, jos tietokannassa olisikin tuhansia objekteja ja ne kaikki pitäisi päivittää tietokannan mukana, olisi tämä todennäköisesti hyvin kuormittavaa prosessi.

14.6.

Tavoite:

Viedä loppuun huolto-ohjelmien muuttaminen pdf-tiedostoiksi. Jos tästä jää aikaa, niin aiotaan myös korjata testeissä huomattuja virheitä, joko käyttöliittymän asettelussa tai sen toiminnassa.

Tulos:

Huolto-ohjelman muuttaminen pdf-tiedostoksi on tehty Flutter:issa melko suoraviivaiseksi. Pdf kasataan samalla tavalla kuin mikä tahansa käyttöliittymä kehyksessä. Toisaalta, kuten minkä tahansa käyttöliittymän muokkaaminen, jossa vaaditaan sovelluksen uudelleen käynnistys tulosten katsomiseen, tämäkin oli hyvin aikaa vievää. Suurin osa päivästä kuluikin tähän. Toista tavoitetta saatiin tehtyä hieman. Yhden käyttöliittymän valikkoa muokattiin siten, että alas putoavasta valikosta siirryttiin pop-up-valikkoon. Tämä siitä syystä, koska alas putoavasta valikosta voi olla työlästä etsiä yksittäistä objektia, jos niitä on tuhansia.

Koska tällaista listaa näyttävää pop-up-valikkoa tarvitaan todella monessa paikassa, päätettiin luoda universaali valikko, mille tahansa objektijoukolle. Näillä toki pitää olla jotain yhdistäviä muuttujia, jotta ne voidaan esittää järkevästi. Tämä onnistui Dart-kielen dynaamisella datatyypillä, joka voi esittää mitä tahansa muuta datatyyppiä.

17.6.

Tavoite:

Korjata aiemmissa tuotoksissa ilmenneitä virheitä, kuten käyttöliittymän tekstin vääristyminen, kun tekstin pituus ylittää tietyn merkkimäärän, tai turhan API-kutsun tekeminen, kun jokin sivu avataan.

Tulos:

Useimmat virheet saatiin korjattua. Yksi päivitys ei onnistunut, jossa yritettiin saada aikaan funktiota, joka järjestäisi listan oliot aakkosjärjestykseen. Päivitysten tekemien kaatui käyttöliittymän arkkitehtuuriin. Se miten käyttöliittymä hoitaa tilan muutoksia, on lähes yhteensopimaton kaavailtujen päivitysten kanssa. Esimerkiksi jos komponentti ottaa muuttujansa suoraan välittäjän listoista, tuomatta koko listaa ensin omaan käyttöönsä, on tuon listan manipuloiminen huomattavasti hankalampaa. Tämä oli suurin este erinäisten listojen järjestämiseen aakkosjärjestykseen. Toisaalta olisi voinut olla mahdollista järjestää muuttujat jo välittäjässä, kun ne on haettu tietokannasta.

On myös tullut vastaan käyttöliittymän virhe. Vaikka ohjelma sinänsä toimii täysin moitteettomasti, niin käyttöliittymä ei jostain syystä heijasta näitä muutoksia oikein. Tämä siitä huolimatta, että välittäjät ja kuluttajat on asennettu niin kuin pitää. Vielä ei tiedetä mistä tämä johtuu, mutta on todennäköistä, että kyseessä on useammasta välittäjästä tai vastaanottajien päällekkäisyyksistä, joka aiheuttaa sekaannusta.

18.6.

Tavoite:

Korjata huoltoja ja resursseja vaivaavia käyttöliittymävirheitä. Resurssit on myös saatava aakkosjärjestykseen.

Tulos:

Resurssit saatiin aakkosjärjestykseen. On myös kehitetty funktio, joka voi järjestää lähes mitä tahansa olioita aakkosjärjestykseen, jos niillä on nimi tai etunimi. Tämän pitäisi tehdä aakkostamisesta tulevaisuudessa huomattavasti helpompaa.

Suurin osa tämän päivän työtunneista menivät käyttöliittymän muokkaamiseen siten, että se toimisi myös kapeilla näytöillä. Tämä useammassa tapauksessa tuntui vaativan paljon samanlaisen koodin toistamista, sillä kun ruutu kapenee, on yleensä parasta muuttaa rivissä olevat objektit jonoksi. Tämä tuntui aluksi vaativan lähes jokaisen rivin sisällä olevan komponentin kirjoittamista kahdesti ja huomattiin nopeasti, että piti kehittää parempi tapa. Huomattiin, että komponentteja ei itsejään tarvinnut muuttaa useimmissa tapauksista ollenkaan, joten kyse oli vain niiden siirtämistä rivistä jonoon. Kosta rivit ja jonot ottavat vastaan listan komponentteja, saatettiin näiden ajatusten pohjalta luoda funktio, joka ottaa vastaan listan komponentteja ja asettaa ne jonoon tai riviin tarpeen mukaan.

19.6.

Tavoite:

Lisätä resurssien tietoikkunaan mahdollisuus muokata resurssia. Toisena on ottaa selvää, miten käyttäjäkohtaiset kalenterit voi synkronoida googlen kalenterin kanssa.

Tulos:

Resurssien tietoikkunan päivitys ei ollut kovin monimutkainen. Ainoa mikä aiheutti hankaluuksia, oli kalenterin resurssit, joiden logiikkaa piti muuttaa tavallisten resurssien mukana.

Toinen tavoite muuttui päivän aikana. Tavoitteeksi tuli etsiä keino tallentaa käyttäjän lataama kuva jotenkin. Kuvia ei ole hyvä tallettaa tietokantaan sellaisenaan, koska ne vievät paljon tilaa, joten on löydettävä jokin muu tapa tallettaa kuvat. Tämänhetkinen tapa mitä yritetään, on kuvien lokaali tallentaminen jonkinlaiseen kansiojärjestelmään ja vain noiden kuvien polkujen tallentaminen tietokantaan. Tämän järjestelmän mahdollisuutta ei vielä tiedetä, mutta on onnistuttu lataamaan kuva käyttöliittymään ja esittämään kuva tuossa käyttöliittymässä.

Tämä oli odotettua monimutkaisempaa. Flutter:issa on eri funktiot eri alustoille kuvan latausta varten. Oikean löytäminen vei aikaa, varsinkin kun väärät tavat ohjelmoida kuvan lataaja opittiin kantapään kautta.

Tällä hetkellä on kuitenkin mahdollista saada kuva käyttäjän koneelta muuttujaksi ohjelmaan. Nyt täytyy enää selvittää, miten tuo muuttuja saadaan tallennettua toisen koneen kansioon.

20.6.

Tavoite:

Tutkia mahdollisuuksia tallentaa kuvia, suurissa määrissä. Kuvien tallentaminen suoraan tietokantaa ei ole suositeltavaa, koska kuvatiedostot vievät paljon tilaa. Tästä syystä tulisi, joko keksiä keino kutistaa kuvia, jotta tallettaminen helpottuisi tai löytää sopiva paikka, jonne kuvat voitaisiin tallentaa, ja tallettaa tietokantaan vain osoite kuvan olinpaikkaan.

Tulos:

Kuvien koon muokkaaminen on melko rajallista, jos kuvassa on useita eri pikseliväreji tai sävyjä. Suuria määriä samoja pikseleitä voi tiivistää melko helposti, jos kuvatiedoston muuttaa ensin sarjaksi numeroita tai kirjaimia. Jos samoja kirjaimia on monta jonossa, niin voi tuon jonon korvata yhdellä kirjaimella ja numerolla, joka kuvastaa kirjainjonon pituutta. Tämän voi sitten toisella funktiolla purkaa, ja todellinen kuva on valmis.

Toinen vaihtoehto on muuttaa kaikki tai suurin osa kuvista webp-muotoon, joka on tiedostokooltaan noin kolmanneksen pienempi muihin kuvatiedostoihin verrattuna. Flutter kykenee käsittelemään webp-tiedostoja moitteitta, mutta tällä hetkellä käytössä olevalla käyttöjärjestelmällä ei ole yhteensopivaa muuttamaan muita kuvatyyppejä webp-kuviksi.

Tällä hetkellä näyttää siltä, että paras vaihtoehto olisi tallettaa kuvat johonkin tietopankkiin ja tallettaa tuon tietopankin osoitteet tietokantaan ja vain referoida kuvia tuon pankin kautta. Tosin tämä nostaa kysymyksen siitä, miten tällainen palvelu kehitetään turvautumatta kolmannen osapuolen ohjelmistoon.

24.6.

Tavoite:

Selvittää onko JavaScript:illä mahdollista muuttaa jpg- tai png-kuvatiedostoja webp-kuvatiedostoiksi. Toisena tavoitteena on luoda yksinkertaisille, kuville pakkausmenetelmä.

Uutena tavoitteena on luoda jonkinlainen tapa tallentaa kuvia paikalliseen kansioon, joka on erillään muusta sovelluksesta.

Tulos:

Tiedostojen muuntaminen jpeg-muodosta webp-muotoon ei onnistunut. Suunniteltu pakkausmenetelmä ei myöskään toiminut png-tiedostojen kanssa, joten molemmat ovat tällä hetkellä jäässä. Uusi tavoite sai tänä päivänä suurimman huomion ja aikaan saatiin karkea prototyyppi, jolla voi tallentaa kuvatiedostoja tekstitiedostoiksi. Tällä hetkellä pyritään luomaan se siten, että samaan tekstitiedostoon voisi tallentaa useita kuvia. Tämä tosin vaatii ylimääräistä tunnistamislogiikkaa.

25.6.

Tavoite:

Tavoitteena on saada toimimaan edellisenä päivänä aloitettu systeemi, jolla voi tallentaa kuvia tekstitiedostoiksi. Tällä hetkellä joko kuva tai kuvaan asetettava sarjanumero ei tallennu oikein. Tämän lisäksi tavoitteena on muokata kuvista käyttöliittymän puolella yhteensopivia tämän järjestelmän kanssa.

Tulos:

Systeemillä voi tallentaa kuvia tekstitiedostoihin ja myöhemmin ladata niitä käyttöliittymän käyttöön. Tämä on tosin vain mahdollista sovelluksen profiilikuvilla. Tavoitteeseen pääseminen vaati tiedostomanipulaation opettelua JavaScript-kielellä ja sen mukaa fs-kirjaston käyttöä. Tämä oli suhteellisen suoraviivaista, mutta aiheutti siitä huolimatta ongelmia, sillä jouduttiin käyttämään useita asynkronisia funktioita, joka usein johti virheisiin. Tämän tyyppiset virheet voivat olla hankalasti huomattavissa, koska ne eivät suoraan anna virheviestiä, vaan saattavat vain pysäyttää ohjelman toiminnan tai aiheuttaa loputtomia silmukoita.

26.6.

Tavoite:

Viimeistellä kuvien tallentamisjärjestelmä. Järjestelmään täytyy lisätä kansiot jokaiselle käyttäjälle ja huolehtia kuvien tai kansioiden poistamisesta. Uutena tavoitteena yhdistää laskutusjärjestelmään logo ja yhdistää se tallennusjärjestelmään.

Tulos:

Kuvien tallennusjärjestelmä saatiin toimimaan. Kuvat tallennetaan tekstitiedostoihin ja ne puretaan sen mukaan, kun käyttöliittymästä tulee kutsuja. Jotta kuvien koko pysyisi alhaisena ja tilaa säästyisi, tehtiin erinäisiä pakkausfunktioita. Esimerkiksi png-tiedoston muuttaminen jpg-

tiedostoksi ja jpg-tiedostojen laadun alentaminen, kun kuva on ladattu. Nämä funktiot mahdollistavat myös sen, että ladattujen kuvien kokoa ja tiedostotyyppiä voidaan monitoroida jatkossa suhteellisen helposti.

