

Joona Heinikari, Juho Montonen

CASE : FUNCTION APP, SUUNNITTELU JA TOTEUTUS

LAB-ammattikorkeakoulu
Liiketalous Lappeenranta
Digitradenomi

Opinnäytetyö 2020

Tiivistelmä

Joona Heinikari, Juho Montonen

Case : Function App, suunnittelu ja toteutus 58 sivua, NN liitettä

LAB-ammattikorkeakoulu

Liiketalous Lappeenranta

Digitradenomi

Opinnäytetyö 2020

Ohjaajat: lehtori Jouni Könönen, LAB-ammattikorkeakoulu

Opinnäytetyön tarkoitus oli suunnitella ja kehittää alustava versio sovelluksesta, joka auttaa henkilöitä arkisissa asioissa. Sovelluksen tehtävänä on muistuttaa käyttäjää seuraavasta arjen askareesta, esimerkiksi kouluun lähdöstä ajoissa. Jokaisella käyttäjällä on oma aikataulu, jonka hän on itse tai muiden avustuksella suunnitellut.

Sovellus toteutettiin hyödyntämällä monia eri teknologioita. Kehityksessä käytetyt teknologiat olivat tekijöille suurimmilta osin ennestään tuttuja sekä kehittäjien yleisessä suosiossa olevia tunnettuja ohjelmistokehyksiä. Sovellus käyttää relaatio-tietokantaa ja jakaa tietoa .NET ohjelmistokehyksellä kehitetystä RESTful periaatteiden mukaisesta ohjelmointirajapinnasta. Sovellusta käytetään kahden käyttäjäliittymän kautta, toinen aikataulun luomiseen ja muokkaamiseen, toinen muistuttamaan asiakasta hänen seuraavista askareistaan.

Sovelluksen suunnittelu tapahtui tilaajan kanssa yhteisissä palavereissa sekä sähköpostitse. Suunnittelun sekä muun kartoituksen jälkeen sovelluksen kehitystyö pystyttiin aloittamaan suhteellisen tiukalla aikataululla. Sovelluksen alustava versio saatiin kuitenkin valmiiksi aikataulussa, jonka jälkeen ensimmäiset käyttäjät pääsivät testaamaan sovellusta.

Sovelluksen jatkokehitystä pohdittiin yhdessä tilaajan kanssa testausvaiheen aikana ja sen jälkeen. Asiakkailta saatiin onnistuneesti hyvää palautetta ja tietoa sovelluksen tilasta sekä toimivuudesta testausjakson aikana.

Asiasanat: Käyttäjäliittymä, ohjelmointirajapinta, ohjelmistokehys, sovellus

Abstract

Joona Heinikari, Juho Montonen
Function App, 58 Pages, Number of Appendices
LAB University of Applied Sciences
Business Administration Lappeenranta
Degree Programme in Business Information Technology
Bachelor's Thesis 2020
Instructor(s): Mr Jouni Könönen, Lecturer

The objective of the project was to design and develop an initial version for application that helps people on their daily routines. The mission of the application is to remind users about their next tasks, for example taking the bus to school on time. Every user has their own schedule that has been planned by themselves with or without help from someone else.

This application was developed with many different technologies. All of the technologies and frameworks in this application technology stack are familiar for the developers and popular among the software development community. This application uses a relational database to store data. That data is shared through an API that was developed using .NET Core Framework and complies with RESTful design principles. This application is used with two different user interfaces, one to build schedules and one to remind user about their current tasks.

This application was designed together with the client in meetings and via email. After the planning phase the development started with quite tight schedule. The initial version was ready just in time so the first testers could start testing the application.

Further development was discussed about with the client during and after the development period. Some useful insights from the testers were gained about the application during the first testing phase.

Keywords: User Interface, API, framework, application

Sisällys

| | | |
|--------|--|----|
| 1 | Johdanto | 7 |
| 2 | Arkkitehtuuri | 8 |
| 2.1 | Nykyaikaiset internet-sovellukset | 8 |
| 2.2 | Käyttöliittymät | 8 |
| 2.2.1 | Käyttäjät | 9 |
| 2.2.2 | Käyttöympäristö | 9 |
| 2.2.3 | Käytettävyys | 9 |
| 2.2.4 | Äänensävy ja tekstin muotoilu | 10 |
| 2.2.5 | Yhdenmukaisuus | 10 |
| 2.3 | REST-pohjainen web-rajapinta | 11 |
| 2.3.1 | Asiakas-palvelin | 11 |
| 2.3.2 | Kerroksittainen järjestelmä | 11 |
| 2.3.3 | Tilattomuus | 11 |
| 2.3.4 | Yhtenäinen rajapinta | 12 |
| 2.3.5 | Välimuisti | 12 |
| 2.3.6 | Ladattava koodi | 12 |
| 3 | Toteutus : Case FunctionApp | 13 |
| 3.1 | Vaatimusmäärittely | 13 |
| 3.1.1 | Yleiskuvaus | 13 |
| 3.1.2 | Asiakas | 14 |
| 3.2 | Käyttötapaukset | 15 |
| 3.3 | Menetelmät | 16 |
| 3.3.1 | Ketterä kehitysmalli | 17 |
| 3.3.2 | Versiohallinta | 17 |
| 3.4 | Tekninen toteutus | 17 |
| 3.4.1 | Android | 17 |
| 3.4.2 | .NET Core Framework | 18 |
| 3.4.3 | Angular | 19 |
| 3.5 | Työkalut | 19 |
| 3.5.1 | Android Studio | 19 |
| 3.5.2 | Visual Studio | 20 |
| 3.5.3 | JetBrains Webstorm | 21 |
| 3.5.4 | Microsoft SQL Server Management Studio | 21 |
| 3.6 | Testaus | 22 |
| 3.7 | Järjestelmän osa-alueet | 23 |
| 3.8 | Tietokanta | 24 |
| 3.9 | Tietomalli | 24 |
| 3.10 | Ohjelmointirajapinta | 24 |
| 3.10.1 | .NET Core Framework | 25 |
| 3.10.2 | EF Core Framework | 25 |
| 3.10.3 | Taustajärjestelmän toteutus | 25 |
| 3.10.4 | Swagger | 29 |
| 3.10.5 | Postman | 31 |
| 3.11 | Natiivisovellus | 32 |
| 3.11.1 | Natiivisovelluksen konfigurointi | 32 |
| 3.11.2 | Natiivisovelluksen rakenne | 34 |
| 3.11.3 | Käyttöliittymät | 36 |
| 3.11.4 | Retrofit | 40 |

| | | |
|--------|--|----|
| 3.11.5 | Notifikaatiot | 42 |
| 3.11.6 | Testaus, luvat ja asennus | 46 |
| 3.12 | Angular web-käyttöliittymä | 47 |
| 3.13 | Julkaisu..... | 48 |
| 4 | Julkaisu sovelluskauppaan | 49 |
| 5 | Testaus | 55 |
| 5.1 | Asiakkaat | 56 |
| 5.2 | Käyttäjäkokemukset ja jatkokehitys | 56 |
| 5.2.1 | Käyttäjäkokemukset | 56 |
| 5.2.2 | Jatkokehitys | 57 |
| 6 | Pohdinta..... | 57 |
| | Lähteet..... | 60 |

Keskeisiä käsitteitä

Android – käyttöjärjestelmä, jolle järjestelmän mobiilikäyttöliittymä kehitettiin

Angular – ohjelmistokehys, jolla toteutettiin järjestelmän nettikäyttöliittymä

Käyttöliittymä – käyttäjälle näkyvä osa sovelluksesta

.NET (dotnet) – Sovelluksen taustajärjestelmän kehitykseen käytetty ohjelmistokehys

Notifikaatio – ilmoitus/hälytys, joka näkyy käyttäjän puhelimessa

Ohjelmointirajapinta – rajapinta, jonka avulla ohjelmat voivat keskustella tai jakaa tietoa keskenään

Ohjelmistokehys – abstraktio, joka nopeuttaa ohjelmistokehitystä tarjoamalla geneerisiä uudelleenkäytettäviä toiminnallisuuksia

Ohjelmointiympäristö – ohjelma tai ohjelmat, joita käytetään ohjelmiston suunnitteluun ja toteutukseen

1 Johdanto

Projektissa toteutettu järjestelmä suunniteltiin ja toteutettiin yhteistyössä eräälle Lappeenrannan Business Millin kanssa yritykselle. Järjestelmän tarkoituksena on toimia apuvälineenä elämänhallinnan kanssa vaikeuksissa oleville henkilöille. Loppukäyttäjän sovellus muistuttaa käyttäjää ennalta määrätyistä jokapäiväisistä askareista, ja kannustaa niiden hoitamiseen. Sovelluksen avulla saadaan esimerkiksi dataa siitä, kuinka luotettavasti käyttäjä suorittaa askareitaan, tekeekö hän ne vai jäävätkö ne tekemättä.

Järjestelmän avulla voidaan myös laskea tukihenkilöiden työtaakkaa, ja tarjota uudenlaista dataa loppukäyttäjän kehityksestä. Sovelluksen avulla voidaan karottaa käyttäjän arjen ongelmakohtia ja tarjota tukea niistä selviytymiseen. Sovelluksen avulla voidaan myös seurata käyttäjän kehitystä ja sen mukaan muokata käyttäjän aikataulua, kun tiettyihin asioihin ei enää esimerkiksi uppoa niin paljon aikaa kuin ennen.

Sovelluksella oli yksi kilpailija, joten sovelluksen kehitysvaiheessa ei voitu aikailla vaan sovellus oli saatava nopeasti testauskuntoon. Tämän raportin kirjoitushetkellä suoria tai epäsuoria kilpailijoita näyttäisi olevan useampikin. Monet näistä sovelluksista on tehty myös opinnäytetyönä alun perin.

Tämän sovellusversion kehitystyö alkoi helmikuussa 2020 kahden opiskelijan työpanoksella. Sovellukselle oli aikaisemmin tuotettu toinen versio saman ryhmän opiskelijoilla. Aikaisempi versio jäi kuitenkin hyvin aikaiseen vaiheeseen, eikä sitä voitu hyödyntää tämän järjestelmän kehitykseen.

2 Arkkitehtuuri

Tässä luvussa käydään läpi projektiin liittyviä asioita ja käsitteitä teoreettisesta näkökulmasta. Kappale sisältää asioita käyttöliittymien suunnittelusta sekä ohjelmointirajapintojen kehityksestä.

2.1 Nykyaikaiset internet-sovellukset

Nykyisiltä nettisovelluksilta vaaditaan enemmän kuin koskaan aikaisemmin niin käyttäjämäärien kuin vaatimuksien osalta. Nettisovelluksien on oltava saatavilla 24 tuntia vuorokaudessa sekä niitä pitäisi pystyä käyttämään millä laitteella tahansa käyttöjärjestelmästä tai ruutupäätteen koosta sekä muodosta riippumatta. Nettisovelluksien tulee olla tietoturvallisia, skaalautuvia sekä tarpeeksi notkeita selviytyäkseen niihin kohdistuvista suurien käyttäjävirtojen aiheuttamista piikeistä (Characteristics of Modern Web Applications 2019).

Näistä vaatimuksista selviytyäkseen täytyy sovelluksen olla hyvin suunniteltu. Nettisovellus on yleensä isännöity johonkin pilvipalveluun, jolloin ainakin skaalautumiseen tarvittavia resursseja on saatavilla eikä laskentavoima lopu kesken. Sovelluksille voi olla myös kirjoitettu testejä, joilla varmistetaan esimerkiksi yksittäisten komponenttien tai palvelujen toimivuus automaattisesti (Characteristics of Modern Web Applications 2019).

Taustajärjestelmiin kohdistuvia vaatimuksia voidaan siirtää myös käyttäjien laitteille. Esimerkiksi perinteiset sovellukset ovat saattaneet turvautua palvelimeen, joka hoitaa kaiken aina navigaatiosta tietokantakutsuihin asti, jonka jälkeen se palauttaa valmiin nettisivun laitteelle näytettäväksi. Nykyään tätä vastuuta on siirretty yhä enemmän päätelaitteiden vastuulle, jolloin laite itse esimerkiksi renderöi näytettävän sivun, jolloin taustajärjestelmä lähettää päätelaitteelle vain sen tarvitseman datan (Characteristics of Modern Web Applications 2019).

2.2 Käyttöliittymät

Suunnittelimme kaikki sovellusjärjestelmän käyttöliittymät itse. Tässä luvussa käsitellään käyttöliittymien suunnitteluun liittyvää teoriaa, ja tärkeimpiä suunnitteluvaiheissa huomioitavia asioita.

2.2.1 Käyttäjät

Käyttöliittymää suunniteltaessa ensisijainen huomio tulee aina olla käyttäjässä. Hyvä tapa toteuttaa tätä ajatusta, on pohtia mitä käyttäjä haluaa tehdä missäkin vaiheessa sovelluksen käyttöä. Käyttöliittymä tulee suunnitella niin, että kyseinen tehtävä pystytään suorittamaan mahdollisimman helposti ja vaivattomasti. (ustwo.)

Myös demograafiset tekijät, kuten esimerkiksi käyttäjien ikä tai sukupuoli, ja psykograafiset tekijät, kuten esimerkiksi arvot ja elämäntyyli kannattaa ottaa käyttöliittymän suunnittelussa huomioon. Esimerkiksi moderni, paljon eleitä ja typografiaa hyödyntävä käyttöliittymä voi olla nuoren käyttäjäkunnan mieleen, kun taas vanhemmalle väestölle perinteisempi ja tutumpi käyttöliittymä ilman alan uusimpia ominaisuuksia on usein helpommin omaksuttava. (ustwo.)

2.2.2 Käyttöympäristö

Sovelluksen käyttöympäristö ei tarkoita vain alustaa jolla sovellusta tullaan käyttämään, vaan myös käyttötapaa. Esimerkiksi älytelevisioissa käytettävissä sovelluksissa täytyy ottaa täysin eri asioita huomioon kuin mobiilisovelluksissa — televisiota katsotaan huomattavasti kauemman matkan päästä, niitä käytetään lähes aina vain sisätiloissa, ja niitä operoidaan kaukosäädintä käyttämällä. Muun muassa kaikki nämä asiat tulee ottaa huomioon, kun suunnitellaan sovelluksen ulkonäköön liittyviä asioita, kuten tekstin kokoa, sovelluksen värejä, tai kontrasteja. (ustwo.)

2.2.3 Käytettävyys

Käytettävyys on tärkeä asia sovelluksen käyttökokemusta, ja sillä on merkitystä kaikille sovelluksen käyttäjille. Käytettävyyttä suunnitellessa muistetaan kyllä usein perinteiset käytännöt kuten helppokäyttöisyys ja käyttöliittymän selkeyttäminen, mutta erilaisia toimintarajoitteita omaavat käyttäjät unohdetaan kuitenkin usein. (ustwo.)

Sovelluksen käyttöä vaikeuttavia toimintarajoitteita on olemassa neljää päätyyppiä: näkörajoitteet, kuulorajoitteet, kosketukseen liittyvät rajoitteet, ja kognitiiviset

rajoitteet. Kognitiivisilla rajoitteilla viitataan kykyyn käsitellä kolmen edellä mainitun aistin tuntemuksia. (ustwo.)

Koska käyttäjillä voi olla rajoitteita tiettyjen aistien kanssa, sovelluksen käyttö ei saa olla riippuvainen vain yhdestä aistista. Tämän sijaan sovelluksen tulee mahdollisuuksien mukaan hyödyntää useampaa aistia samanaikaisesti. Työkaluja tämän toteuttamiseksi ovat esimerkiksi mahdollisuus puheen muuntaminen tekstiksi näköhäiriöistä kärsiviä käyttäjiä varten, tai tekstitykset kuulohäiriöisiä varten. (ustwo.)

2.2.4 Äänensävy ja tekstin muotoilu

Tekstit ja typografia ovat yhtä tärkeitä elementtejä sovelluksen käyttöliittymän suunnittelussa, kuin hienot kuvat ja väritkin. Hyvin kirjoitetut ja muotoillut tekstit eivät tee sovelluksesta vain helposti ymmärrettävämpää, vaan antavat sille myös persoonallisuutta äänensävyyn kautta. Sovelluksen tekstejä suunniteltaessa kannattaa muistaa, että käyttäjätkin ovat ihmisiä, eivät koneita. Heille kannattaa siis myös puhua kuin ihmisille. Kun tekstit suunnitellaan ihmisen luettaviksi, käyttäjä muodostaa helpommin positiivisen tunneyhteyden sovellukseen, joka parantaa tuotteen käyttökokemusta. (ustwo.)

2.2.5 Yhdenmukaisuus

Yhdenmukaisuutta pidetään yhtenä käytettävyyden kulmakivistä. Totuttujen käytäntöjen noudattaminen auttaa käyttäjää yhdistämään sovelluksen muihin käyttämiinsä tuotteisiin, ja käyttökokemus tuntuu luonnolliselta heti ensimmäisestä kerrasta lähtien. (ustwo.)

Ihmiset yhdistävät esimerkiksi tietyt värit ja muodot automaattisesti eri toiminnallisuuksien kanssa. Vihreä väri ja valintamerkki tarkoittavat usein jotakin positiivista, kun taas punainen väri ja risti merkitsevät päinvastaista. Keltainen väri ja kolmio taas yhdistetään usein varoituksiin, kun taas sininen väri ja ympyrä informaatioon. Näitä normeja rikkomalla voidaan helposti aiheuttaa turhaa sekaannusta. (ustwo.)

2.3 REST-pohjainen web-rajapinta

RESTful rajapintojen periaatteet vaikuttavat tehokkuuteen, skaalautuvuuteen, yksinkertaisuuteen, yhteensopivuuteen, viestintään, komponenttien siirrettävyyteen sekä luotettavuuteen. Nämä periaatteet voidaan kiteyttää kuuteen rajoitteeseen, jotka ohjaavat RESTful rajapintojen suunnittelua. (Reynders 2018, 5)

2.3.1 Asiakas-palvelin

Asiakas-palvelin rajoitteella saadaan eristettyä käyttäjä tai käyttöliittymä taustajärjestelmästä, jolloin ne ovat riippumattomia toisistaan. Tällöin käyttöliittymä lähettää taustajärjestelmälle kyselyn ja taustajärjestelmä vastaa lähettämällä kysytyt tiedot sisältävän vastauksen. Tämä keskustelu käydään taustajärjestelmän rajapinnan kautta. Erottamalla käyttöliittymä ja taustajärjestelmä toisistaan mahdollistaa niiden kehittämisen erillään niin pitkään, kun niiden väliseen rajapintaan ei tule rikkovia muutoksia. (Reynders 2018, 6)

2.3.2 Kerroksittainen järjestelmä

Asiakas-palvelin rajoitteen lisäksi RESTful järjestelmissä on yleensä kerroksittainen rakenne. Kerroksittainen rakenne estää kajoamisen järjestelmän alempiin kerroksiin suoraan rajapinnan kautta. Järjestelmän jakaminen kerroksiin parantaa uudelleenkäytettävyyttä sekä tekee järjestelmästä modulaarisemman, joka puolestaan helpottaa järjestelmän kehittämistä tai muuta muokkaamista. (Reynders 2018, 6)

2.3.3 Tilattomuus

Tilattomuus on asiakas-palvelin rajoitteen perusta. Asiakkaan ja palvelimen välinen liikenne tulee olla tilatonta, eli jokaisen kutsun mukana täytyy tulla kaikki tarvittava tieto kutsun ymmärtämiseen ja toteuttamiseen. Asiakasohjelma on käytännössä vastuussa koko session tilan hallinnoimisesta, eikä se voi luottaa siihen, että palvelin muistaisi jotain niiden välisestä liikenteestä. Tämä ei kuitenkaan tarkoita, että aivan kaikki asiakkaan ja palvelimen väliseen tilaan liittyvä pitäisi aina kulkea jokaisessa pyynnössä mukana. Tila voidaan säilöä jonnekin muualle, jolloin sille voidaan kutsua jollakin tunnisteella esimerkiksi tilaa luettaessa tai päivitettäessä. (Reynders 2018, 6)

2.3.4 Yhtenäinen rajapinta

REST-rajapintojen avainominaisuus on yhtenäinen rajapinta. Tämä rajoite koostuu neljästä osa-alueesta, jotka ovat vastuussa resurssien tunnistamisesta, resurssien muokkaamisesta, itseselitteisillä vastauksilla sekä tilan hallinnoimisesta. (Reynders 2018, 6)

Yhtenäinen rajapinta mahdollistaa sen, että usea eri asiakasohjelma voi hyödyntää tätä rajapintaa sekä sen tarjoamia resursseja. (Reynders 2018, 6)

2.3.5 Välimuisti

Välimuisti-rajoite johtuu tilattomuusrajoitteesta ja vaatii, että palvelimelta saapuvat vastaukset sisältävät tunnisteiden, joka kertoo asiakasohjelmalle voiko vastauksen tallentaa välimuistiin myöhempää käyttöä varten. Vastauksen tallentaminen välimuistiin parantaa nopeutta ja latenssia, koska vastaus voidaan hakea välimuistista palvelimen sijaan. Välimuistia voidaan käyttää kummallakin puolella, eli asiakasohjelmassa sekä palvelimella. (Reynders 2018, 6)

2.3.6 Ladattava koodi

JSON- ja XML-resurssien lisäksi selain voi ladata koodia palvelimelta ja suorittaa sen selaimessa. Tämän vapaaehtoisen rajoitteen avulla asiakasohjelmasta voidaan tehdä yksinkertaisempi, koska sen ominaisuuksia voidaan karsia. Hyviä esimerkkejä voisi olla ladattava JavaScript-komponentti, selain lataa komponentin koodin palvelimelta ja sen jälkeen suorittaa sen selaimessa. (Reynders 2018, 7)

3 Toteutus : Case FunctionApp

Tässä luvussa käydään läpi, miten ohjelmistoprosessia lähdettiin viemään eteenpäin. Luvussa kuvataan kevyesti projektin vaatimusmäärittely sekä käyttötapaukset, joiden jälkeen tutustutaan projektin tekniseen toteutukseen.

Teknisessä osiossa käydään läpi projektin toteutus sekä projektissa käytettyjä teknologioita. Tarkoituksena on kertoa ymmärrettävästi, mitä jokainen sovelluksen osa-alue tekee, miten se on toteutettu ja mikä on sen tarkoitus. Osa-alueet on kuvattu suurin piirtein samassa järjestyksessä kuin ne ovat toteutettu itse projektissa.