Varsinainen tallennusjärjestelmä on vielä prototyyppi. Se on yhdistetty suoraan tietokannan käsittelyohjelmaan ja se tallentaa kuvat tämän ohjelman kansioihin. Tähän ratkaisuun tyydytään toistaiseksi, kunnes saadaan selville, minne kuvat voitaisiin tallentaa. Järjestelmän rakentama tiedostopuu on myös vielä alkeellinen, mutta vielä ei tiedetä, minkälainen tiedostorakenne olisi paras. Tämänhetkinen ratkaisu hyödyntää jokaiselle käyttäjälle luotua uniikkia sarjanumeroa ja yksinkertaista tiedostonimeä tiedoston tunnistamiseen. Enne kuin saadaan varmuus halutusta tiedostoarkkitehtuurista niin kuvien tallennusjärjestelmän kehitys on jäässä.

Logon lisääminen laskutusjärjestelmään ei tuottanut hankaluuksia. Sama komponentti, jota käytetään profiilikuvassa, toimii yhtä hyvin logon asettamiseen.

27.6.

Tavoite:

Kuvien tallennusjärjestelmän ollessa jäässä, on aikaa korjata joitakin ohjelman virheitä. Virheet kohdistuvat huoltoihin ja huolto-ohjelmiin. Esimerkiksi, kun huollon poistaa, niin se ei poistu huolto-ohjelmasta ja aiheuttaa virheviestin. Sama tapahtuu huoltojen kommenttien kanssa. Tämä ei sinänsä aiheuta virhettä, mutta vie turhaa tilaa muistista.

Tulos:

Virheiden korjaus onnistui suurimmalta osin. Huoltojen ja resurssien poistamisesta aiheutuvat virheviestit saatiin loppumaan lisäämällä resurssit ja huollot huolto-ohjelmasta poistava funktio yksittäisen huollon tai resurssin poistologiikkaan. Tätä ennen yritettiin muitakin vaihtoehtoisia järjestelmiä, kuten vielä olemassa olevien huoltojen tai resurssien tarkastusta, kun huolto-ohjelma näkyy ruudulla tai ladataan. Näillä järjestelmillä ei jatkettu, koska niillä oli vaikea saada pysyviä muutoksia huolto-ohjelmiin. Niin sanottuja haamukommentteja ei saatu korjattu.

28.6.

Tavoite:

Saada kommentit poistumaan huolloista, kun niihin liitetty huolto-ohjelma poistetaan. Pyritään myös lisäämään poistomahdollisuus kommentteihin.

Tulos:

Kommenttien poisto huolloista huolto-ohjelman poistuessa saatiin toimimaan samalla kaavalla kuin edellisen päivän ongelmat. Kun huolto-ohjelma poistetaan, se kutsuu erillistä funktiota, joka huolto-ohjelman tietojen mukaan poistaa huolloista oikeat kommentit.

Yksittäisten huoltojen poistaminen manuaalisesti saatiin myös toimimaan. Ainoa asia mikä aiheutti hankaluuksia, oli huolto-ohjelmien käyttöliittymään muodostunut hieman sekava kuluttajaohjelmien verkosto. Saman kuluttajaohjelman seuraamaa välittäjäohjelmaa välitettiin funktiosta funktioon yhä syvemmälle. Jostain syystä tämä esti kommenttien ponnahdusikkunan päivittymisen ja sille jouduttiin lisäämään oma kuluttajaohjelma. Viimeistään tässä vaiheessa tulisi ottaa

selvää, miten kuluttajien ja etenkin välittäjien kanssa tulisi toimia, jotta ohjelma päivittyisi oikein eikä syntyisi liikaa turhia välittäjäohjelmamuuttujia. Kun kommenttien poistaminen oli saatu toimivaksi, niin huomattiin, että vaikka kaikki kommentit olisi poistettu jää noita kommentteja säilönyt kartta-olio viemään tilaa tietokannassa. Tämän ratkaiseminen ei sinänsä ollut vaikeaa, mutta se aiheutti useita virheviestejä, koska oletusarvoisesti kommenttien ponnahdusikkuna ei voisi olla olemassa ilman kommentteja.

Näiden tavoitteiden saavuttamisen jälkeen, aikaa oli sen verran vähän, että loppu päivästä kului käyttöliittymän kääntämättömien sanojen yhdistämiseen käännösjärjestelmään. Tämä mahdollistaa niiden näkymisen oikealla kielellä, kun sovelluksen kieliasetuksia muuttaa.

1.7.

Tavoite:

Ensimmäisenä tulee korjata edellisinä päivinä tehdyn laskutusjärjestelmän logon paikka. Se asetettiin laskutuslomakkeen yläosaan, vaikka tarkoitus oli, että se asetettaisiin alimmaiseksi. Toisena tavoitteena on lisätä työmääreisiin mahdollisuus paikkatietojen syöttämiseen. Päivän aikana tuli uusi tavoite lisätä edellä luotuun kuvien tallennusjärjestelmään jonkinlainen koonseuranta ja rajoitin sille, kuinka suuria kuvia tai kuinka suuren resoluution kuvia voi tallentaa.

Tulos:

Laskutusjärjestelmän logon asettelu oli jo tehty jonkun toisen ohjelmoijan toimesta. Tämä huomattiin, kun muutoksia lähdettiin puskemaan etäkansioon. Tulevaisuudessa tuleekin tarkistaa epäselvissä tilanteissa, jos päivitys on jo tehty niin voi välttyä turhalta työltä. Kolmannen tavoitteen ollessa suuremmalla prioriteetilla toiseen tavoitteeseen nähden, toinen tavoite jäin kesken. Toiminnallisuus saatiin lisättyä, mutta sen visuaalisella puolella ilmeni joitain lataushäiriöitä, kuten tyhjien arvojen pohjalta luotujen objektien hetkellinen välähtäminen ruudulla, ennen kuin oikea arvo ilmestyy.

Kolmas tavoite saatiin jokseenkin toimimaan. Kun yrittää ladata liian suurta kuvaa, niin ohjelma pysäyttää tämän toiminnon. Siitä kuuluu myös näkyä jonkinlainen virheviesti, mutta vielä tässä vaiheessa tuntemattomasta syystä, tämä virheviesti ei ilmesty. Kuva saneerataan tällä hetkellä vain bittikoon puolesta ja vielä tulisi selvittää, kuinka resoluutioita voi tarkastella Flutter:in avulla.

2.7.

Tavoite:

Ensisijainen tavoite on viedä loppuun edellisenä päivänä aloitettu ladattujen kuvien moderointi. Täytyy selvittää mikä estää virheviestin ilmestymisen, kun liian suuri kuva ladataan. Uutena tavoitteena on saada yrityksen logo tallentumaan ylempänä luotuun testikuvapankkiin.

Tulos:

Moderointi saatiin toimivaksi. Ainoa mikä vielä on systeemissä epäselvää, on järkevän maksimiresoluution määrittäminen. Järjestelmässä resoluutio määritellään kuvan korkeuden ja leveyden tulona pikseleissä, eikä tuolle määritelmälle löytynyt helposti suositeltua kokostandardia.

Liian suuren kuvan lataamisesta ilmestyvän ilmoituksen vika johtui siitä, että tuon ilmoituksen luova ponnahdusikkuna suljettiin ennen kuin virheviesti luotiin. Kun näiden operaatioiden paikka vaihdettiin, ongelma katosi. Ei olla vielä varmoja, miksi tämä ei toiminut kyseisessä kontekstissa, mutta muualla ohjelmassa ei olla huomattu vastaavaa reaktiota.

Logon tallennus kuvapankkiin jäi kesken. Kuva saatiin tallennettua onnistuneesti, mutta sen lataamisessa on jokin ongelma. Syyllinen lienee olevan se, että kuvan polku ei tallennu tietokantaa oikein.

3.7.

Tavoite:

Saada logo ladattua testikuvapankista ja sen polku tallennettua tietokantaan oikein. Uutena tavoitteena on tutkia, miten kuvan lataaminen ja käsittely ohjelmassa vaikuttaa sen kokoon ja muuttaa testi tietokanta tallentamaan kuvat kuvatiedostoiksi tekstitiedostojen asemesta.

Tulos:

Logon tallennus ja lataus saatiin toimimaan, kun huomattiin, että tietokannan käsittelyohjelmassa muutama muuttuja oli nimetty väärin.

Ladattujen kuvien kokoja tarkastellessa huomattiin, että kuvapankkiin ladattujen kuvien koko oli lähes aina alkuperäistä kuvaa suurempi. Ei olla täysin varmoja mistä tämä johtuu, mutta todennäköisesti kuvan kasvamisen takana on siihen tehdyt muunnokset ladatessa ja tiedostotyypistä toiseen. Esimerkiksi, kun kuvan ladatessa valitsee korkeamman laadun, niin kuvan koko voi kohota yli sen alkuperäisen koon.

Itse kuvatiedostojen tallennusprosessi oli melko yksinkertaista saada toimimaan. se vaati melkein samat funktiot, joilla luotiin tekstitiedostot. Merkkijonona lähetetty kuva piti vain muuttaa eri muotoon ennen tiedostoksi kirjoittamista.

4.7.

Tavoite:

Saada kuvantallennusjärjestelmä tunnistamaa eri kuvatyypit ja käsittelemään kuvia niiden mukaisesti. Tällä hetkellä tallennusjärjestelmä tallentaa kaikki kuvat jpg-tiedostoina vaikka ne olisivat alun perin png-tiedostoja. Toisena tavoitteena on selvittää, miten kuvia saataisiin pakattua, joko käyttöliittymän puolella tai viimeistään kuvantallennusjärjestelmässä.

Tulos:

Kuvien tunnistaminen ja tallentaminen oikeassa muodossa saatiin toimimaan. Kävi ilmi, että kun kuva on merkkijonona, sen ensimmäinen kirjain paljastaa kuvan tyypin. Tällä tiedolla tunnistimen luominen oli hyvin suoraviivaista.

Kuvien pakkaaminen käyttöliittymän puolella osoittautui liian hankalaksi toteuttaa. Tähän tarkoitukseen luodun kirjaston funktiot heittivät tunnistamatonta virheviestiä annetuista parametreistä huolimatta. Tämä ei johtanut mihinkään, joten päätettiin siirtyä tutkimaan, jos pakkaus onnistuisi taustajärjestelmässä, joka käyttää eri kehystä. Taustajärjestelmässä saatiin toimimaan funktio,

joka voi pakata kuvia huomattavasti pienempään muotoon ja myös muuttamaan niiden tiedostotyyppiä. Täten myös edellisinä päivinä yritetty kuvien muuntaminen webp-kuviksi on mahdollista. Tämä onnistui Node.js:sään luodulla Sharp-kirjastolla.

5.7.

Tavoite:

Edellisenä päivänä saatiin kuvien pakkaaminen ja muuntaminen webp-muotoon toimimaan, mutta prosessissa pakkaamaton kuva ei poistu. Tämä aiheuttaisi jokaisen kuvan tallentumisen kahdesti. Tavoitteena on keksiä keino poistaa alkuperäinen kuva, kun kuva on pakattu.

Tulos:

Systeemin virhe saatiin korjattua, mutta se oli suuremman työn takana kuin aluksi oletettiin. Ensimmäinen ratkaisu oli yrittää poistaa vanhaa kuvaa pakkaamisen jälkeen, mutta tätä ei saatu toimimaan. Ei selkeästi tiedetä tarpeeksi asynkronisten funktioiden toiminnasta, sillä joka kerta ratkaisuyrityksen pysäytti samantapainen virheviesti. Tämä viesti heitettiin aina kun vanhaa kuvaa yritettiin poistaa ja se antoi ymmärtää, että poistettava tiedosto oli jonkin muun ohjelman käytössä. Aluksi oletettiin, että tämä johtui siitä, että kuvia tarkasteltiin erillisessä tiedostojenhakuohjelmassa, mutta tuon ohjelman sulkeminen ei muuttanut tilannetta. Kokeiltiin useita eri tapoja varmistaa, että kuvan muokkaaminen olisi mahdollista, mutta lopulta ajatuksesta luovuttiin.