3.1 Vaatimusmäärittely

Tähän järjestelmään kohdistui erilaisia vaatimuksia monilta eri tahoilta, mutta suurin osa näistä vaatimuksista kohdistui käytettävyyteen asiakkaiden näkökulmasta ainakin ensimmäisen version osalta. Muita vaatimuksia olivat esimerkiksi tietoturva-asiat sekä sovelluksen toimivuus esimerkiksi ilman internet-yhteyttä. Seuraavaksi käydään läpi järjestelmän vaatimusmäärittely lyhykäisyydessään eri vaatimusten osalta.

3.1.1 Yleiskuvaus

Ohjelmistoa käyttävät asiakkaat sekä heidän avustajansa. Ohjelmiston asiakkailla on mahdollisia puutteita arjen hallinnassa, ja avustajat auttavat asiakasta hänen arjessaan luomalla henkilökohtaisen päiväohjelman arjen tueksi. Asiakkaat käyttävät ohjelmistoa omilla älypuhelimillaan ja avustajat luovat päiväohjelmia sekä seuraavat asiakkaan kehitystä nettisovelluksen kautta. Mobiilisovellus ohjaa asiakasta antamalla palautetta ja visualisoimalla onnistumisen sekä edistymisen. Avustajat voivat seurata asiakkaiden toimintaa sovelluksen käytöstä kertyneen datan avulla. Järjestelmän tarkoitus on siis lyhykäisyydessään:

- auttaa jäsentämään asiakkaan toimintaa tarkoituksenmukaisesti
- aikatauluttaa sekä ohjata asiakkaan toimintaa muistutusten sekä hälytysten avulla
- tukea asiakkaan taitojen automatisoitumista harjoittelun avulla

- motivoida asiakasta antamalla palautetta ja mahdollistamalla onnistumisia
- tukea asiakasta tilanteissa, joissa hän ei osaa toimia

3.1.2 Asiakas

Ohjelmistoa käyttävät asiakkaat, jotka tarvitsevat tukea arjen hallinnassa. Asiakasryhmän luonteen takia sovelluksen tulee olla helppokäyttöinen. Helppokäyttöisyyteen vaikuttavat monet asiat sovelluksessa, joten sovellusta kehitettäessä on pyritty huomioimaan seuraavanlaisia asioita.

Selkokieliisyys

Sovelluksen tekstisisältö on pyritty kirjoittamaan mahdollisimman ymmärrettävästi sekä selkokielellisesti. Turhan pitkiä selostuksia tai muita ohjetekstejä on pyritty laatimaan mahdollisimman helposti luettaviksi ja ymmärrettäviksi, eikä sovelluksen käyttäminen edellytä pitkän ohjekirjasen lukemista.

Käyttöliittymät

Käyttöliittymiä suunniteltaessa täytyi ottaa huomioon kohderyhmä sekä heidän tarpeensa. Käyttöliittymien tuli olla selkeitä sekä helppokäyttöisiä. Käyttöliittymissä ei myöskään saanut olla niin sanottuja ”turhia ärsykeitä”, ja niiden tuli olla mahdollisimman pelkistettyjä. Esimerkiksi tietyt värit tai muodot saattoivat aiheuttaa käyttäjissä ei-toivottua käytöstä tai yllättäviä reaktioita riippuen päivästä.

Sisältö

Sovelluksen sisältöä laadittaessa pyrittiin kiinnittämään huomiota eri elementtien väreihin ja muotoihin. Tietyt värit tai muodot voivat aiheuttaa neuropsykologisia oireita kärsivässä käyttäjässä ei-toivottuja reaktioita tai päähänpistoja, jotka voivat taas vaikuttaa muuten asiakkaan toimintaan. Sovelluksen muun sisällön kuten tekstien sekä ohjeiden tuli olla myös helposti ymmärrettäviä. Sovelluksessa hyödynnettiin papunetin kuvapankkia ja muu sisältö saatiin asiakkailta, ohjaajilta sekä tilaajalta.

Huolenaiheena oli se, miten hälytyksistä saataisiin tarpeeksi voimakkaita, että ne saisivat käyttäjän huomion, vaikka hänellä olisikin jotain muuta meneillään. Tämä oli myös yksi syy, jonka vuoksi sovellus päätettiin toteuttaa natiivina Android-sovelluksena, jolloin toivottiin saatavan enemmän valtaa laitteeseen, jossa sovellus on.

Tietosuoja

Sovelluksen täytyi myös olla tietoturvallinen sekä oli huomioitava tiedonkäsitteelyyn liittyviä asioita. Sovellus onnistuttiin kuitenkin pitämään vielä niin kevyenä, että siihen ole pakollista tallentaa käyttäjiä yksilöivää tietoa, kuten nimeä tai osoitetta (Tietosuoja 2020).

3.2 Käyttötapaukset

Ohjelmistoa käyttävät asiakkaat, joilla on arjen hallinnassa puutteita, sekä heidän avustajansa (vanhemmat, terapeutit ja valmentajat).

Järjestelmän käyttötarkoitus on helpottaa henkilöiden (asiakkaiden) toimintaa, joilla on arjen hallinnassa puutteita. Asiakkaiden valmentajat luovat tietokoneella asiakkaalleen henkilökohtaisen päivä- tai viikko-ohjelman arjen tueksi. Järjestelmä hälyttää/muistuttaa äänellä, kuvalla ja tekstin avulla asiakasta tehtävistä, jotka hänen ohjelmaansa on kirjattu. Asiakkaan tulee kuitata tehtävä suoritetuksi, jonka jälkeen hän saa kannustavan palautteen. Tehtävät on ajastettu ja priorisoitu.

Jos jokin tehtävä jää suorittamatta, hälytyksiä/muistutuksia annetaan tehtävän prioriteetin mukaan useampi kappale. Kaikki yksittäiseen tehtävään liittyvät tapahtumat, kuten esimerkiksi hälytykset ja kuittaukset, kirjataan ja tallennetaan tietokantaan.

Avustajan käyttötapaus

Avustajat määrittelevät/suunnittelevat päiväohjelman asiakkaalleen yhdessä asiakkaan kanssa. Tämä tehdään tietokoneella web-käyttöliittymää käyttäen.

Avustaja luo asiakkaalleen tehtävälisan, esimerkiksi aamurutiinit. Tehtävälisa koostuu pienemmistä osista, kuten esimerkiksi aamurutiinien kohdalla peseytymisestä, hampaiden pesusta, vaatteiden pukemisesta, ja aamupalan syömisestä. Jokaiselle osalle määritellään tehtävän nimi, aputeksti, prioriteetti, ja tehtävän suorittamiselle varattu aika. Tehtävälisalle määritellään tehtävälisan nimi ja alkuajankohta.

Tehtävälisan ensimmäisen tehtävän ilmoitus tulee tehtävälisan alkuajankohdaksi annetun päivämäärän ja ajan mukaan, ja seuraavien tehtävien ilmoitukset tulevat sitä mukaan, kun edelliselle tehtävälle varattu aika on kulunut umpeen.

Asiakkaan käyttötapaus

Asiakkaat suunnittelevat päiväohjelmansa yhdessä avustajansa kanssa. Avustaja lisää luodun ohjelman järjestelmän tietokantaan, ja asiakas asentaa natiivisovelluksen puhelimeensa.

Ensimmäisellä käynnistyskerralla sovellus kysyy käyttäjältä tarvittavat oikeudet, jonka jälkeen asiakas kirjautuu sovellukseen sisään. Kun sovellukseen on kirjautettu onnistuneesti sisään asiakas voi sulkea sen. Käyttäjä pysyy sisään kirjautuneena, kunnes sovelluksessa käydään manuaalisesti kirjautumassa ulos.

Asiakkaan sovellus antaa hälytyksiä avustajan kanssa suunnitellun ohjelman pohjalta. Sovelluksen lähettämää ilmoitusta painaessa avautuu ikkuna, jossa asiakas voi pyytää lisäohjeistusta, tai merkitä tehtävän hylätyksi tai suoritetuksi.

3.3 Menetelmät

Projektin aikana käytettiin erilaisia toimintatapoja ja menetelmiä, joita ohjelmistoprojektien läpiviennissä yleisesti käytetään. Kaikkia toimintatapoja ei kuitenkaan noudatettu välttämättä sanatarkasti projektin luonteen takia, tai niitä saatettiin muokata vielä ketterämmiksi tarpeen mukaan. Esimerkiksi projektin kaksi ohjelmistokehittäjää pysyivät hyvin ajan tasalla toistensa tekemisistä ilman kiveen hakattua ajankohtaa viikoittaiselle Scrum-tapaamiselle.

3.3.1 Ketterä kehitysmalli

Sovelluksen kehitystyöhön sovellettiin hyvin ketteriä kehitystapoja, jotka muistuttivat hyvin paljon Scrum-mallissa käytettyjä sprinttejä. Sovelluksen nykyisestä tilasta pysyttiin kartalla Kanban-tyylisellä ”boardilla”. Tämä tarkoittaa, että sovelluksen tilaa pystyi tarkastelemaan seinällä olevien tarralappujen avulla. Näissä tarralapuissa oli sovelluksen eri ominaisuuksia tai vaatimuksia sekä nämä tarralaput oli järjestelty kyseisen asian tilan mukaan. Tila saattoi olla ”todo”, työn alla, valmis tai bugi, joka tarkoittaa kyseisen ominaisuuden olevan jollain tavalla rikki.

Näitä tarralappuja tarkasteltiin ja niiden tilaa päivitettiin noin viikoittain pidetyssä palaverissa, jolloin sovelluksen uutta tilaa päästiin tarkastelemaan, mitä oli tehty ja mitä on vielä edessä. Tilan päivitys tapahtui siirtämällä tarralappuja toisesta tilaruudusta toiseen, esimerkiksi ”työn alla”-ruudusta ”valmis”-ruutuun.

3.3.2 Versiohallinta

Koko järjestelmän versiohallinta toteutettiin git-työkalulla ja GitHub-palvelulla. GitHub-palveluun on mahdollista tallettaa lähdekoodia sekä projektiin liittyvää materiaalia. Jokaiselle eri osa-alueelle luotiin oma projekti GitHubiin, jonne kyseistä osa-aluetta, esimerkiksi käyttöliittymää kehitettiin eteenpäin.

3.4 Tekninen toteutus

Sovellus päätettiin toteuttaa kehittäjille vähintään osittain tutuilla teknologioilla kehitystyön mahdollistamiseksi. Täysin uuden opettelu olisi ollut liian aikaa vievää ja kehitystyö olisi voinut venyä turhan pitkäksi. Tässä kappaleessa käydään läpi miksi ja miten kyseisiin teknologioihin on päädytty sekä kuvaus jokaisesta teknologiasta.

3.4.1 Android

Sovelluksen pääkäyttöliittymä ja toiminnallisuus päätettiin toteuttaa natiivi Android-sovelluksena erilaisten lupien ja oikeuksien käyttämisen mahdollistamiseksi. Kyseinen käyttöjärjestelmä tuntui olevan myös suosittu sovellukselle kaavaillun asiakasryhmän keskuudessa, joten se oli myös hyvä vaihtoehto.

Vaihtoehtona natiiville Android-sovellukselle olisi ollut React Native tai Xamarin-ohjelmistokehykset, mutta ne olivat kehittäjille täysin tuntemattomia. Projektin mahdollisen venymisen vuoksi koimme Android-natiivisovelluksen olevan paras vaihtoehto. Toisin kuin natiivisovellukset, ohjelmistokehykset eivät myöskään pysty hyödyntämään aivan kaikkia puhelimen ominaisuuksia. Yksi tällaisista ominaisuuksista on esimerkiksi koko puhelimen näytön haltuun ottavat ilmoitukset tai hälytykset. Koska hälytykset ovat keskeinen osa käyttäjäsovelluksen toimintaa, päädyimme valitsemaan kehitysympäristöksi natiivisovelluksen. (Google Developers 2020)

3.4.2 .NET Core Framework

Sovelluksen taustajärjestelmä päätettiin toteuttaa Microsoftin .NET Core -ohjelmistokehyksellä sen ollessa kehittäjille hyvin tuttu ja ominaisuuksiltaan sopiva ohjelmistokehys.

Sovelluksen ohjelmointirajapinnan tehtäviin kuului tiedon välittäminen järjestelmän eri osien välillä sekä datan muuntelu tai keräily. Tällaisia tehtäviä olivat esimerkiksi salasanojen muuntaminen ei-selkokieliseen muotoon tai erilaisten tietoliikenteiden muodostaminen raakadatasta. Tärkein tehtävä ohjelmointirajapinnalle on kuitenkin hakea ja tallentaa tietoa tietokantaan sekä lähettää sitä eteenpäin.

Taustajärjestelmän kehitykseen käytetty .NET on Microsoftin kehittämä ilmainen avoimen lähdekoodin alusta, jolla voidaan kehittää erilaisia sovelluksia aina Windows-työpöytäsovelluksista älykelloille tarkoitettuihin sovelluksiin (Microsoft 2020).

Taustajärjestelmä toteutettiin aluksi .NET Core 2.2-versiolla, mutta se päivitettiin kehitystyön lomassa silloin uunituoreeseen 3.0-versioon. Uuden version mukana tuli muun muassa tuki C#-kielen 8.0-versiolle (Microsoft 2020).

Vaihtoehtona olisi voinut olla JavaScriptillä kehitettävä NodeJS, mutta se oli täysin tuntematon kehittäjille eikä sen ominaisuuksia tutkittu sen tarkemmin .NET Core-ohjelmistokehyksen ollessa kehittäjien suosiossa.

3.4.3 Angular

Sovelluksen toinen käyttöliittymä toteutettiin Angular-ohjelmistokehyksellä nettisovelluksena. Tämän käyttöliittymän tehtävänä oli auttaa käyttäjiä suunnittelemaan ja luomaan aikatauluja, joiden mukaan natiivisovellus sitten ohjailee käyttäjää myöhemmin.

Angular on Googlen kehittämä avoimen lähdekoodin ohjelmistokehys, jolla voidaan suunnitella ja kehittää toimivia SPA-sovelluksia. Angular-sovelluksia kehitetään käyttämällä TypeScript-ohjelmointikieltä.

Angularille vaihtoehtona olisi ollut ReactJS-kirjasto, joka on erittäin suosittu kirjasto nettisovellusten kehittäjien keskuudessa. Päädyimme kuitenkin valitsemaan nettisovelluksen kehitysympäristöksi Angularin, koska hallitsimme sen henkilökohtaisesti parhaiten.

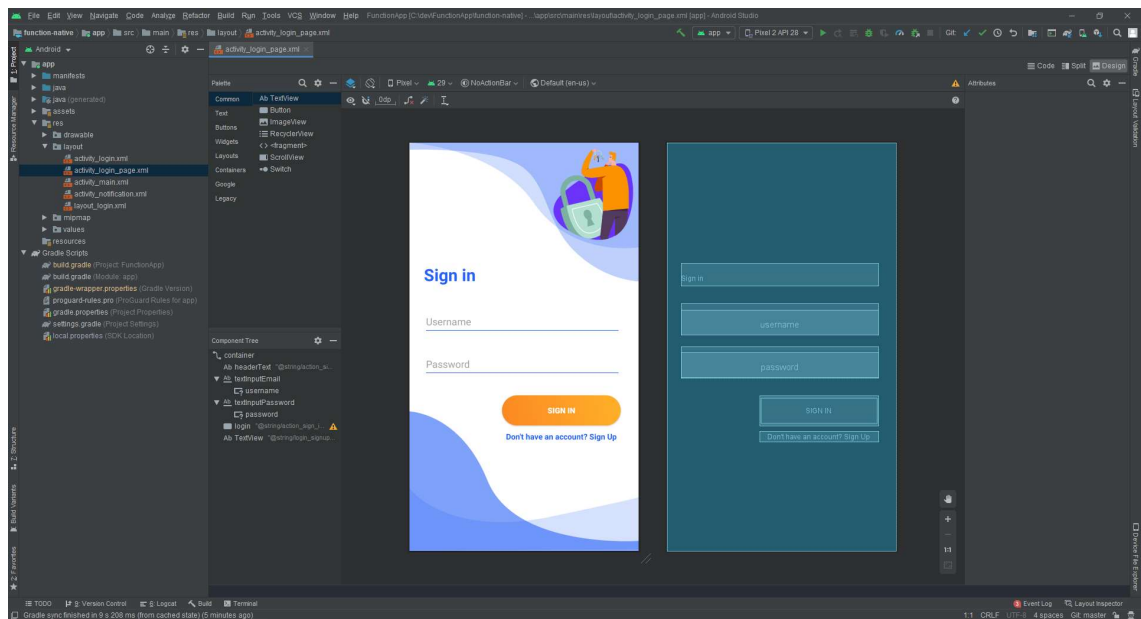
3.5 Työkalut

Sovelluksen eri osa-alueita kehitettiin erilaisissa ympäristöissä erilaisilla työkaluilla. Käytetyt työkalut ovat suosituimmasta päästä kyseisille teknologioille ja tuovat kehitystyötä helpottavia ominaisuuksia kielille, joille ne ovat suunniteltu ja kehitetty. Tässä luvussa kuvaus kehitystyössä eniten käytetyistä työkaluista.

3.5.1 Android Studio

Android Studio on Googlen julkaisema virallinen ohjelmointiympäristö Android-natiivisovelluksille. Android Studion ensimmäinen versio julkaistiin joulukuussa 2014, ja se toimii Windows, macOS, ja Linux -käyttöjärjestelmillä.

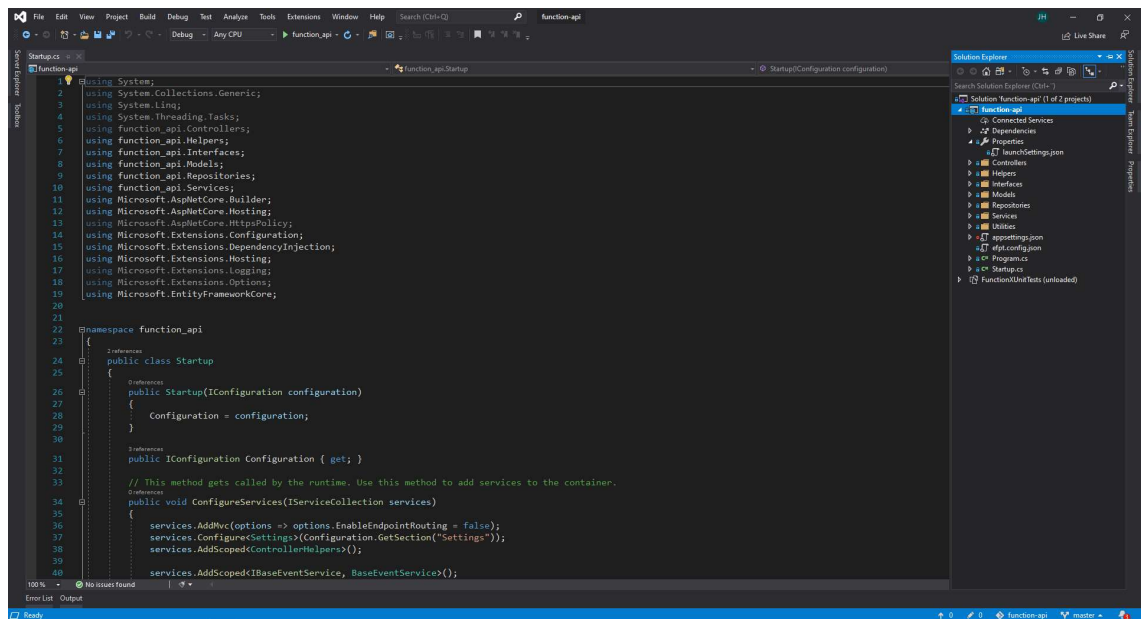
Tärkeimpiä kehitystyötä helpottavia ominaisuuksia Android Studiossa ovat älykäs tekstinsyöttöä hyödyntävä koodieditori, useita käyttöympäristöjä ja ominaisuuksia tukeva Android-emulaattori, ja käyttöliittymien toteutusta nopeuttava visuaalinen käyttöliittymäeditori (kuva 1). Käytimme Android Studiota natiivisovelluksen käyttöliittymän kehitykseen. (Android Studio 2020)



Kuva 1. Kuvankaappaus Android Studioon käyttöliittymästä

3.5.2 Visual Studio

Visual Studio on Microsoftin kehittämä ohjelmointiympäristö, jolla voidaan kehittää muun muassa tietokoneohjelmia, pelejä, verkkosovelluksia, tai mobiilisovelluksia. Ohjelmointiympäristö tukee lähes kaikkia yleisimpiä alustoja ja käyttöjärjestelmiä, ja erilaisten lisäosien ansiosta myös lähes kaikkia ohjelmointikieliä. Käytimme Visual Studiota taustajärjestelmän kehitykseen. Visual Studion käyttöliittymän näkee kuvasta 2. (Visual Studio 2019 2020)



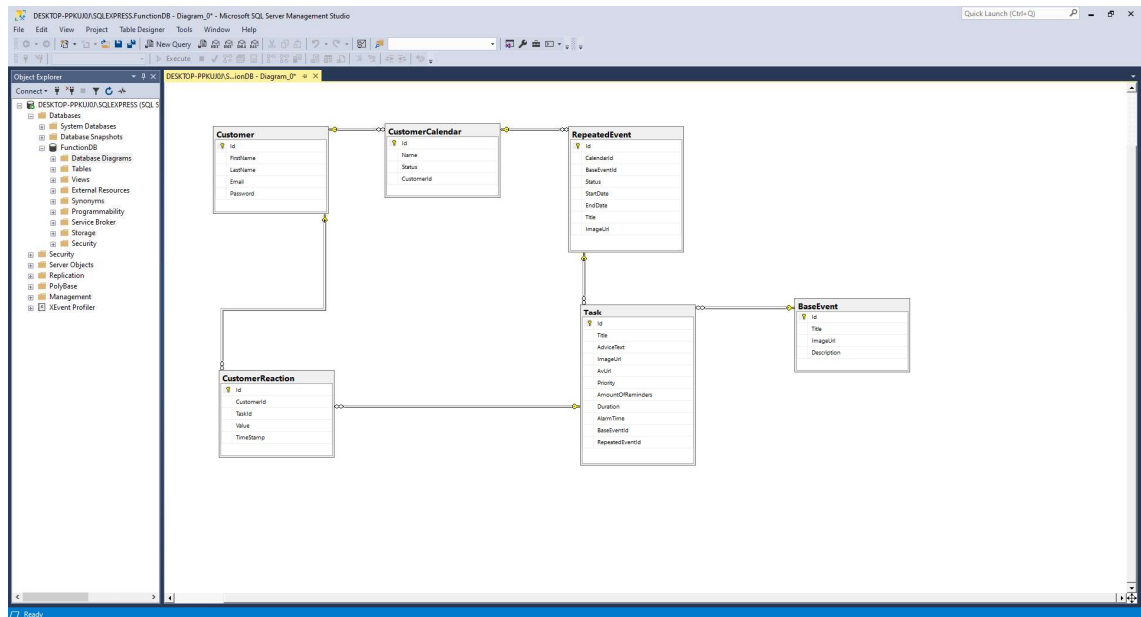
Kuva 2. Kuvankaappaus Visual Studioon käyttöliittymästä