Toinen ratkaisu tuotti tulosta. Koska ongelma johtui yksinomaan siitä, että pakkaamiseen käytetty funktio tarvitsi alkuperäisen kuvan ladattuna kuvapankkiin, niin sitä lähdettiin muokkaamaan siten, että se voisi käyttää suoraan käyttöliittymästä ladattua base64-tiedostoa. Lyhyen tiedon etsinnän jälkeen ratkaisu löytyi ja kuva pakkausjärjestelmä saatiin toimimaan halutulla tavalla.

8.7.

Tavoite:

Korjata ladatuissa kuvissa esiintyvä vääristyminen ja kuvien kasvaminen leveydessä tai pituudessa kuvan pakkauksen aikana. Toisena tavoitteena on lisätä yrityskäyttäjien sosiaalisia medioita tietokantaan ja korjata laskutusjärjestelmässä ilmenevää logon vääristymistä. Kolmantena tavoitteena on jatkaa joitakin päiviä sitten aloitettu paikkatietojen lisääminen työmääreisiin ja asiakastietoihin.

Tulos:

Ladattujen kuvien vääristymät saatiin loppumaan ottamalla alkuperäisen kuvan mitat pakkausfunktioon. Sosiaalisten medioiden lisääminen tietokantaan vaati vain parin uuden muuttujan lisäämisen. Logon vääristyminen ratkesi, kun logon muoto päätettiin vaihtaa ympyrästä nelikulmioksi. Paikkatietojen lisääminenkään ei tuottanut ongelmia. Joku muu ohjelmoijista oli korjannut työmääreiden lisäyslomakkeen puutteet. Loput päivästä kului huolto-ohjelmien toiminnallisuuksien muuntelemisessa ja ulkoasun päivittämisessä.

9.7.

Tavoite:

Saada hakujärjestelmä toimimaan huolto-ohjelmien kanssa. Tällä hetkellä tietyt haut eivät toimi tai aiheuttavat virheviestejä. Tavoitteena on myös lisätä hakumahdollisuuksia, kuten mahdollisuus hakea huolto-ohjelmia kommenttien tai resurssien pohjalta. Tulee myös lisätä hakukenttä kommenttien ponnahdusikkunaan ja yksittäisiin huolto-ohjelmiin.

Tulos:

Päivän tehtävät eivät niinkään olleet vaikeita siinä missä ne olivat aikaa vieviä. Hakujen aiheuttamat virheet johtuivat hakufunktiosta, joka oli melko laaja sikermä if-toteamuksia ja for-silmukoita. Näiden lukeminen sisäkkäin on melko työlästä ja siten myös ongelmakohtien löytäminen vei aikaa. Virheet saatiin kuitenkin korjattua.

Hakumahdollisuuksien lisääminen oli yksinkertaisten suodattimien lisäämistä hakufunktioon. Aikaa vievin osa tätä oli käyttöliittymän asentaminen, jotta noita suodattimia voitaisiin hallita. Saatiin lisättyä uusi yleinen komponentti käytettäväksi myöhemmin. Tämä komponentti on hakukenttä ja toimii siten, että se ottaa vastaan tekstineditointi ohjaimen ja funktion, joka ajetaan, kun tekstiä muutetaan. Tätä komponenttia voidaan nyt käyttää aina kun käyttöliittymässä tarvitaan hakukenttää.

Huomattiin että asynkroniset funktiot voivat aiheuttaa hetkellisiä virheviestejä, jos niitä käytetään, kun Flutter kasaa komponenttipuun uudestaan. Tämän voi kiertää asettamalla asynkroniset funktiot ennen muita funktioita, mutta tulisi silti yrittää ottaa selvää, onko tälle parempaa järjestelyä.

10.7.

Tavoite:

Toteuttaa mahdollisuus lisätä työntekijä huollon "tehty"-ilmoitukseen, siten että työn tekijä voidaan nähdä. Saman tapainen järjestelmä tulee myös olla alihankkijan ilmoittavassa kohdassa. Tosin siihen valitaan työntekijän sijasta kontakti. Uutena tavoitteena on katsoa mikä on työmääreissä vikana. Ne ovat jumissa API-kutsuluupissa ja niiden käyttöliittymä on vääränlainen.

Tulos:

Huoltoihin voi nyt lisätä yksittäisen kontaktin tai käyttäjän tiedot ja nuo tiedot voidaan poistaa. Kaikki tämä saatiin myös yhdistettyä tietokantaan. Tämä ei vaatinut mitään ennestään tuntemattomia toimenpiteitä. Toisaalta tehdyt muutokset tekivät yhdestä vanhemmasta funktiosta todella pitkän ja hankalalukuisen. Pitäisi myöhemmin yrittää selvittää, jos sen pilkkominen pienempiin osiin tai pakkaaminen olisi mahdollista.

Huomattiin, että jos luodaan väliaikainen muuttuja, joka on kopio toisesta muuttujasta, muutokset tuossa muuttujassa heijastuvat oikeaan muuttujaan. Tästä ei aluksi oltu varmoja mutta lyhyen testauksen jälkeen tämä varmistui. Tämä on hyödyllistä, kun työskennellään useiden listojen kanssa ja noissa listoissa olevia muuttujia pitää manipuloida.

Työmääreiden ongelma johtui välittäjien virheellisestä käytöstä. Tämä virhe on kirjoitettu ohjelmaan melko tiiviisti sisään, joten sen korjaamiseen tulee todennäköisesti kulumaan odotettua enemmän aikaa. Tämän ongelman korjaaminen jäi kesken.

11.7.

Tavoite:

Ensimmäinen tavoite on korjata työmääreissä havaittu ongelma. Tämän jälkeen aloitetaan kuvien lisääminen huoltoihin, kommentteihin ja resursseihin.

Tulos:

Työmääreiden ongelma oli paikkatietojen hakeminen uudestaan ja uudestaan jatkuvana luuppina tietokannasta. Tämän lisäksi välittäjäohjelmia ei kutsuttu oikealla tavalla työmääresivun latautuessa. Ensimmäinen saatiin ratkaisuta poistamalla välittäjää kutsuvat funktiot komponenteista, ellei niitä oltu sidottu nappulan painalluksiin tai muihin vastaaviin toimintoihin. Jälkimmäinen ongelma aiheutti enemmän päänvaivaa. Sivun toiminta oli toteutettu hyvin eri tavalla, kun mihin oli totuttu. Tämän takia meni melko kauan ennen kuin täysin ymmärrettiin, miten sivu oikeastaan toimii.

Kuvien lisäämisessä ei päästy kovinkaan pitkälle. Lähdettiin kuitenkin kehittämään komponenttia, jolla voisi sekä ladata että visualisoida kuvan. Tätä voisi käyttää monissa käyttöliittymän kohdissa, joihin tulee olla mahdollista lisätä kuvia.

12.7.

Tavoite:

Resursseihin pitää lisätä kuvien lataamismahdollisuus. Toisena tavoitteena on lisätä joitakin parametrejä resursseille.

Tulos:

Koska kuvia tullaan lisäämään moneen eri osaan projektissa, katsottiin järkeväksi luoda mahdollisimman universaali kuvanlataajakomponentti. Jos samaa komponenttia voitaisiin käyttää kaikissa paikoissa, tämä säästäisi myöhemmin paljon aikaa ja tiedostotilaa. Tarkoituksena oli siis luoda komponentti, jonka avulla voi ladata kuvan ja esittää tuo kuva käyttöliittymässä. Tätä varten oli jo edellisinä päivinä luotu joitakin komponentteja, kuten komponentti, joka voi pitää kuvaa sisällään ja joitakin asiaan liittyviä dialogi-ikkunoita. Näistä huolimatta tähän tehtävään kului suurin osa päivästä. Suurin ongelma oli se, miten ladatut kuvat voitaisiin ikään kuin siirtää universaalin komponentin käytöstä varsinaisen komponentin käyttöön. Tämä ratkaistiin lisäämällä kuvanlatausohjelmaan funktioparametri, joka kuvanlataajassa ottaa parametrikseen ladatun kuvan. Kun tämä funktio sitten annetaan parametriksi kuvanlataajaa käyttävän komponentin puolella, tuohon kuvaan voidaan vaikuttaa tällä funktiolla. Myöhemmin kuvanlataajasta luotiin myös skaalautuva versio, joka luo listan kuvanlataajia ja noita lataajia voi mielensä mukaan lisätä listaan.

Resurssien uudet parametrit eivät olleet mitään, mitä jo ei olisi tehty useampaan kertaan. Parametrit olivat yksinkertaisia muuttujia, joten niiden visualisoiminen käyttöliittymässä ja yhdistäminen tietokantaan oli tuttua työtä.

15.7.

Tavoite:

Korjata kuvanlataajassa huomattu puute. Ladattua kuvaa ei voi poistaa. Resurssien parametrit eivät myöskään näy vielä kaikkialla, jossa niiden tulisi. Näiden korjauksien jälkeen on kuvien latausmahdollisuus pitää lisätä huoltojen kommentteihin.

Tulos:

Kuvanlataajaan saatiin lisättyä poistotoiminto, mutta tässä huomattiin virhe. Jostain syystä oikea kuva ei poistunut, kun kuvaa yritettiin poistaa. Lukuisten testien jälkeen tultiin siihen tulokseen, että kuvanlataajan tila ja sen isäntäohjelman tila ovat keskenään ristiriidassa. Tästä syystä kuvanlataajaa yritettiin muuttaa tilasidonnaisesta komponentista funktioksi ja hetken kokeilujen jälkeen tämä onnistui.

Resurssien parametrit saatiin lisättyä vaadittuihin paikkoihin käyttöliittymässä ilman ongelmia.

Kuvien lisääminen kommentteihin oli niin ikään suoraviivaista. Ainoa mitä piti muuttaa, oli kommenttilomakkeen muoto funktiosta komponentiksi, jotta se voisi käsitellä omia tilojaan.

Edellä mainittujen töiden tekemiseen kului niin vähän aikaa, että saatettiin aloittaa huolto-ohjelmien pdf-latausohjelman päivittäminen. Vanha fontti ei jostain syystä toiminut pdf-kääntäjässä, joten aluksi jouduttiin selvittämään, miten ladata uusi fontti kääntäjän käytettäväksi. Kun uusi fontti oli saatu ladattua, keskityttiin pdf-dokumentin ulkoiseen asetteluun. Tätä ei saatu valmiiksi.

16.7.

Tavoite:

Luoda pdf-kääntäjä huolloille ja huolto-ohjelmille.

Tulos:

Pieniä virheiden setvimisiä lukuun ottamatta pdf-kääntäjien luominen sujui melko helposti. Suurin ongelma syntyi siitä, että kun yrittää ladata yhteen pdf-tiedoston sivuun liikaa dataa niin se aiheuttaa sen, ettei tuo sivu näy ladatussa pdf:ässä. Tähän löydettiin myöhemmin ratkaiseva funktio.

Pdf-kääntäjien luominen voisi olla otollisin kohde automatisoidulle testaukselle. Pdf-kääntäjien testaaminen manuaalisesti on erittäin hidasta koska pdf pitää ladata joka kerta uudestaan, kun halutaan tarkastella työn tuloksia. Jos tämän prosessin voisi automatisoida, niin se pienentäisi työaikaa luultavasti yhdellä kolmanneksella.

17.7.

Tavoite:

Korjata muutama ohjelmassa huomattu virhe. Huoltojen käyttöliittymä ei toimi oikein ja resurssien ajankohdat voivat olla ristiriidassa toistensa kanssa. Uutena tavoitteena on viedä eteenpäin kuvapankin toimintaa. Tallennussysteemi kaipaa uutta tapaa lajitella uusien käyttäjien kansiot ja noiden käyttäjien eri resurssien kansiot. Tavoitteena on luoda systeemi, jossa ei tarvitsisi tallentaa juuri lainkaan osoitteita kuville.

Tulos:

Virheet ohjelmassa saatiin korjattua ilman sen suurempia vastoinkäymisiä. Ainoa joka aiheutti hieman päänvaivaa oli fakta, ettei Dart-kielen DateTime-luokan muuttujia voi asettaa muuttumattomiksi. Tämän takia eräässä funktiossa jouduttiin vapaaehtoisten parametrien sijaan turvautumaan null-arvoihin.