3.5.3 JetBrains Webstorm

WebStorm on JetBrainsin kehittämä ohjelmointiympäristö, joka erikoistuu verkkopohjaisten sovellusten kehitykseen. Sovellus on maksullinen, mutta opiskelijat saavat opiskelujen ajaksi lisenssin ilmaiseksi. Käytimme WebStormia verkkopohjaisen sovelluksen käyttöliittymän toteutukseen. (Atlassian 2020)

3.5.4 Microsoft SQL Server Management Studio

Microsoft SQL Server Management Studio on hallintaympäristö erilaisille SQL-palvelimille ja tietokannoille. Hallintaympäristön kautta voidaan esimerkiksi konfiguroida asetuksia, muokata sisältöä, tai luoda komponentteja. Käytimme Microsoft SQL Server Management Studiota projektissa käytetyn tietokannan hallinnoimiseen. SQL Server Management Studion käyttöliittymän näkee kuvasta 3. (What is SQL Server Management Studio (SSMS) 2020)



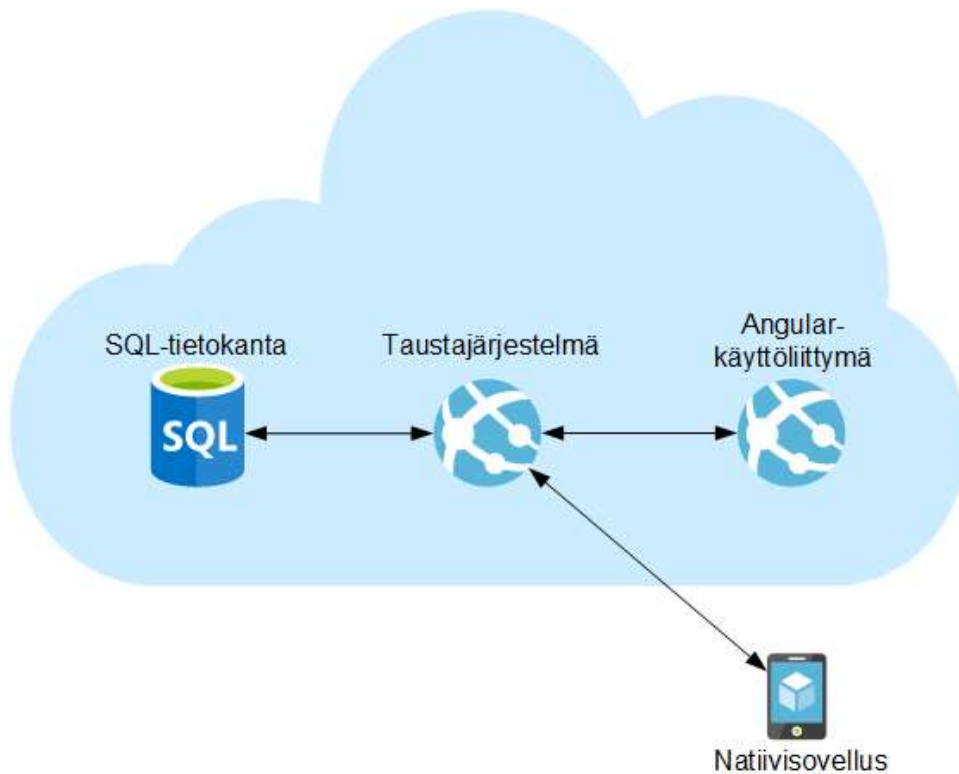
Kuva 3. Kuvankaappaus SSMS käyttöliittymästä

3.6 Testaus

Järjestelmän yksikkötestaaminen jäi kokonaan toteuttamatta ajanpuutteen takia, joten järjestelmän eri osa-alueiden testaus suoritettiin käsin kehitystyön ohessa. Testauksessa kokeiltiin järjestelmän toimivuutta sekä muutamia eri ääritapahtumia sekä esimerkiksi virheellisiä syötteitä käyttäjältä. Ohjelmointirajapinnan toimivuutta testattiin vielä erikseen Postman-työkalulla. On tärkeää testata taustajärjestelmälle lähetettäviä pyyntöjä, koska ne ovat järjestelmän toimivuuden kannalta kriittisiä.

Jokaiselle sovelluksen eri osa-alueelle oli tarkoitus kehittää automaattisia testejä, mutta niiden toteuttaminen jäi tällä erää tikeiksi jatkokehityksen saralle. Suurin osa sovelluksen logiikasta on toteutettu niin, että sille olisi mahdollista kehittää esimerkiksi yksikkötestejä.

3.7 Järjestelmän osa-alueet



Kuva 4. kuvaus järjestelmästä

Kuvassa 4 on kuvattuna järjestelmän eri osa-alueet sekä niiden suhteet toisiinsa. Kuvasta voi huomata, miten esimerkiksi käyttöliittymät eivät keskustele keskenään taikka tietokannan kanssa suoraan, vaan kaiken tiedon liikkumisen hoitaa ohjelmointirajapinta (API). Tiedon liikkumista kuvaavat kaavioon sijoitetut nuolet. Kaaviosta voi myös huomata, miten kaikki osa-alueet paitsi natiivisovellus ovat sinisen alueen sisällä. Tämä alue kuvastaa Azure-pilvipalvelua sekä sinne luotuja resursseja. Järjestelmän kaksi web-palvelua eli nettisovellus, joka toimittaa toisen käyttöliittymän roolia sekä ohjelmointirajapinta, joka vastaa tiedon välittämisestä. Myös tietokanta on sijoitettu pilveen sen tuomien etujen takia, joita ovat esimerkiksi se, että ei tarvitse huolehtia omasta palvelimesta taikka tietokannan varmuuskopioinnista.

3.8 Tietokanta

Järjestelmä suunniteltiin database-first ajattelumallilla, jolloin tietokanta ja tietorakenne toteutettiin ensimmäisenä. Tietokanta päätettiin toteuttaa relaatiotietokantana, jossa tieto säilötään tauluihin, joita voidaan yhdistellä avainten avulla, esimerkiksi rivin ID:llä.

Tietokanta perustettiin ensin lokaalisti ja siirrettiin jälkeenpäin Microsoftin Azure-pilvipalveluun. Tietokantaan luotiin tietomallin mukaisia tauluja datan säilömistä varten.

Tietokantaan ei säilötä tulenarkaa tietoa, kuten salasanoja selkokielellisesti. Järjestelmän kehityksen jatkuessa tietokantaan todennäköisesti luotaisi myös maskit, joilla muukin salainen tieto, esimerkiksi henkilötiedot voitaisiin turvata suoraan tietokannassa. Esimerkiksi Azure-pilvipalvelussa on tällainen ominaisuus valmiina. (Dynamic Data Masking 2020)

3.9 Tietomalli

Tietomallin laatiminen oli yksi projektin tärkeimmistä asioista ja sen tekemiseen käytettiin kohtuullisesti aikaa. Sovelluksen tietomallissa pyrittiin siihen, että erilaisia tapahtumia ja tehtäviä tallennettaisiin tietokantaan valmiiksi. Tällöin tapahtumia monistettaisiin niin, että niihin viitattaisiin hälytyksessä. Hälytys sisältäisi siis käyttäjän, tapahtuman sekä tiedon milloin se tapahtuu ja kauanko se kestää.

Tietomallin relaatiot mahdollistavat tiedon säilömistä eri tauluissa, jolloin yksittäisen taulun koko pystytään pitämään pienempänä. Tietoa pyrittiin hajauttamaan eri tauluihin ja tietokantaoperaatioiden määrää pyrittiin optimoimaan, jottei tietokantaan kohdistuisi liian suurta kuormaa kerralla.

Tietomallia hiottiin läpi projektin. Tällä hetkellä se kuitenkin sallii jokaisen järjestelmän tarvitseman ominaisuuden eikä sen käytössä ole ilmennyt pullonkauloja.

3.10 Ohjelmointirajapinta

Järjestelmän API eli ohjelmointirajapinta toteutettiin Microsoftin ASP.NET Core -ohjelmistokehyksellä rakentamalla internetin kautta käytettävä web-rajapinta.

Tätä rajapintaa käytetään järjestelmän datan hakemiseen ja käsittelyyn. Ohjelmointirajapinnan tehtävänä on keskustella tietokannan kanssa sekä käsitellä ja jakaa tietoa järjestelmän eri osien kanssa.

Rajapinta noudattaa REST-arkkitehtuuria ja toimii välikätenä kaikelle järjestelmän datalle. Rajapinta keskustelee käyttöliittymien eli natiivisovelluksen ja web-käyttöliittymän kanssa sekä käsittelee tietoa tietokannassa. Liikennöinti käyttöliittymien kanssa tapahtuu http-kutsuilla ja tietokantayhteys on rakennettu käyttäen EF Core-ohjelmistopakettia.

3.10.1 .NET Core Framework

Taustajärjestelmä kehitettiin hyödyntäen .NET Core ohjelmistokehystä sen tuomien ominaisuuksien vuoksi. Kehitystyö tällä ohjelmistokehyksellä sujui kuin tanssi sen tuomien kirjastojen avulla. Varsinkin taustajärjestelmän kontrollerien tekeminen on helppoa lukuisten valmiiden http-vastausten avulla, jolloin jokaista lähetettävää vastausta ei tarvitse itse kirjoittaa kokonaan otsikkotietoja myöten.

Taustajärjestelmän Startup.cs-tiedostossa konfiguroitiin taustajärjestelmän ominaisuuksia sekä esimerkiksi tietokantayhteydet. Tässä tiedostossa pystyy myös injektoimaan eri palveluja taustajärjestelmän käyttöön, jolloin niitä voidaan kutsua keskenään. Esimerkiksi Repository- sekä Service-kerrokset tuli injektoida, jotta taustajärjestelmä osaisi käyttää niitä.

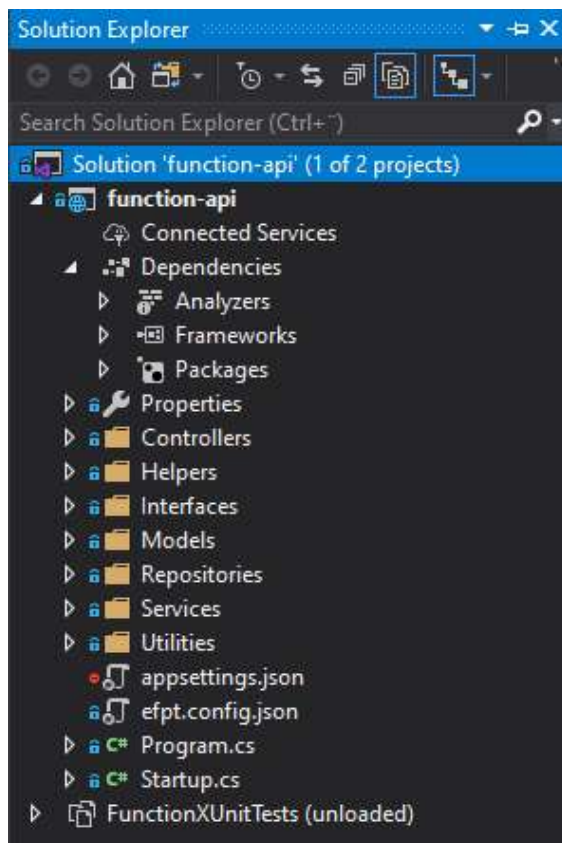
3.10.2 EF Core Framework

Taustajärjestelmän tietomalli ja tiedon käsittely toteutettiin EF Core-ohjelmistokehyksellä sekä EF Core Power Tools NuGet-paketilla. EF Core Framework huolehti tietokannan kanssa asioinnista sekä Power Toolsien avulla pystyttiin luomaan tietokannan mukaiset tietomallit taustajärjestelmään paria nappia painamalla. EF Core loi myös DbContext-luokan, jonka avulla se tietää mitä tietoa tietokannasta haetaan ja missä muodossa.

3.10.3 Taustajärjestelmän toteutus

Taustajärjestelmä suunniteltiin toteuttamaan kerroksittainen arkkitehtuuri, eli sovelluksen eri kerrokset keskustelevat keskenään erilaisten rajapintojen kanssa.

Tällainen kapselointi auttaa taustajärjestelmän jatkokehityksessä sekä mahdollisten ongelmien ratkaisussa. Toinen apu tästä on myös eri komponenttien sekä palvelujen uudelleenkäytettävyys, jolloin jotain komponenttia voidaan uusiokäyttää jossain muualla, koska sitä ei olla suunniteltu vain yhtä tiettyä tehtävää varten. Taustajärjestelmän kansiorakenne pyrittiin pitämään mahdollisimman siistinä ja suunniteltiin järjestelmän arkkitehtuurin mukaan (kuva 5).



Kuva 5. taustajärjestelmän kansiorakenne

3.10.3.1 Taustajärjestelmän kerrokset

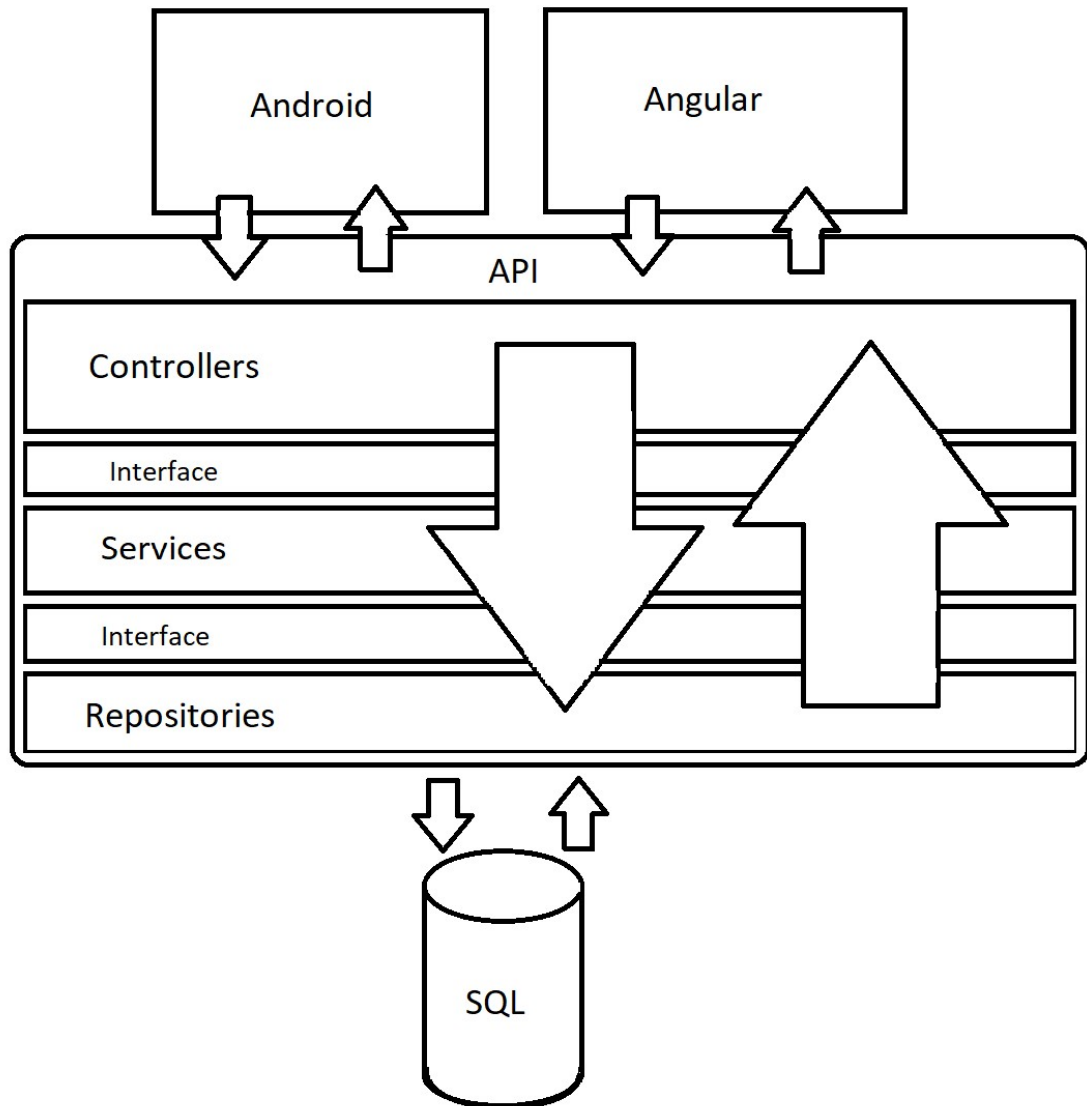
Repository-kerroksessa keskustellaan tietokannan kanssa eli haetaan, tallennetaan, päivitetään tai poistetaan tietoa sieltä. Repository-kerros pyrittiin pitämään mahdollisimman kevyenä sekä yksinkertaisena, jolloin tiedolle tehtävä käsittely hoidetaan service-kerroksessa.

Service-kerroksessa tapahtuu suurin osa rajapinnan "business"-logiikasta eli täällä tehdään monimutkaisempia operaatioita esimerkiksi tiedon yhdistämistä tai muuta käsittelyä. Esimerkkinä tällaisesta käsittelystä voi olla tiedon suodattaminen jonkun tunnisteiden avulla tai eri tietojen yhdisteleminen esimerkiksi kahden

eri tietotaulun kesken. Tämä kerros on sikäli tärkeä, että seuraavassa kerroksessa tieto pyritään lähettämään sellaisenaan, eli siinä muodossa kuin tämä kerros sen tuottaa.

Controller-kerros eli kontrollerit hoitavat keskustelun rajapinnasta asiakasohjelmiin. Tämä kerros hakee tietoa service-kerrokselta ja lähettää sen eteenpäin. Tämän kerroksen tehtävänä on huolehtia liikennöinnistä internetin välityksellä.

Kerroksien välisiä suhteita havainnollistaa kuva 6, jossa on myös kuvattuna taustajärjestelmän ja projektin muiden osa-alueiden välinen keskustelu. Kaikki taustajärjestelmässä kulkeva tieto kulkee jokaisen kerroksen läpi hyödyntäen kerroksien välisiä rajapintoja.



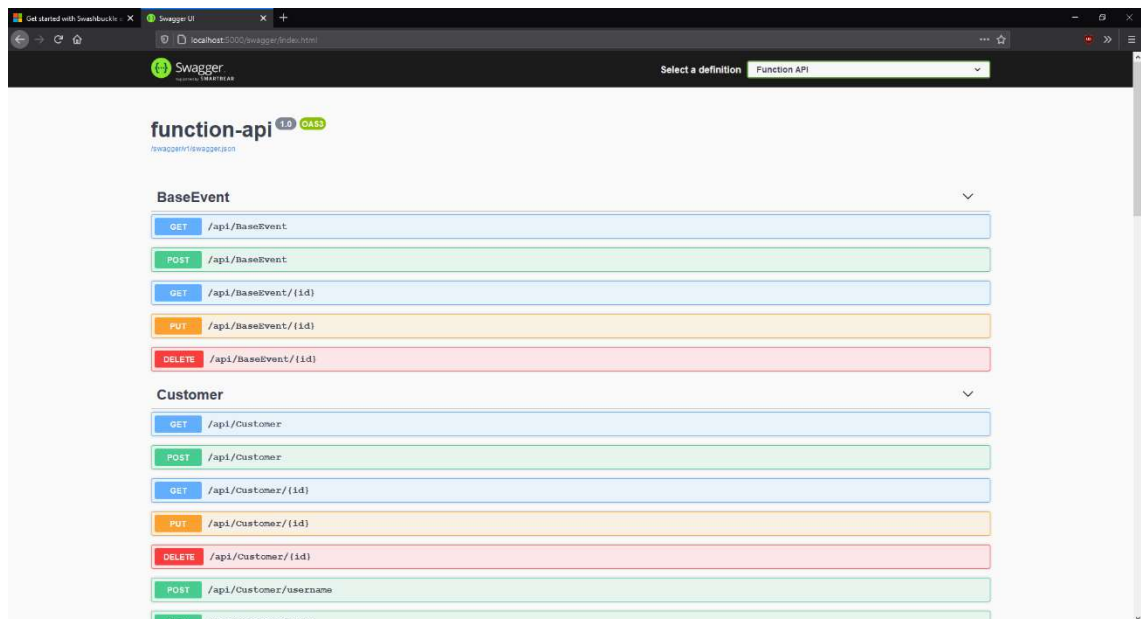
Kuva 6. Kuvaus taustajärjestelmän toiminnasta

3.10.3.2 REST-arkkitehtuuri

Taustajärjestelmä noudattaa RESTful-arkkitehtuurin rajoitteita, kuten asiakas-palvelin-suhdetta, tilattomuutta sekä kerroksittaista järjestelmää. Taustajärjestelmän ulospäin näkyvä rajapinta on suunniteltu niin, että se olisi mahdollisimman yhtenäinen.

3.10.4 Swagger

Ohjelmointirajapinta dokumentoitiin Swagger-työkalulla, joka on nykyiseltä nimeltään OpenAPI. Swagger on suosittu työkalu tukemaan ohjelmointirajapintojen kehitystä, testausta sekä dokumentaatiota. Tässä kappaleessa OpenAPIa käsitellään sen vanhalla nimellä Swagger sekä kuvataan työkalun käyttöönotto ja sen ohjelmointirajapinnasta tuottama dokumentaatio. (What Is OpenAPI?)