Kuvapankin toimintaa saatiin suoraviivaistettua. Ennen kuvapankki loi jokaiselle käyttäjälle ja yritykselle oman kuvapankki tunnisteensa, mutta tämä tunniste saatettiin korvata yrityksen omalla tunnisteella, jolloin koko tunnistesysteemi saatettiin kirjoittaa uudelleen yksinkertaisemmin. Seuraavaksi on ongelmana miten saataisiin eroteltua ohjelman eri osien kuvat kuvapankissa, ilman että kuvapankki täyttyisi tuhansilla kansioilla.

18.7.

Tavoite:

Jatkaa kuvapankin päivittämistä. Tavoitteena on onnistuneesti tallentaa ja ladata resursseihin lisättävät kuvat.

Tulos:

Tavoitetta ei saavutettu kokonaan. Ohjelma pystyy tallentamaan kuvia ja antamaan niille nimet, joiden mukaan ne voidaan myöhemmin tunnistaa. Kuvien lataaminen onnistuu myös, mutta kuvien poistaminen ei vielä toimi oikein. Jos kuvia tallentaa 3 ja myöhemmin poistaa niistä yhden, niin ohjelma ei tietokannan puolella tunnista mikä kuva poistettiin vaan kirjoittaa vain uudet kuvat vanhojen päälle, jättäen ylimääräiset rauhaan. Tämä johtuu siitä, ettei ohjelmassa ole varsinaista poistamisfunktiota vaan päivitetty kuvalista tallennetaan vanhan päälle. Tälle voisi kirjoittaa funktion, joka poistaa kaikki kuvat, joilla on numerotunniste yli tallennettavien kuvien määrän tai jotain vastaavaa ennen kuin kuvat tallennetaan.

19.7.

Tavoite:

Ladata tallennetut resurssien kuvat onnistuneesti kuvapankista.

Tulos:

Kuvien tallentaminen ja poistaminen saatiin toimimaan. Poistamisongelma ratkaistiin siten, että kaikki tietyllä tunnisteella varustetut kuvat poistetaan aina kun kuvat tallennetaan. Tällöin uudet kuvat korvaavat vanhat. Täten varsianista poistologiikkaa ei tarvitse luoda, vaan se tapahtuu ohjelman rutiinitoimintojen lomassa.

Uutena tavoitteena on saada kuvien tallentaminen toimimaan huoltojen kommenttien kuvilla. Tämä ei onnistunut päivän aikana. Koska kommentit eivät ole niin sanotusti itsenäisiä olioita tietokannassa vaan ovat aina osa huoltoa, niin niille ei ole tarvinnut luoda omaa tunnistetta. Nyt kun kommenteilla tulee olla kuvia, ja nuo kuvat tulee jotenkin liittää oikeisiin kommentteihin, tulee kommenteilla olla tunniste. Tämän saavuttaminen oli melko suuren työn takana. Aluksi oli ongelmana saada muutettua tietokantaan lähetetty kommenttidata kommenttiolioksi ja luotua siitä tietokannan kanssa yhteen sopiva versio, jotta tietokanta loisi sille tunnisteen. Tämä saatiin

onnistumaan hyödyntämällä jo olemassa olevaa logiikkaa, jolla huollon kommentit oli liitetty tietokantaan ennen.

Tämä ei kuitenkaan luonut kommenteille omia tunnisteita, joten kommenteille piti luoda omat tallennusfunktionsa, jotta tietokanta loisi niille tunnisteet. Tämän takia kommentit kuitenkin tallentuivat tietokantaan omina olioinaan, joka ei ollut toivottavaa. Tästä syystä ohjelmaan lisättiin funktio, joka poistaa itsenäisen kommentin tietokannasta, kun tuo olio on ensin lisätty huollon kommenttilistaan. Tällä tavalla saatetiin luoda tunnisteella varustettuja kommentteja, jotka ovat olemassa vain huoltojen yhteydessä. Tästä eteenpäin ei tämän päivän aikana ehditty, joten tavoitteeseen ei päästy.

22.7.

Tavoite:

Saada kommenttien tallennus- ja latausjärjestelmä valmiiksi.

Tulos:

Kommenttien kuvat saatiin tallentumaan kuvapankkiin ja latautumaan oikein. Kun kommenteilla oli omat tunnisteensa, niin oli suhteellisen suoraviivaista tehdä niille omat funktiot, jotka tallensivat kuvat. Tämä oli lähinnä pitkällinen prosessi, koska kommenttien käsittelyyn vaaditaan useita sisäkkäisiä silmukoita. Tämä oli pakollista kommenttien ollessa säilöttynä listan sisään Map-olion sisällä. Tämän lisäksi kokemuksen puute karttaolioiden kanssa työskentelemisestä JavaScript-ohjelmointikielellä vaikeutti työtä.

Kommenttien poistamista oli aluksi vaikea saada toimimaan. Koska kommentit ovat jo kadonneet huollon listoilta, ennen kuin ne siirretään tietokannan puolelle, ei ohjelma voi tunnistaa minkä kommentin kuvat tulisi poistaa. Muutaman epäonnistuneen toteutuksen jälkeen päätettiin palata samaa strategiaan, jota oli menneisyydessä käytetty. Kun huolto tallennetaan, kaikki tuon huollon kommenttien kuvat poistetaan. Tämän jälkeen huollon mukana tulleet kuvat tallennetaan poistettujen tilalle. Tämä saattaa aiheuttaa ylimääräistä rasitetta ohjelmalle, jos kuvatiedostoja on paljon, mutta tässä vaiheessa toteutus on helposti ymmärrettävä ja suoraviivainen.

Jotta kommenttien tallentaminen saatiin toimimaan, jouduttiin kirjoittamaan joitakin uusia funktioita kuvien käsittelyjärjestelmään. Osa näistä osoittautuivat myös hyödyllisiksi muissa järjestelmän osissa. Esimerkkinä funktio, joka poistaa kansioita tai funktio, joka poistaa kaikki tietyllä tunnisteella varustetut kuvat. Tässä vaiheessa funktioiden määrä ja samankaltaisuus oli alkanut vaikeuttaa koodin luettavuutta ja tapa jättää koodi kommentoimatta hidasti niin ikään työtä. Tässä kohtaa päätettiin lisätä kommentit jokaiseen funktioon, jotta niiden toiminnallisuus olisi ilmiselvää myöhemmin. Jotkin funktiot olisivat myös yhdistettävissä yhdeksi funktioksi, koska niiden toiminnallisuus on lähes sama. Tällä tavalla voitaisiin poistaa turhaa koodia.

23.7.

Tavoite:

Edellisten päivien aikana huomattiin kaksi virhettä ohjelman toiminnassa. Jos kommentteja poistaa useamman kuin yhden, tämä poistaminen ei näy käyttöliittymässä.

Jos huolto poistetaan, tämä aiheuttaa virheviestin huolto-ohjelmassa, jossa tuo huolto oli. Uutena tavoitteena on lisätä jonkinlainen aakkosjärjestelmä kuvapankkiin. Kun yritykselle tehdään oma

kansio kuvapankkiin, tuolle kansio tulee tallentaa yrityksen nimen mukaan kirjaimella nimettyyn kansioon. Tuo kirjain vastaa yrityksen nimen ensimmäistä alkukirjainta.

Tulos:

Vieläkään ei saatu selville, mikä aiheuttaa kommenteissa sen, että vain ensimmäinen poistotoimenpide heijastuu käyttöliittymään. Tämä ongelma kuitenkin korjattiin muuttamalla suoraan sitä huoltoa, josta lista luodaan käyttöliittymän puolella, eikä muuttamalla tietokannan huoltoa ja lataamalla se uudestaan käyttöliittymään. Tämä tallennusprosessi tapahtuu edelleen, kun kommentteja muokataan, mutta käyttöliittymän kasaaminen ei enää riipu siitä.

Huolto-ohjelmien virhettä ei enää saatu toistettua, joten oletettiin, että virhe johtui jotenkin kommenttien toiminnasta.

Kuvapankkia ohjaavassa ohjelmistossa on funktio, joka luo polkuja kuvien tallentamista varten. Ohjelma myöhemmin luo tuolla funktiolla kirjoitetun polun. Aakkoskansion lisääminen kuvapankkiin vaati siis yhden välivaiheen lisäämistä polun luovaan funktioon. Suurempi ongelma oli luoda järjestelmä, joka ottaisi huomioon sen, että yrityksen nimi voi muuttua. Tämä oli teoriassa melko yksinkertaista. Luodaan ensin funktio, joka kopioi yrityksen kansion uuteen kansioon ja sitten toinen funktio, joka poistaa yrityksen vanhan kansion. Ennalta ladatussa moduulissa on työkaluja tätä varten, mutta eniten aikaa vei funktion sovittaminen muun ohjelman kanssa. JavaScript ei ilmoita ristiriidoista muuttujien tai funktioiden kutsumisissa. Tämä tekee virheiden löytämisestä ja virheviestien ymmärtämisestä joskus melko haastavaa.

Ei vielä tiedetä voiko MongoDB-tietokannasta hakea yksittäistä osaa yhdestä dokumentista. Tämä voi olla hyödyllistä, sillä kuvanpankin uudistuksen myötä, käyttäjien ja yritysten nimet ovat nousseet tärkeäksi osaksi kuvapankin toimintaa. Tällä hetkellä koko käyttäjä haetaan erikseen tietokannasta aina kun nimeä tarvitaan. Ei tiedetä voisiko se olla tehokkaampaa tai selkeämpää hakea pelkkä nimi. Jos tämä käytäntö yleistyy, käyttäjiä ja yrityksiä tullaan hakemaan tietokannasta hyvin usein ja suuren tietomäärän siirtäminen JSON:illa toistuvasti voi olla raskasta verkkoyhteydelle.

24.7.

Tavoite:

Päivittää joitakin osia huoltojen-, huolto-ohjelmien- ja resurssien käyttöliittymästä. Jotkin osat siitä ovat karun näköisiä tai sotkuisia. Niitä tulee siistiä. Tämän lisäksi on tarkoitus lisätä hakutoiminnot resursseihin ja huoltoihin.

Tulos:

Huoltojen ja resurssien käyttöliittymää saatiin päivitettyä miellyttävämmän näköisiksi. Myös huolto-ohjelmissa ulkonäköä saatiin parempaan suuntaan, mutta sen osalta työ on vielä kesken. Eniten aikaa kului, kun huolto-ohjelmien sivun kahdessa eri osassa olevat kappaleet piti sijoittaa yhteen linjaan. Nämä osat käyttöliittymää eivät olleet juuri missään yhteydessä toisiinsa, joten järjestämiseen ei voitu käyttää mitään komponenttia a suoraan, joka voisi järjestää ne linjaan toistensa suhteen.

Ongelmaa yritettiin ratkaista useilla eri menetelmillä, kuten komponenteilla, jotka levittävät lapsikomponenttinsa käyttöliittymään tasaisesti tai sijoittamalla molemmat komponentit riveihin ja

kokeilemalla tuolle riville eri asetteluparametrejä. Tulosta ei kuitenkaan syntynyt, joten lopulta asettelu päätettiin hoitaa silmämääräisesti vaihtamalla toisen aseteltavan osan lapsikomponenttien leveyksiä, kunnes ne vastasivat toisen osan lapsien leveyttä osapuilleen. Lopputulos ei ole varmasti paras mahdollinen, mutta tyydyttävä ja toimii myös, kun ruudun leveyttä muuttaa.

On huomattu, että huoltojen, resurssien ja huolto-ohjelmien sivujen välillä käytetään paljon samaa koodia. Näissä koodipätkissä on joitakin eroja, mutta uskotaan, että tuo toistettu koodi voitaisiin muuttaa yhdeksi funktioksi tai komponentiksi.

25.7.

Tavoite:

Huolto-ohjelmien käyttöliittymän ulkoasun päivittäminen ja hakufilttereiden asettaminen erilliseen ponnahdusikkunaan. Nuo hakufiltterit vievät ruudulta muuten liikaa tilaa.