Kuva 7. Ohjelmointirajapinnan Swagger-kuvaus

Kuvassa 7 näkyy Swaggerin tuottama dokumentaatio, jossa näkyy ohjelmointirajapinnan eri resurssikokoelmat sekä niille luodut CRUD-toiminnot. CRUD on lyhenne englanninkielisistä sanoista "Create, Read, Update, Delete" eli kaikki toiminnot, joita datan käsittelyssä tarvitaan. Joillekin resurssikokoelmille on toteutettu myös toimintoja, jotka palauttavat haluttuja vastauksia, esimerkiksi kahden eri resurssikokoelman tietoja samassa vastauksessa tai jollain tapaa muunneltua dataa.

Kuvassa näkyy jokaisen eri toiminnon tyyppi esimerkiksi "GET" tai "POST". Tyyppin jälkeen näkyy toiminnon osoite, josta käy ilmi resurssikokoelma ja mahdollisesti tarvittava parametri, esimerkiksi "{id}". Erityyppisillä pyynnöillä tehdään erilaisia asioita, "GET"-pyynnöllä pyydetään ohjelmointirajapinnalta dataa, "POST"-pyynnöllä lähetetään ohjelmointirajapinnalle dataa. "UPDATE"-pyynnöllä halutaan yleensä päivittää olemassa olevaa tietoa, ja sen mukana toimitetaan

yleensä jokin tunnistetieto, esimerkiksi tiedon ID parametrina kohtaan "{id}". "DELETE"-pyynnöllä poistetaan dataa, sen mukana yleensä toimitetaan myös jokin tunniste tai pyyhitään suoraan koko kokoelma ilman tunnistetietoa.

Swagger tuottaa jokaiselle resurssikokoelmalle oman kohtansa dokumentaatioon, jossa näkyy kaikki toiminnot sekä niiden lisätiedot. Toimintojen lisätiedoissa näkyy esimerkki toiminnolle lähettävästä pyynnöstä ja toinen esimerkki toiminnon lähettämästä vastauksesta. Näiden lisätietojen avulla eri toimintojen testaaminen tai niiden kanssa tutuksi tuleminen helpottuu erittäin paljon. Tällaisen dokumentaation avulla eri toimintoja ei tarvitse testata itse käsin tai koittaa lähdekoodia tarkastelemalla arpoa mitä kyseinen toiminto mahdollisesti tekee taikka palauttaa.

Add and configure Swagger middleware

Add the Swagger generator to the services collection in the `Startup.ConfigureServices` method:

```
C# Copy
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<TodoContext>(opt =>
        opt.UseInMemoryDatabase("TodoList"));
    services.AddControllers();

    // Register the Swagger generator, defining 1 or more Swagger documents
    services.AddSwaggerGen();
}
```

In the `Startup.Configure` method, enable the middleware for serving the generated JSON document and the Swagger UI:

```
C# Copy
public void Configure(IApplicationBuilder app)
{
    // Enable middleware to serve generated Swagger as a JSON endpoint.
    app.UseSwagger();

    // Enable middleware to serve swagger-ui (HTML, JS, CSS, etc.),
    // specifying the Swagger JSON endpoint.
    app.UseSwaggerUI(c =>
    {
        c.SwaggerEndpoint("/swagger/v1/swagger.json", "My API V1");
    });

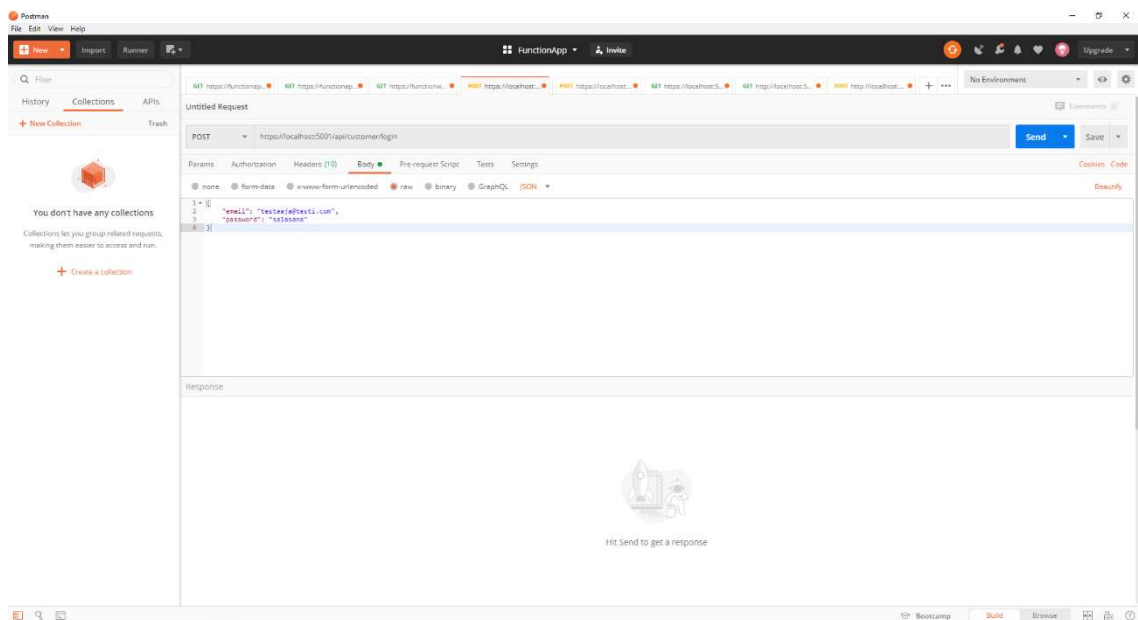
    app.UseRouting();
    app.UseEndpoints(endpoints =>
    {
        endpoints.MapControllers();
    });
}
```

Kuva 8. Swaggerin käyttöönotto .NET Core Frameworkissa

Swaggerin käyttöönotto on erittäin yksinkertaista ja sen käyttöönotto on kuvattuna Microsoftin dokumentaatioissa hyvin yksityiskohtaisesti. Käyttöönotto tapahtuu asentamalla Swagger-paketti projektin lisäosiin sekä muokkaamalla projektin konfiguraatiosta vastaavaa kooditiedostoa kuvan 8 mukaisesti. (Get started with Swashbuckle and ASP.NET Core 2020)

3.10.5 Postman

Postman-työkalua käytettiin ohjelmointirajapinnan testaamiseen lähettämällä sille erilaisia pyyntöjä. Tällä työkalulla myös tarkasteltiin taustajärjestelmän asiakassovelluksille lähettämiä vastauksia.



Kuva 9. Postman-työkalun käyttöliittymä

Kuvassa 9 näkyvässä Postman-työkalussa on näkyvillä sisäänkirjautumiskutsun tekemiseen tarvittava tieto. Kutsu lähetetään "POST"-pyyntönä taustajärjestelmään osoitteeseen */api/customer/login*. Kutsun mukana lähetetään käyttäjän mahdollisesti kirjautumisen yhteydessä syöttämät tiedot eli tunnus ja salasana. Kutsun sisältö lähetetään raakana tietona JSON-formaatissa, joka on nykypäivänä yleisesti käytetty formaatti varsinkin REST-ohjelmointirajapintojen keskuudessa (Web API design, Microsoft 2018).

3.11 Natiivisovellus

Asiakkaalle ilmoituksia lähettävä Android-natiivisovellus toteutettiin Java-ohjelmointikielellä. Yksinkertainen sovellus koostui kolmesta eri näkymästä: kirjautumisruudusta, etusivusta ja ilmoitusnäkymästä. Kehitykseen käytimme Android Studio-ohjelmointiympäristöä sekä AVD Manageria emulaattorin ajamiseen. Sovelluksen kehittäminen on suoraviivaista varsinkin käyttöliittymien osalta näillä työkaluilla, sillä työn jäljen näkee ruudulla melkein välittömästi.

Natiivisovellus on itsenäinen osa järjestelmää, joka pyytää taustajärjestelmältä dataa, jonka mukaan toimia. Sovellus on kirjoitettu melkein kokonaan omin käsin muutamien kirjaston apua lukuun ottamatta. Tärkeimmät sovelluksessa käytetyt kirjastot on kuvattu seuraavissa kappaleissa ainakin osittain.

3.11.1 Natiivisovelluksen konfigurointi

Ennen kuin sovelluksen kehitys voitiin aloittaa, täytyi sovellukselle asettaa tietyt raamit, kuten haluttu versio Android-käyttöjärjestelmästä. Muita huomioon otettavia asioita olivat esimerkiksi sovelluksen nimi, sekä alustava versionumero.

```
1  apply plugin: 'com.android.application'
2
3  android {
4      compileSdkVersion 29
5      buildToolsVersion "29.0.2"
6      defaultConfig {
7          applicationId "com.functionapp.functionapp"
8          minSdkVersion 21
9          targetSdkVersion 29
10         versionCode 3
11         versionName "1.0"
12         testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
13     }
14     buildTypes {
15         release {
16             minifyEnabled false
17             proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
18         }
19     }
20 }
21
22 dependencies {
23     implementation fileTree(dir: 'libs', include: ['*.jar'])
24     implementation 'androidx.appcompat:appcompat:1.1.0'
25     implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
26
27     implementation 'com.android.volley:volley:1.1.1'
28     implementation 'com.squareup.retrofit2:retrofit:2.7.2'
29     implementation 'com.squareup.retrofit2:converter-gson:2.7.2'
30
31     implementation 'com.android.support.constraint:constraint-layout:1.1.3'
32     implementation 'com.google.android.material:material:1.1.0'
33     implementation 'com.android.support.design:design:28.0.0'
34     implementation 'androidx.annotation:annotation:1.1.0'
35     implementation 'androidx.lifecycle:lifecycle-extensions:2.2.0'
36
37     implementation 'com.squareup.picasso:picasso:2.71828'
38
39     testImplementation 'junit:junit:4.12'
40     androidTestImplementation 'androidx.test.ext:junit:1.1.1'
41     androidTestImplementation 'androidx.test.espresso:espresso-core:3.2.0'
42 }
43
44 android {
45     compileOptions {
46         sourceCompatibility JavaVersion.VERSION_1_8
47         targetCompatibility JavaVersion.VERSION_1_8
48     }
49 }
```

Kuva 10. Build.gradle-tiedosto

Sovelluksen kääntämiseen tarvittavia asioita on määritelty *build.gradle*-tiedosto (kuva 10). Android-osiossa määritellään kaikki Androidia koskevat säännöt, esimerkiksi *compileSdkVersion* kertoo, mille Android-käyttöjärjestelmän versiolle sovellus käännetään, jotta voidaan käyttää sille kuuluvia ominaisuuksia. Sovelluksen riippuvuudet on määritelty *dependencies*-osiossa. (Add build dependencies).

```
app > src > main > AndroidManifest.xml
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3    package="com.example.functionapp">
4
5    <uses-permission android:name="android.permission.INTERNET" />
6    <uses-permission android:name="android.permission.WAKE_LOCK" />
7
8    <application
9      android:allowBackup="true"
10     android:icon="@mipmap/ic_launcher"
11     android:label="@string/app_name"
12     android:roundIcon="@mipmap/ic_launcher_round"
13     android:supportRtl="true"
14     android:theme="@style/AppTheme"
15     android:usesCleartextTraffic="true">
16
17     <receiver android:name=".receivers.CustomerReactionReceiver" >
18       <intent-filter>
19         <action android:name="com.example.functionapp.true_action" />
20         <action android:name="com.example.functionapp.false_action" />
21       </intent-filter>
22     </receiver>
23
24     <receiver android:name=".receivers.AlarmReceiver" />
25     <activity android:name=".MainActivity">
26       <intent-filter>
27         <action android:name="android.intent.action.MAIN" />
28         <category android:name="android.intent.category.LAUNCHER" />
29       </intent-filter>
30     </activity>
31
32     <activity
33       android:name=".LoginPage"
34       android:theme="@style/AppTheme.NoActionBar">
35
36     </activity>
37     <activity
38       android:name=".NotificationActivity"
39       android:theme="@style/AppTheme.NoActionBar"
40       android:exported="true">
41     </activity>
42
43     <activity android:name=".helpers.StateChecker"
44       android:exported="true" />
45   </application>
46 </manifest>
47
```

Kuva 11. AndroidManifest.xml-tiedosto

Jokaisella Android-sovellusprojektilla täytyy olla *AndroidManifest.xml*-niminen tiedosto. Tämä tiedosto kuvailee sovelluksen kannalta tärkeitä tietoja eri tahoille, kuten Androidin kääntötyökaluille, käyttöjärjestelmälle sekä Google Play-kaupalle. Monien muiden asioiden lisäksi tämän tiedoston täytyy kertoa seuraavat asiat (App Manifest Overview):

- Sovelluksen "*package name*", eli sovelluksen paketin nimi, joka yleensä on myös sama kuin lähdekoodissa käytetty nimiavaruus.
- Sovelluksen eri komponentit, kuten esimerkiksi aktiviteetit tai palvelut sekä niiden ominaisuudet.