Tulos:

Käyttöliittymän päivittäminen ei ollut vaiheessa vaikeaa, varsinkin kun työohjeissa oli tarkat toiveet lopputuloksesta. Tarkkojen mittojen ja ulkoasujen testaaminen vain vei runsaasti aikaa. Filttereiden siirtäminen ponnahdusikkunaan oli niin ikään yksinkertainen muutos.

26.7.

Tavoite:

Käyttöliittymän tekstien ja värien määrittäminen yleisen teeman kautta. Uutena tavoitteena on muokata joitain käyttöliittymän osia. Työn palautuksen jälkeen nousi esiin useita kohtia, joita voitaisiin parantaa.

Tulos:

Käyttöliittymän teeman määrittäminen ei ollut niinkään vaikeaa, kuin aikaa vievää. Lähes jokainen tekstinpätkä, tekstikenttä ja väri käyttöliittymässä piti kirjoittaa uudestaan. Tässä tehtävässä ctrl+f toiminnosta oli paljon hyötyä, kun saattoi vain kirjoittaa vaikka "text" ja kaikki text-sanat koodissa näkyivät selkeämmin. Tällä tavalla muutokset saatettiin tehdä järjestelmällisesti ja kohtia unohtamatta.

Käyttöliittymiin tehtäviä uusia parannuksia ei ehditty tehdä kokonaan, mutta ei uskota, että se tuottaa suuria ongelmia.

29.7.

Tavoite:

Korjata edellisenä työpäivänä ilmi tuodut puutteet käyttöliittymän asettelussa.

Tulos:

Tässä vaiheessa huomattiin, että tietotaito käyttöliittymän luomiseen Flutter:illa on melko vajavainen, mitä tulee käyttöliittymän eri osien ulottuvuuksien määrittämiseen. Muut osat

käyttöliittymän asettelusta ovat suhteellisen selkeitä, mutta koon määrittäminen virheellisesti tuottaa eniten koodin uudelleen kirjoitusta. Suurin ongelma tuntuu olevan se, että vielä ei täysin ymmärretä periaatetta, jossa lapsikomponentti saa ulottuvuutensa rajat vanhemmaltaan, ja miten se vuorovaikuttaa prosentuaalisten ulottuvuuksien kanssa. Tuntuu siltä, että joissain tapauksissa, kun käyttöliittymän osille antaa niin sanotusti vapaan tilan täytettäväksi, ne täyttävät vain sen tilan, jonka niiden äitiohjelma antaa, mutta toisinaan taas ne levittäytyvät äitiohjelman rajojen yli ja aiheuttavat virheviestin.

30.7.

Tavoite:

Edellisen päivän päivityksissä huomattiin virhe, joka aiheuttaa virheviestin ohjelman windows-versiossa. Tämän lisäksi käyttöliittymässä huomattiin joitain suunnitteluvirheitä, jotka haittaavat käyttöliittymän toimintaa tai myöhempää muokkausta. Tavoitteena on korjata nämä virheet

Tulos:

Virhe windows-versiossa paljastui yleiseksi virheeksi ohjelmassa. Jos resurssin päivämääriä ei määritellä, kun tuo resurssi luodaan, se aiheuttaa virheviestin. Tämä ongelma korjattiin luomalla päivämäärien käsittelyyn logiikka, joka ottaa huomioon, jos päivämäärä on jätetty tyhjäksi.

Muutokset käyttöliittymässä olivat suurimmalta osin aikaa vieviä. Muutettavien käyttöliittymän osien toimintalogiikka jouduttiin kirjoittamaan osin uudelleen ja jo olemassa olevat osat piti muokata sopinaan tuohon uuteen muottiin. Samalla huomattiin joitakin uusia puutteita ja visuaalisia ristiriitoja käyttöliittymän muissa osissa. Nämä kaikki kuitenkin saatiin korjattua, mutta se fakta, että jokaisen muutoksen jälkeen ohjelma aloittaa itsensä alusta pidentää muokkauksiin kuluvaa aikaa huomattavasti.

Ei tiedetä voisiko käyttöliittymän asettelun testaamista automatisoida jotenkin. Flutter:illa on mahdollista testata käyttöliittymän toimintaa, mutta käyttöliittymän koostumuksen testaamisesta ei olla varmoja. Joka tapauksessa, jonkinlainen testausmenetelmä edes käyttöliittymän virheiden havaitsemiselle voisi nopeuttaa työtä huomattavasti.

31.7.

Tavoite:

Luoda huoltojen, huolto-ohjelmien ja resurssien pdf:fille esikatseluikkuna. Tämän lisäksi noiden pdf:fien ulkoasua tulee päivittää. Tämänhetkinen ulkoasu on hyvin karu.

Tulos:

Esikatseluikkunan pohja oli jo tehty toisen ohjelmoijan toimesta. Tämä komponentti piti vain yleistää käyttämään muitakin pdf:fiä. Tämä sujui melko nopeasti ja vaivattomasti. Vanhasta esikatselusta saatiin versio, jota voidaan myös tulevaisuudessa käyttää minkä tahansa pdf:fän esikatseluun.

Itse pdf:fien ulkoasun muokkaaminen oli lähinnä aikaa vievää syistä, joita ollaan jo aiemmin käyty läpi. Toisaalta nyt lisätty esikatselu nopeuttaa prosessia melkoisesti, kun ei tarvitse enää ladata pdf:fää koneelle, jotta sitä voidaan tarkastella. Varsinaista ohjenuoraa pdf:fän päivittämiselle ei

ollut, joten kokeiltiin muutamia eri vaihtoehtoja, kunnes löydettiin yksi joka näytti tekijän silmään hyvältä. Ainoa ongelmatilanne tuli vastaan yhden pdf:fän sisäisen logiikan kanssa. Joskus Flutter:in virheviestit ovat hieman ylimalkaisia, eivätkä aina osoita mitään riviä koodissa tai vastaavaa. Tässä tapauksessa vastassa oli tällainen ja sen aiheuttajan etsimiseen meni jonkin aikaa.

1.8.

Tavoite:

Päivän alussa testattiin menetelmää suoraviivaistaa tietokannan käsittelyn koodia. Päivän aikana tuli tavoite päivittää huoltojen kommenttien ulkoasua ja lisätä kommentteihin niiden kirjoittajien nimet. Toisena uutena tavoitteena oli lisätä tiimeihin esimies ja päivittää tiimien käyttöliittymän ulkoasua.

Tulos:

Ajatus tietokannan toimintojen suoraviivaistamisesta ei johtanut mihinkään. Huomattiin, että näennäisesti turhalla koodilla oli tarkoitus tietyissä konteksteissa. Kuitenkin tulevaisuudessa nyt tunnetaan uusi tapa kirjoittaa yksinkertaisia päivitys- ja luontifunktioita. Ei olla varmoja, mutta saattaisi olla mahdollista, että tällaiset funktiot voitaisiin yleismaalistaa.

Kommenttien päivittäminen ei ollut mitään, mitä ei olisi jo tehty. Ainoa uusi komponentti, joka piti kirjoittaa, oli eräänlainen kuvasikermä. Tämän sikermän tarkoitus on näyttää useita kuvia pienessä tilassa, tällöin säästäen tilaa ruudulta, jos kuvia on paljon. Jotta kuvia voitaisiin tarkastella tarkemmin, tuohon sikermään lisättiin funktio, joka paljastaa kaikki sikermän kuvat erillisessä ponnahdusikkunassa.

Tiimien käyttöliittymästä löytyi useita persustavanlaatuisia puutteita. Tässä voitiin huomata menneiden kuukausien aikana tapahtunut kehitys tietotaidossa. Tämä puutteellinen ja hieman sekavasti kirjoitettu käyttöliittymä oli kirjoittajan itsensä tekemään työsuhteen alkuajoilta. Tässä onkin täydellinen tilaisuus verrata opittuja menetelmiä vanhoihin ja soveltaa näiden kuukausien aikana luotuja komponentteja parantamaan tätä vanhaa käyttöliittymää. Viimeiseen tavoitteeseen ei kuitenkaan päästy.

2.8.

Tavoite:

Lisätä tiimeihin esimies ja parantaa tiimien käyttöliittymän ulkoasua. Toisena tavoitteena on lisätä yrityskäyttäjille profiilikuvat. Jos tästä jää aikaa, niin tulee myös parannella yleisesti käyttäjäroolien- ja yrityskäyttäjien käyttöliittymää.

Tulos:

Esimiehen lisääminen tiimeihin kesti jonkin aikaa. Tiimien käyttöliittymää piti ensin muokata parempaan muotoon, jotta sitä voitaisiin alkaa järkevästi laajentamaan. Työssä törmättiin joihinkin vanhoihin ratkaisuihin, jotka nyt saatettiin korvata opituilla menetelmillä. Varsinkin kuluttajaohjelmien kanssa toimimisessa oli huomattavissa monia puutteita, jotka nyt saatettiin korjata.

Profiilikuvien lisääminen tuotti hieman hankaluuksia. Koodi, johon ne piti lisätä, oli melko laaja rivimäärältään, joten oikeiden kohtien etsiminen ja yleisesti koodin lukeminen oli hieman haastavaa. Muuten työ sujui ongelmitta.

Käyttöliittymän puutteita saatiin korjattua tiimien osalta, mutta muut osat työn alla olevasta osiosta jäivät koskemattomiksi.

Tuon osion arkkitehtuuri on melko vaikeaselkoista. Koko osio on kirjoitettu lähes kokonaan yhteen suureen tiedostoon, joka tekee muokkaamisesta työlästä. Jotta osion toimintaa voitiaisiin jatkossa muokata helpommin, on seuraavaksi tehtävä joitain perustavanlaatuisia muutoksia osion arkkitehtuuriin. Suurimpana esimerkkinä on yrittää paloitella osion yksi suuri tiedosto muutamaksi pienemmäksi tiedostoksi.

5.8.

Tavoite:

Tehdä käyttäjät-sivusta paremmin hallittava pilkkomalla se pienempii osiin ja eri tiedostoihin. Tämän jälkeen tulee tuon sivun ulkoasua parantaa. Jos aikaa jää, työntekijätiimeille tulee antaa mahdollisuus lisätä useita esimiehiä.

Tulos:

Sivun koodin pilkkominen sujui ilman suurempia ongelmia. Koodissa huomattiin joitakin kohtia, joita saatettiin parantaa, kuten hyödytön rullattava lista toisen sisällä. Toimenpiteeseen liittyvän niin sanotun mekaanisen työn ja tällaisten koodin puutteiden takia, työssä meni kuitenkin suhteellisen kauan.

Edellisinä päivinä opittujen metodien avulla käyttöliittymän siistimisessä meni tällä kertaa huomattavasti vähemmän aikaa. Osin toisaalta myös siitä syystä, että tässä käyttöliittymässä on paljon vähemmän osia kuin edellisissä.

Tiimien esimiesten lisääminen tuotti hieman vaikeuksia. Aluksi yritettiin säästää koodia ja käyttää jo olemassa olevaa logiikkaa, jolla käsitellään tiimin jäseniä. Huolimattomuus johti siihen, että tiimin jäsenten ja esimiesten toimintalogiikka sotkeutui soisiinsa ja aiheutti arvaamattomia tulokisa ohjelman toiminnassa. Näiden setvimiseen kului aikaa. Toisaalta samalla keksittiin käyttöliittymä näiden kahden listan ylläpitämiselle, joka on toiminnaltaan lähempänä muuta sovellusta. Tämän pitäisi helpottaa sovelluksen sisäisen logiikan ymmärtämisessä.

6.8.

Tavoite:

Selvittää mikä aiheuttaa huolto-ohjelmissa virheviestejä ja korjata nuo virheet. Tämän jälkeen täytyy selvittää voiko laajemmat profiilikuvat sovittaa pienempään kuvakehykseen ilman että kuva leikkautuu.

Uutena tavoitteena tuli korjata työntekijätiimien käyttöliittymässä ilmeneviä virheitä. Jostain syystä työntekijöiden roolit eivät näy oikein ja hakufiltterit eivät vastaa.

Tulos:

Huolto-ohjelmien virheet osoittautuivat monimutkaisiksi korjata. Huolto-ohjelmien tietoikkunassa on painike, josta voi avata huolto-ohjelman muokkausikkunan. Molemmat näistä toimivat itsenäisesti oikein, mutta jos huolto-ohjelmaa muokkaa tämän samalla kun tietoikkuna on auki, niin muutokset eivät heijastu oikein tietoikkunaan. Aluksi muutokset eivät joko näkyneet ollenkaan tai jos muutokset haluttiin mitätöidä, mitätöinti ei onnistunut. Pitkällisen tutkimuksen jälkeen huomattiin sen johtuvan siitä, etteivät tietoikkunan ja muokkausikkunan huolto-ohjelmat olleet varsinaisessa yhteydessä toisiinsa, eivätkä siten heijasta toistensa muutoksia. Tämä korjattiin asettamalla muokkausikkuna palauttamaan muokattu huolto-ohjelma muokkauksen jälkeen tietoikkunalle, joka korvaa oman huolto-ohjelmansa tällä huolto-ohjelmalla.

Ongelmat kuitenkin jatkuivat huolto-ohjelmien huoltojen ja resurssien päivittymisen kanssa näiden ikkunoiden välillä. Tämän ongelman aiheuttajaksi todettiin, että muokkausohjelma muokkaa huolto-ohjelman sisältöä liian suoraan ja täten muutosten mitätöinti ei onnistu. Huolto-ohjelman resursseista vastaa erillinen olio, joka haettiin suoraan huolto-ohjelmien välittäjältä. Tämän takia, jokainen muutos tuon huolto-ohjelman resursseihin tai huoltoihin jäi pysyväksi. Kun tästä oliosta tehtiin syväkopio muokkausohjelman käyttöön, saatettiin kaikki muokkaukset tehdä tuohon kopioon ilman että alkuperäinen muuttuu.

Profiilikuvan leikkautuminen saatiin korjattua vaihtamalla kuvaohjelman yhtä asetusta. Työntekijöiden käyttöliittymän ongelmat johtuivat edellisinä päivinä tehdyistä laajoista muutoksista. Hakuohjelman toimivuutta ei ollut tarkistettu ja nyt huomattiin, että hakuohjelma ei toiminut uudessa järjestelyssä. Koodi oli vieläkin jokseenkin tuntematonta, joten tämän ongelman selvittäminen oli melko haasteellista. Useissa kohdissa ongelma piili tilojen hallinnassa. Lopulta ratkaisu saavutettiin kokeilemalla erilaisia vaihtoehtoja ja poistamalla koodista turhia välivaiheita.

7.8.

Tavoite:

Lisätä kuvanlataajaohjelmiin jonkinlainen kuvan koon muuttaja, jotta tietokantaan ei tallennettaisi liian suura kuvia.

Tulos:

Koska kokoa muuttavat funktiot oli jo tehty, niin ne täytyi vain lisätä kuvanlataajiin. Tämä tosin ratkaisi vain puolet ongelmasta. Toinen puoli ongelmaa on se, ettei tietokantaan voida lähettää 80 kilotavua suurempia kuvia. Jos yrittää, tämä aiheuttaa virheviestin. Ongelmaksi muodostuikin siis keksiä keino pakata kaikenlaiset kuvat korkeintaan tämän kokoisiksi. Tässä kohtaa vajaa tietämys tiedostojen pakkaamisesta tuli vastaan. Useita eri menetelmiä yritettiin, mutta ne joko eivät toimineet kaikkien kuvakoiden ja kuvatyyppien kanssa tai sitten toimivat, mutta veivät liikaa aikaa. Tällainen pitkä funktio saatiin aikaan, kun kuvaa ajettiin toistuvasti pakkausfunktion läpi verraten tuloksen kokoa vaadittuun kokoon. Kun vaadittu koko oli saavutettu, silmukka katkaistiin. Tämä toimi melko hyvin, mutta varsinkin suurempien kuvien kohdalla pakkaamiseen kului kymmeniä sekunteja. Tavoitteeseen siis päästiin vain osittain.

Mahdollisia ratkaisuja olisi löytää jokin uusi tapa kuvien pakkaamiselle, joka toimisi universaalimmin tai sitten jakaa pitemmät pakkausfunktiot erilliselle säikeelle, jolloin se voisi suorittaa toimintonsa nopeammin tai ainakin ohjelman taustalla.

8.8.

Tavoite:

Keksiä keino kaikenlaisten kuvien skaalaamiseen kuvia ladatessa. Koska kuvat voivat olla kaiken kokoisia ja skaalautua eri tavoilla niin tähän täytyy kehittää jokin silmukka tai vastaava, jolla kuva voitaisiin pakata useampaan otteeseen, jos ensimmäisellä kerralla ei synny haluttuja tuloksia.

Uutena tavoitteena on muokata aloitussivua siten, että jos käyttäjällä ei ole yrityksiä, aloitussivussa ei näy yrityspalkkia ja jos yrityksiä on useita, niin käyttäjä voi valita mihin yritykseen palkin nappulat johdattavat.

Tulos:

Kuvien pakkaamista ei saatu kunnolla toimimaan. Koska automaattinen kuvan ajaminen pakkaussilmukan läpi huomattiin vievän liikaa aikaa, niin päädyttiin vaihtoehtoon, jossa käyttäjä voi itse valita, haluaako pakata kuvaa pitemmälle vai ladata pienemmän kuvan. Tämä onnistui kun-silmukkaa hyödyntämällä. Kun kuva on liian iso, niin ohjelma kysyy käyttäjältä, haluaako hän pakata kuvaa, jolloin myöntävän vastauksen jälkeen kuvaa pakataan uudelleen. Joka kerta pakkauksen parametrejä muuttaen. Tämä saatiin toimimaan, mutta jostain syystä uusiopakkauksesta kysyvää ikkunaa ei saatu katoamaan silmukan aikana. Huomattiin, että tämä johtui pakkausfunktion raskaudesta. Kun tuo funktio käynnistyy, se jäädyttää käyttöliittymän paikaleen.

Tätä ongelmaa yritettiin aluksi korjata muuttamalla pakkausfunktion ajamiskohtaa, lisäämällä pientä viivettä ponnahdusikkunan poistumisen ja pakkauksen alkamisen väliin ja lopulta siirtämällä pakkausfunktio uuteen säikeeseen. Mikään näistä ei tuottanut tulosta. Viimeisimmän mahdollisuudet ovat vielä hieman auki, sillä Flutter:in pääfunktio useampien säikeiden käsittelyyn ei toimi verkkoversiossa. Tälle löydettiin kirjasto, jonka pitäisi korjata tämä puute, mutta tuon kirjaston funktioita ei saatu toimimaan.

Loppujen lopuksi, tehdyt muutokset pyyhittiin ja tehtävä hylättiin toistaiseksi. Päivän viimeiset hetket tutustuttiin aloitussivun toimintaan ja onnistuttiin poistamaan yrityspalkki, kun yrityksiä ei ole.

9.8.

Tavoite:

Aloitusnäytössä näkyy edelleen valikko yritykselle, vaikka käyttäjällä ei yrityksiä olisikaan. Tällä hetkellä ei ole myöskään mitään tapaa hallita, mitä yritystä tuo valikko koskee. Tavoitteena on korjata tämä ja lisätä navigointifunktiot aloitusnäytön yrityksen ohjauspaneeliin.

Tulos:

Aloitusvalikkoa näytetään ohjelmassa monessa eri paikassa ja kontekstissa. Tälle on myös useita eri isäntäohjelmia ja näihin liittyen muutama eri toimintalogiikkaa. Tämä teki aluksi koodin ymmärtämisestä haasteellista ja niin ikään muokkauksesta hidasta. Tavoitteen ensimmäinen puolisko saavutettiin kuitenkin melko helposti. Se miten tavoite saavutettaisiin ei ollut ongelma vaan, miten tuo ratkaisu saataisiin sopimaan muun koodin joukkoon.

Ohjauspaneelin nappuloihin navigointijärjestelmän lisääminen tuotti enemmän vaikeuksia. Aihealue ei ollut ennestään Flutter:issa tuttua. Sivuille siirtyminen ei niinkään ollut vaikeaa. Jos nuo

sivut vaativat muuttujia toimiakseen ja varsinkin jos nuo muuttujat eivät saaneet olla tyhjiä, niin toteutus muuttui mutkikkaammaksi. Navigointifunktioihin jouduttiin tekemään vaihtoehtoisia reittejä, jos jostain syystä aiottuun päämäärään tarvittavat parametrit olisivat tyhjiä. Tämä ratkaisu sai ohjelman toimimaan, mutta saattaa tulevaisuudessa aiheuttaa odottamatonta käytöstä ohjelmalta, jos parametrit jostain syystä ovat tyhjiä.

Täytyy vielä yrittää tutustua siihen, miten navigoinnissa tulisi menetellä tyhjien muuttujien kanssa.

12.8.

Tavoite:

Selvittää miksi pdf-lataaminen ei toimi ohjelman Windows-versiossa ja korjata asia. Jos aikaa jää, niin selvittää voisiko taustajärjestelmän asettamaa tiedostokokorajoitusta kohottaa. Tällä hetkellä 80 kilotavua suurempia tiedostoja ei voi lähettää.

Tulos:

Pdf-tiedoston lataamisen esti windows-version kyvyttömyys syystä tai toisesta käyttää erikseen ladattuja fonttitiedostoja. Syytä tälle puutteelle Windows-versiossa ei tiedetä tai saatu selville, mutta huomattiin, ettei pdf:fän ulkoasu muuttunut juurikaan fonttien poistamisen jälkeen. Kun tämä oli suoritettu, pdf-pystyttiin lataamaan ongelmitta.

Taustajärjestelmän tiedostokokoa ei saatu muutettua. Muutamaa eri ratkaisua yritettiin, mutta ne joko eivät toimineet tai olivat vanhentuneita järjestelmiä. Tämän jälkeen yritettiin jälleen kerran saada kuvien pakkaamisesta jouhevampaa. Tässäkään ei päästy sen pidemmälle, kuin että saatiin lisättyä resoluution tarkistus ja pienennys kuvanlataajaohjelmiin. Loppupäivän tunnit kuluivat projektien tehtävien hakujärjestelmän hiomisessa. Siihen lisättiin hakutulosten järjestäjä.

13.8.

Tavoite:

Projektien tehtävien hakufiltteriin tulee lisätä filtteri, joka järjestää tehtävät luomispäivän mukaan. Tämän lisäksi aloitusnäytön "Älä näytä tätä uudelleen"-painikkeesta pitää saada toimiva.

Tulos:

Tehtävien hakufiltterin tekeminen vaati luomispäivän lisäämisen tehtävään. Tämä ei tuottanut ongelmia. Itse filtterin luominenkaan ei ollut tässä vaiheessa mitään mitä ei olisi jo tehty. Filtterifunktioille tosin tulisi yrittää etsiä jonkinlainen pohja, sillä nyt niistä muodostuu monimutkaisia if-toteamusten ja for-silmukoinen täyttämiä funktioita. Tällisten funktioiden muokkaaminen tulevaisuudessa tulee olemaan varmasti muuten vaikeaa.

Aloitusnäytön muokkaaminen osoittautui vaikeammaksi kuin aluksi oletettiin. Aloitusnäytön logiikkaa hyödynnetään ympäri sovellusta, joka teki niin sanotusti oikean aloitusnäytön löytämisestä tiedostoista vaikeaa. Myös virhetilanteet aiheuttivat enemmän päänvaivaa tästä syystä. Loppupuolella päivää jumiuduttiin myös päivittämään Flutter:ia ja setvimään tuon päivityksen tuomia ongelmia muiden riippuvaisuuksien kanssa.

14.8.

Tavoite:

Lisätä uusia navigointimahdollisuuksia kotisivun yrityspalkkiin. Tutkia uusia mahdollisuuksia kuvien pakkaamista varten. Tutkia miksi aloitusnäytön ilmestymisen estäminen joissain tapauksissa muuttaa ruudun mustaksi.

Tulos:

Navigointimahdollisuuksien lisääminen vaati lähinnä samantapaisten merkkijonojen ja koodinpätkien kirjoittamista uudestaan ja uudestaan. Tämä ei ollut vaikeaa, mutta oli hidasta.

Kuvien pakkaukseen ei saatu uusia järjestelyjä. Tällä hetkellä samaa prosessia viedään pariin otteeseen silmukan läpi toivoen, että se pienentäisi kuvaa tarpeeksi. Jos kuva ei ole tarpeeksi pieni, sen lataaminen estetään ja tästä annetaan ilmoitus käyttäjälle. Tämä ei varmasti ole paras vaihtoehto, mutta se näyttäisi toimivan useimpien kuvien kohdalla.

Mustan ruudun ongelman syyksi ilmeni edellisenä päivänä lisätty container-komponentti. Tämä komponentti oli lisätty pitämään paikkaa kohdassa, jonka sisältöä aluksi oletettiin turhaksi. Kävi kuitenkin ilmi, että tämä container oli se musta ruutu ja sen täyttämällä sisällöllä oli tarkoituksensa. Tämän muutoksen mitätöimisen jälkeen aloitusnäyttö toimi taas normaalisti.

15.8.

Tavoite:

Vaihtaa yrityspuolen kotisivu sivusta ponnahdusikkunaksi ja selvittää, mikä aiheuttaa aloitussivun monistumisen tai katoamisen, kun sovellus käynnistetään.

Tulos:

Yrityspuolen kotisivun vaihtaminen ponnahdusikkunaksi ei tuottanut juurikaan vaikeuksia. Ainoa minkä löytäminen oli hieman vaikeaa, oli se komponentti, joka todella kutsuu kotisivun. Kotisivujen toimintalogiikka on hieman tuntemattomalla tavalla ohjelmoitu.

Aloitussivun katoamisen selvittäminen kesti melko kauan. Sen toistaminen oli epäsäännöllistä ja virhe tuntui muutenkin tapahtuvan melkein aina vain windows-versiossa. Siihen liittyvien virheviestien etsiminen verkosta ei myöskään valaissut asiaa juurikaan. Lopulta yritettiin tutkia tarkemmin koodia, joka kutsuu aloitussivun, huomattiin että edellisenä päivänä tehty muutos, jonka uskottiin poistavan, tuplasti ilmestyvä aloitusruutu aiheutti mustan ruudun ongelman.

Myöhemmin päivän aikana yritettiin selvittää, miksi yrityksen kanban-osiossa navigaatio ei toimi joissain kohdissa oikein. Projektin kanbanjärjestelmä oli entuudestaan jokseenkin tuntematon, joten taas kerran eniten aikaa meni sen ymmärtämisessä.

16.8.

Tavoite:

Korjata yrityspuolen projektien käyttöliittymässä huomatut virheet. Ponnahdusikkunat eivät toimi oletetulla tavalla. Tämän jälkeen on tarkoitus tutkia mahdollisuuksia lähettää viestejä

taustajärjestelmän kautta. Uutena tavoitteena on lisätä yrityksille omat kategoriat. Tällä hetkellä kaikki kategoriat tulevat käyttäjän puolelta.

Tulos:

Projektien käyttöliittymä saatiin korjattua ilman suurempia ongelmia. Viestien lähettäminen taustajärjestelmän kautta jäi vain tutkimuksen tasolle. Näillä näkymin tätä tarkoitusta varten ei ole suoraa pakettia vaan siihen liittyy kolmannen osapuolen palvelu.

Kategorioiden lisääminen yrityksille päätettiin hoitaa periytymisen kautta. Kategoriat yrityksillä ja käyttäjillä ovat muuten identtisiä, mutta yrityskategoriat tunnistetaan yrityksen sarjanumeron kautta, kun taas käyttäjien kategorioilla ei tätä ole. Tämä tosin aiheutti muutamia ongelmia ja osa olemassa olevasta koodista jouduttiin kirjoittamaan uudelleen.

Taustajärjestelmään lisättiin vain uudet tietokantamallit yrityskategorioita varten, mutta kaikkia tietokannan manipulaatiofunktioita ei saatu valmiiksi.

19.8.

Tavoite:

Saada yritysten kategoriat valmiiksi.

Tulos:

Taitamattomuus periytymisen kanssa aiheutti jonkin verran hukattua aikaa. Useissa paikoissa tehtiin tarpeettomia tai myöhemmin ristiriitaisia muuttujia, kun ei ymmärretty, että projektikategoria kategorian lapsena voidaan lisätä kategoriamuuttujaan. Näiden virheiden korjaamiseen ja CRUD-kutsujen sovittaminen käyttöliittymään vei suurimman osan päivästä. Projektikategoriat saatiin kuitenkin lisättyä onnistuneesti. Loput päivästä käytettiin lisäämään näitä uusia kategorioita sovelluksen muistio-osioon. Lisäys saatiin suoritettua, mutta sen yhteydessä ilmeni uusia virheviestejä, joita ei saatu ratkottua.

20.8.

Tavoite:

Saada yrityskategoriat toimimaan muistio-osion kanssa. Tämä toimii periaatteessa, mutta sen toiminta ei ole vielä aukotonta.

Tulos:

Yrityskategorioiden lisääminen muistio-osioon onnistui, kun sen toiminnasta vastaavaa koodia alettiin ymmärtää ja tiedossa olevat välittäjävirheet saatiin korjattua. Suurimmaksi ongelmaksi muodostui sitkeä virheviesti, joka ilmeni aina kun sovelluksen latasi uudestaan, ne muistiot. joihin oli lisätty, kategoria heittivät puuttuvan entiteetin virheen. Melko pitkän virheenmetsästyksen jälkeen saatiin selville, että kun yrityskategorian lisää muistioon, se tallentuu tietokantaan ja tuohon muistioon käyttäjän kategoriana. Tämä johti siihen, ettei muistion kategoriaa löytynyt yrityskategorioiden joukosta, joka aiheutti puuttuvan entiteetin virheviestin.

Tämän virheen aiheuttajaa etsittiin pitkään, mutta sitä ei löydetty käyttöliittymän koodista. Tämän jälkeen päätettiin tarkistaa taustajärjestelmän funktiot ja syy virheelle löydettiin.

Taustajärjestelmä suoritti erillisen kategorian tallennuksen muistioon taustajärjestelmässä, jota ei ollut päivitetty ottamaan huomioon yrityskategorioiden olemassaoloa. Tämän korjaamisen jälkeen kaikki toimi niin kuin piti.

Yleisen tutkinnan aikana huomattiin, että jos kategorian poistaa, joka on liitetty muistioon, tuon muistion aukaiseminen aiheuttaa virheviestin. Loppu päivä yritettiin ratkaista tätä ongelmaa ja se saatiinkin osin ratkaistua, mutta ratkaisuun vaadittavien muutosten jälkeen käyttöliittymä ei enää toiminut oikein. Tätä ongelmaa ei saatu ratkaistua ja päivän lopussa koodiin oli tehty niin monia muutoksia, että muutokset päätettiin peruuttaa ja aloittaa seuraavana päivänä alusta jonkinlaisella uudella ratkaisulla.

21.8.

Tavoite:

Ratkaista muistioiden ja kategorioiden välillä eilen huomattu virhe. Uutena tavoitteena on selvittää miksi huolto-ohjelmien pdf-kääntäjä ei toimi kaikissa tilanteissa.

Tulos:

Suurin osa päivästä kului muistioiden ja kategorioiden välisen ongelman ratkaisemiseen. Ongelma saatiin ratkaistua, kun luovuttiin ajatuksesta, että vanhempi koodi tulisi pitää mahdollisimman muuttumattomana. Yrityksestä huolimatta ei saatu kategorioiden poistamista toimimaan ilman kuolleen valkoisen ruudun ilmestymistä. Kun koodin toimintaa muutettiin merkittävästi tutumpaan muotoon, ongelma saatiin ratkaistua. Valkoinen kuolemanruutu ilmestyi muutamaan otteeseen muutoksien jälkeenkin, mutta se saatiin loppumaan, kun ylimääräisten ponnahdusikkunoiden sulkeminen poistettiin.

Toinen tavoite ei ollut kovin vaikea ratkaista kiitos kattavan virheentunnistuksen pdf-kääntäjissä. Virhe johtui siitä, että päivämääriä käsiteltiin väärin.

22.8.

Tavoite:

Muuttaa ohjelman meno-osion kategoriat käyttämään yritysten kategorioita. Toisena tavoitteena on aloittaa lisäapplikaation kehittäminen, jonka tarkoitus on lähettää viestejä mobiililaitteesta toiseen ja tallentaa nuo viestit ohjelman käyttämään tietokantaan, jotta niitä voidaan tarkastella myöhemmin.

Tulos:

Kategorioiden muuttaminen ei vaatinut muutaman muuttujan nimen vaihdosta suurempia päivityksiä koodiin.

Lisäapplikaatio saatiin alulle. Aluksi tutkittiin internetistä, millaisia mahdollisuuksia Flutter:issa on viestien lähettämiseen. Tälle löydettiin jonkinlainen paketti, joka vaikutti tarkoitukseen sopivalta. Applikaation tarkoitus on pystyä lähettämään viestejä ohjelman käyttäjän tai yrityksen nimissä, ilman että jouduttaisiin turvautumaan kolmannen osapuolen API-järjestelmiin, joita on lähes pakko hyödyntää, jos haluaa lähettää viestejä tietokoneella. Tämän tavoitteen saavuttamiseksi applikaation pitää pystyä tunnistamaan käyttäjät ja täten hyödyntämään samaa

kirjautumislogiikkaa kuin itse pääohjelmakin. Tästä syystä päätettiin aloittaa siitä, että sivuapplikaatio yhdistettäisiin pääapplikaation käyttämään taustajärjestelmään ja että käyttäjä voisi kirjautua sivuapplikaatioon pääapplikaation käyttäjän tiedoilla.

Sivuapplikaation toteutuksessa saatettiin käyttää pitkälti samaa logiikkaa ja samoja komponentteja kuin pääohjelmassakin. Joitain ominaisuuksia pitää poistaa, kuten rekisteröinnin mahdollisuus ja mahdollisuus muokata käyttäjän tietoja. Tässä ohjelman kopioinnissa ilmeni kuitenkin joitain tunnistamattomia ongelmia. Applikaatio jätti muutamien lisäysten jälkeen käyttäjän mustaan ruutuun, eikä vastannut komentoihin. Tästä ei ilmennyt virheviestiä. Ei tiedetä, mistä tämä johtuu.

23.8.

Tavoite:

Lisätä viestiapplikaatioon sisäänkirjautumisjärjestelmä ja yhdistää applikaatio taustajärjestelmään. Tämän lisäksi on tavoitteena lisätä applikaatioon yksinkertainen käyttöliittymä ja siihen navigaatio sivujen välillä.

Päivän aikana uutena tavoitteena tuli ratkaista, miksi pääapplikaation yrityspuolella käyttäjien lisääminen ei toimi.

Tulos:

Käyttäjien lisäämisen toimimattomuus johtui siitä, ettei tyhjää arvoa käyttäjässä otettu huomioon. Tämän korjaaminen ei tuottanut ongelmia. Samalla huomattiin joitain puutteellisia ratkaisuja järjestelmän tavassa käsitellä välittäjiään. Sen sijaan, että järjestelmä olisi käyttänyt kuluttajia, se käytti muuttujia, joka saattaa johtaa odottamattomiin käyttöliittymän päivitysongelmiin, kun käyttöliittymä on vahvasti riippuvainen välittäjän tarjoamasta datasta. Tämä korjattiin samalla.

Sivuapplikaation virhe, joka eilen havaittiin, saatiin korjattua. Kyse oli yhden komponentin puutteesta ohjelman alkurakenteessa. Sisäänkirjautumisjärjestelmä kopioitiin pääapplikaatiosta suurimmaksi osaksi. Joitain osia koodista jouduttiin muuttamaan, jotta sisäänkirjautuminen toimisi uudessa ympäristössä. Samaa strategiaa käytettiin melko monissa osissa sivuohjelman pohjustusta. Tämä teki tavoitteiden saavuttamisesta helpompaa ja vaivattomampaa, mutta virheitä kohdatessa se aiheutti ongelmia. Koska koodi on kopioitua, niin sitä ei täysin ymmärretä, ellei käytetä aikaa sen läpi lukemiseen. Tätä ei ollut tällä kertaa tehty, joten virheiden korjaaminen vaikeutui, koska koodi oli suurilta osin tuntematonta. Jotta tulevaisuudessa koodin korjaaminen ja muokkaaminen olisi helpompaa, tulee tutustua tarkemmin koodiin. Tällaisten virheiden takia tavoitteiden saavuttaminen vei odotettua kauemmin. Odottamattomia ongelmia ei kuitenkaan kohdattu.

Tämä päivä osoitti miten vähän loppujen lopuksi tiedetään Flutter-applikaation luomisesta tyhjästä. Uusia ominaisuuksia osataan luoda vanhojen päälle, mutta täysin uuden applikaation luomisesta ei tiedetä paljoa. Perusasiat täytyy opetella uudelleen, jotta tulevista haasteista voitaisiin selviytyä paremmin.

26.8.

Tavoite:

Luoda käyttökelpoinen käyttöliittymä viestiapplikaatiolle. Tämän lisäksi yritetään lähettää SMS tekstiviesti Flutter:in kautta.

Tulos:

Ensimmäiseen tavoitteeseen päästiin. Viestiohjelmassa on nyt toimiva käyttöliittymä viestine vastaanottajien valikoimiseen ja viestien lähettämiseen. Tämän luominen osoitti, etteivät Flutter:in käyttöliittymän luomisen perusteet olleet vielä tarpeeksi selkeitä. Kun kaksi container-komponenttia asetettiin sisäkkäin, niin sisempi otti kokonsa suoraan uloimmalta. Tämä aiheutti aluksi hämmennystä, mutta syy sille oli melko selkeä. Jos sisemmän komponentin paikkaa suhteessa ulompaan ei ole määritelty, ei Flutter tiedä missä tuon komponentin tulisi näkyä, joten se jättää asetetut kokomuuttujat huomiotta. Tämä saatiin korjattua antamalla sisemmälle komponentille tieto sen paikasta.

Viestiä ei saatu lähetettyä. Tälle toiminnolle on Flutter:issa muutamia eri kirjastoja, mutta ne joita ehdittiin kokeilla eivät joko toimineet uudemmassa Flutter:issa tai uudemmassa Kotlin:issa. Täytyy joko jatkaa etsimistä, tai yrittää kirjoittaa omat funktiot, joilla lähettää viestejä kännykän kautta. Jälkimmäistä pyritään välttämään, koska se on täysin tuntematon teknologian haara ja täten aikaa vievin ratkaisu.

27.8.

Tavoite:

Lähettää viesti viestiohjelman kautta. Tätä varten täytyy joko löytää valmis kirjasto, joka tekee tämän mahdolliseksi, tai selvittää miten tälle voidaan kirjoittaa omat funktiot.

Tulos:

Kesti melko kauan ennen kuin löydettiin toimiva kirjasto viestien lähettämiselle. Useimmat kirjastot toimivat vanhentuneilla Dart-versioilla, joten niitä ei voitu hyödyntää. Tätä yritettiin korjata päivittämällä ohjelman riippuvaisuuksia alempiin versioihin, mutta se ei ratkaissut ongelmaa, ja Dart:in alentamista ei pidetty järkevänä, koska se saattaisi sulkea pois joitain muita käytössä olevia kirjastoja tai Flutter:in ominaisuuksia. Lopulta löydettiin kirjasto, joka toimi vaaditulla Dart-versiolla. Tämän jälkeen itse viestin lähettäminen ei ollut kovin vaikeaa. Kirjaston avulla se onnistui yhdellä funktiolla. Tämän jälkeen aloitettiin viestiohjelman varsinainen yhdistäminen taustajärjestelmään, jotta viestejä voitaisiin lähettää pääohjelman kautta. Tätä ei saatu valmiiksi.

28.8.

Tavoite:

Yhdistää pääsovellus tekstisovelluksen kanssa ja lähettää muistio tekstiviestinä.

Tulos:

Yhdistäminen tapahtuu sovellusten käyttämän taustajärjestelmän kautta. Koska molemmat sovellukset ovat yhteydessä samaan taustajärjestelmään, voivat ne kommunikoida toistensa kanssa sen kautta. Jotta viestisovellus voisi hyödyntää samoja luokkia kuin pääsovelluskin suoraviivaisin ratkaisu oli kopioida pääohjelman koodi tekstiohjelmaan tarpeen mukaan. Tämä ei ollut niinkään vaikeaa kuin aikaa vievää. Useimmissa tapauksissa yhden pääohjelman osan kopiointi johti usean

muun osan kopioimiseen, koska monet osat pääohjelmassa ovat riippuvaisia toisistaan. Joissain paikoissa tätä pystyttiin estämään. Viestiohjelmassa ei tarvita kaikkia pääohjelman toiminnallisuuksia ja täten esimerkiksi jotkin luokkien muuttujat voidaan jättää huomiotta ja muuttaa dynaamisiksi. Täten ne toimivat edelleen taustajärjestelmän kanssa, mutta niiden vastakappaleita ei tarvitse lisätä viestiohjelmaan. Joitain funktioita voitiin myös poistaa tarpeettomina. Esimerkiksi välittäjien päivitysfunktioilla ei tehdä viestitellessä mitään. Viestisovellus ei tule koskaan muokkaamaan juuri mitään datapankin tiedostoja. Yhteys saatiin muodostettua. Nyt pääsovelluksen tallentamat tekstiviesteihin päästään käsiksi tekstisovelluksella. Täten pääsovelluksen kontakteja voidaan myös hyödyntää tekstisovelluksessa.

Tämän lisäksi saatiin toimimaan taustajärjestelmästä haettujen viestin lähettäminen. Viestit voidaan nyt hakea manuaalisesti listana ja tuon listan sisältö voidaan ajaa viestintäfunktioiden läpi. Koska aikaa näiden töiden jälkeen jäi, päätettiin viedä loppuun edellisinä päivinä aloitettu manuaalisen viestinnän työkalu. Tällä työkalulla on tarkoitus voida valita joukko vastaanottajia ja kirjoittaa viesti, joka lähetetään kaikille vastaanottajille. Tälle oli tehty pohja jo aiemmin, joten se ei vaatinut muuta kuin yhdistää viestintäfunktio käyttöliittymää ohjaavaan logiikkaan.

29.8.

Tavoite:

Lisätä erotus yritysten tekstiviestien ja käyttäjän tekstiviestien välille ja luoda pohja vistien lähettämisen automatisoinnille.

Tulos:

Tällä hetkellä viestisovellus hakee kaikki tekstiviestit datapankista ottamatta huomioon mikä käyttäjä on ne sinne alun perin tallentanut. Tämä täytyy korjata, mutta myös yritysten lähettämät viestit on erotettava käyttäjästä, vaikka jokainen yritys toimiikin käyttäjänsä alaisuudessa.

Tämä onnistui lisäämällä tekstiviestiolioon muuttujat käyttäjälle ja yritykselle, joiden sisällön avulla viestin lähettäjä voidaan tunnistaa. Koska viestisovellus hyödyntää samaa kirjautumisjärjestelmää kuten pääsovellus voidaan viestisovelluksen käyttäjän tunnistamisessa käyttää samaa logiikkaa kuten pääsovelluksessakin. Tämän ympärille piti vain kopioida yrityksen hakemislogiikka pääsovelluksesta. Toisaalta tämän logiikan ympärillä on paljon näennäisesti tarpeellista tavaraa, mikä ei ole olennaista tekstisovelluksessa. Tämän karsimiseen sellaiseksi, että se edelleen toimisi taustajärjestelmän kanssa meni huomattava määrä aikaa.

Automatisointi android-käyttöjärjestelmällä on aihealueena tuntematon, joten tekstiviestien lähettäminen automaattisesti alkoi jälleen tutustumisella aiheen mahdollisuuksiin. Melko nopeasti löydettiin Flutter:in kanssa yhteensopiva kirjasto, jolla voidaan luoda funktioita, jotka toimivat mobiililaitteen toimintojen taustalla, vaikka itse sovellus ei olisikaan välittömästi käynnissä. Automaatioon asti ei vielä päästy mutta kirjaston toiminta saatiin varmistettua aktivoimalla yksittäinen toiminto, joka sai puhelimen lähettämään ilmoituksen muutaman minuutin sisällä. Tämä on kirjaston yksi haittapuolista. Se jättää tehtävien suorittamisen ajankohdan käyttöjärjestelmän päätettäväksi, joten yhden tehtävän suorittamiseen voi kulua useita minuutteja. Tämä tekee ohjelman testaamisesta todella hidasta ja epäselvää. Aina ei voi tietää onko kyse ohjelmointivirheestä vai siitä että käyttöjärjestelmä on päättänyt käyttää tehtävän suorittamiseen enemmän aikaa.

30.8.

Tavoite:

Saada automatisoitu viestien lähettäminen toimimaan.

Tulos:

Tätä varten edellisenä päivänä käyttöön otetusta kirjastosta löytyy oma funktionsa, mutta sen käyttöönotto ja virheentarkistus osoittautuivat haasteeksi ja aikaa vieväksi. Virheentarkistukseen liittyy sama ongelma kuin yksinkertaisiin komentoihin. Ei voi tietää varmasti, kuinka pitkä aika kuluu ennen kuin taustatehtävä suoritetaan. Tämän lisäksi toistuvissa tehtävissä, joita automatisoinnissa aiotaan hyödyntää, minimiaikaväli eri tehtävien välillä on 15 minuuttia, joka lisää odotteluaikaa huomattavasti.

Toistuvien tehtävien aktivoiminen ei ollut suurikaan haaste, mutta niiden yhdistäminen viestiohjelman muun logiikan kanssa oli. Ensimmäinen ongelma oli saada lähetettävien viestien sisältö muotoon, jota voidaan käyttää taustatehtävän funktioissa. Taustatehtävän mahdollistavan kirjaston funktioiden välillä voidaan siirtää vain merkkijonoja, kokonaislukuja, totuusarvomuuttujia ja näiden listoja. Tämä tarkoittaa sitä, että viestin sanoma ja viestin vastaanottajat pitää jotenkin muuttaa edellä mainittuun muotoon ja lähettää eteenpäin map-luokan avulla, joka on myös kirjaston funktioissa vaadittu parametri. Tälle yritettiin muutamaa eri lähestymistapaa, kuten jättää data lähettämättä ja ajaa funktiot, jotka hakevat tiedon tietokannasta taustafunktiossa, mutta tämä ei ollut mahdollista. Taustajärjestelmän kanssa keskustelemisen mahdollistavat funktiot vaativat BuildContext-luokan muuttujaa ja sitä ei saatu taustafunktion käyttöön. Toinen vaihtoehto, jota yritettiin, oli kasata kaikista viesteistä merkkijonojen lista. Ensin listassa olisi itse viesti ja sitten joukko puhelinnumeroita, joille ne tulisi lähettää. Jos puhelinnumerot voitaisiin erottaa viesteistä, tiedettäisiin aina, milloin uusi viesti alkaa tässä listassa. Tämä olisi voinut toimia, mutta kesken tämän ratkaisun ohjelmoimisen keksittiin mahdollisesti helpompi ratkaisu. Koska taustafunktio vaatii parametrinään Map-luokan, voidaan tuon luokan rakennetta käyttää viestien erottelemiseen. Map-luokan yksilö kasattiin siten, että viesti toimi tunnisteena ja vastaanottajien lista arvona. Tämä toimi ja data saatiin liikkumaan taustafunktioon. Tämän jälkeen taustafunktioon saatettiin yhdistää viestin lähettävä funktio. Tämä ei aiheuttanut virheviestiä, mutta ei myöskään toiminut. Muutamia eri korjauksia testattiin, mutta viestiä ei saatu lähetettyä.