- Sovelluksen tarvitsemat luvat päästäkseen käsiksi käyttöjärjestelmän suojattuihin ominaisuuksiin tai toisiin sovelluksiin.

- Sovelluksen vaatimat laitteisto- sekä ohjelmistovaatimukset, jotka rajoittavat esimerkiksi laitteita, joihin sovelluksen voi asentaa Google Play-kaupasta.

Esimerkiksi sovelluksen vaatimien oikeuksien käyttö pitää kuitenkin hyväksyttää käyttäjällä ensimmäisellä kerralla. Tämä sovellus tarvitsee käyttäjältä lupaa käyttää internet-yhteyttä sekä oikeutta herättää puhelin virransäästötilasta. Nämä kuvat näkyvät kuvassa 11 riveillä 5 – 6.

3.11.2 Natiivisovelluksen rakenne

Natiivisovellus käyttää kolmea eri näkymää, kirjautumissivua, päänäkymää sekä ilmoitusnäkymää. Kirjautumissivulla käyttäjä voi kirjautua sisään sovellukseen tai tehdä käyttäjätilin painamalla ”Rekisteröidy tästä”-painiketta. Kirjautumissivu näkyy, jos käyttäjä ei ole kirjautunut sovellukseen tai avaa sovelluksen ensimmäistä kertaa. Kirjautumissivulta siirrytään onnistuneen kirjautumisen jälkeen päänäkymään. Päänäkymässä sovellus synkronoi käyttäjän tiedot sekä hakee tapahtumat tai mahdolliset uudet tapahtumat kalenterista. Tapahtumien perusteella sovellus luo uusia hälytyksiä tapahtuman tehtävien perusteella. Päänäkymä aukeaa, kun sovellus käynnistetään niin, että käyttäjä on valmiiksi kirjautunut.

Kun on jonkun tehtävän aika, sovellus hälyttää sekä näyttää ilmoituksen. Ilmoitusta painamalla käyttäjä pääsee ilmoitusnäkymään. Ilmoitusnäkymästä käyttäjä voi ilmoittaa tehneensä tai jättäneensä kyseisen tehtävän tekemättä. Jos käyttäjä kohtaa tehtävän kanssa jonkun ongelmakohdan tai tarvitsee muuten apua, voi käyttäjä painaa keskimmäistä painiketta. Keskimmäinen ”tarvitsen apua”-painike näyttää käyttäjälle tehtävää varten kirjoitetun avustusviestin, jossa on apu tehtävän suorittamiseen. Tällainen apu voi olla esimerkiksi missä murot ovat tai mitä pitää muistaa ottaa mukaan kouluun.

Riippuen käyttäjän toiminnasta, sovellus näyttää joko ohjeviestin tai vastauksen käyttäjän reaktioon. Jos käyttäjä onnistui tehtävässään, sovellus näyttää kannustavan tsemppiviestin. Mikäli käyttäjä ei onnistunut tai jätti tehtävän tekemättä, so-

vellus kehottaa käyttäjää tekemään kyseisen tehtävän sekä mahdollisesti luo uuden hälytyksen riippuen tehtävän tärkeydestä. Jos käyttäjä tarvitsee apua sovellus näyttää tehtävää varten luodut ohjeet tai avustustekstin sovelluksessa.

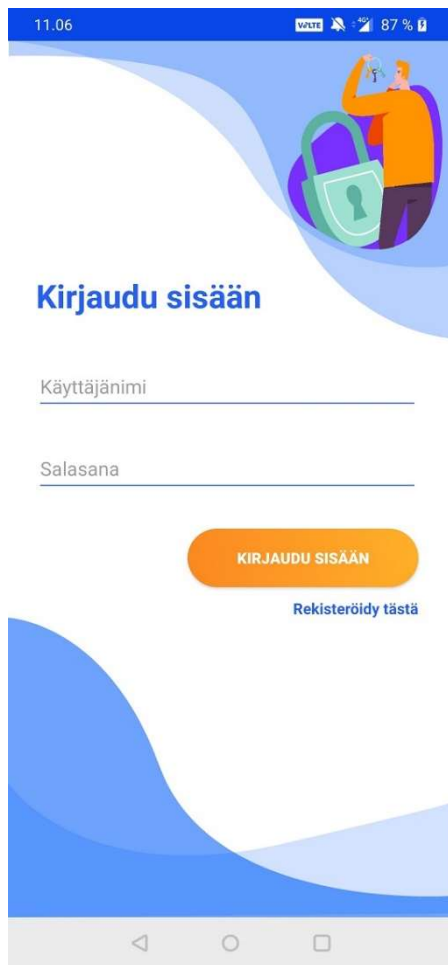
Natiivisovellus käyttää ilmoitusten sekä hälytysten tekemiseen omia ”managereitaan” eli apuohjelmia, jotka huolehtivat ilmoitusten tai hälytysten luomisesta sekä näyttämisestä. Nämä hälytykset otetaan kiinni erilaisilla vastaanottajilla, joille on annettu ohjeet, miten toimia tietynlaisten tapahtumien tapahtuessa.

Sovellus keskustelee taustajärjestelmän kanssa Retrofit-kirjaston avulla, joka lähettää erilaisia pyyntöjä taustajärjestelmän ohjelmistorajapinnalle sekä vastaanottaa sieltä takaisin saapuvia kutsuja. Natiivisovellukseen on luotu järjestelmän tietomallin mukaiset mallit, jotta sovellus tietää millaista tietoa se vastaanottaa.

Natiivisovelluksen toiminnan tukemiseksi on kirjoitettu myös erilaisia avustaja-apuohjelmia. Hyvä esimerkki tällaisesta apuohjelmasta on sovelluksen tilaa tarkkaileva koodinpätkä. Tämä apuohjelma tarkkailee sovelluksen tilaa, onko se näytöllä, taustalla vai sammutettuna. Tällainen tieto on tärkeää sovelluksen toiminnan kannalta, esimerkiksi käyttöliittymien päivityksen tai ilmoitusten lähettämisen kannalta.

Näiden ominaisuuksien ohella natiivisovellukselle on luotu erilaisia resurssitiedostoja, jotta se tietää, mitä näyttää missäkin näkymässä sekä näyttää oikean kieliset tekstit riippuen laitteen kielestä.

3.11.3 Käyttöliittymät

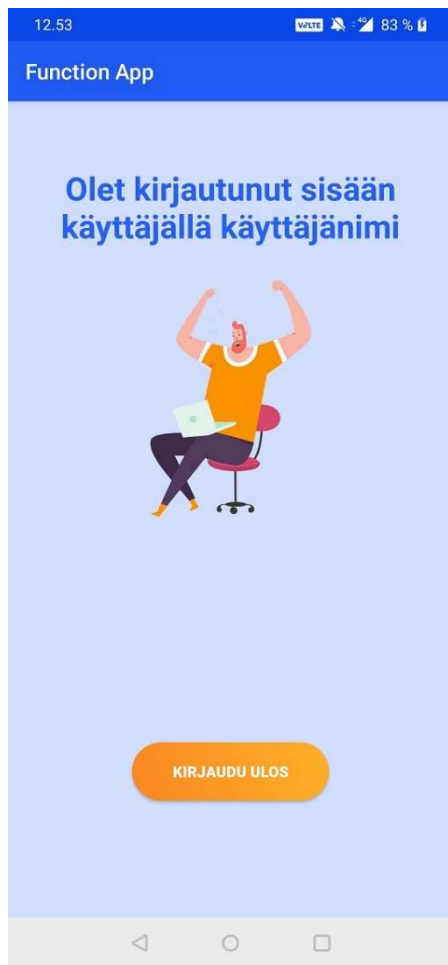


Kuva 12. Kirjautumisenäkymä

Ensimmäisellä käynnistyskerralla sovellus avautuu kirjautumisenäkymään (kuva 12). Näkymä on hyvin yksinkertainen, ja siitä löytyy taustakuvan lisäksi vain kentät käyttäjänimen ja salasanan syöttämistä varten, sekä painikkeet sisäänkirjautumiselle ja rekisteröitymissivulle siirtymiselle.

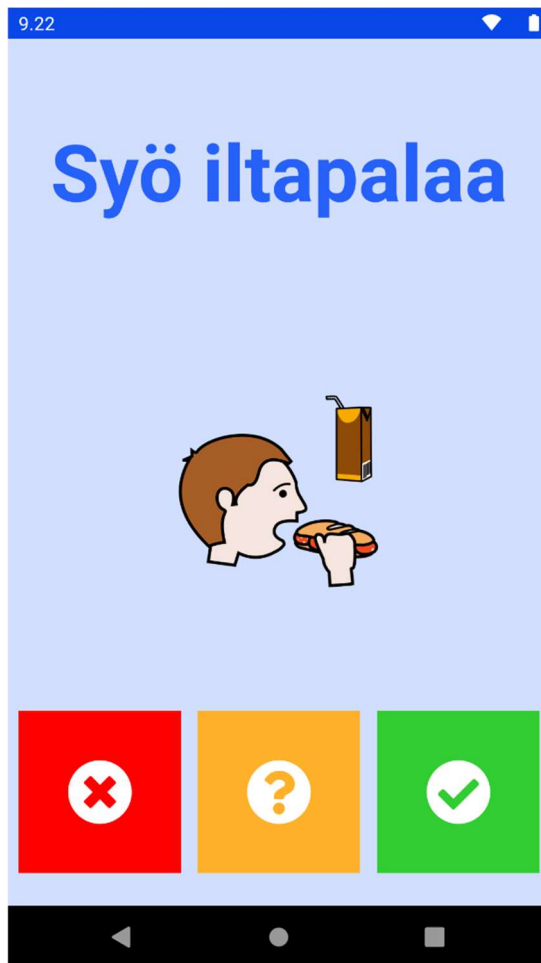
Kirjautumisenäkymän kirjautumislomake on toteutettu erillisenä xml-komponenttina ja vain liitetty sivulle, joten samaa lomaketta voidaan käyttää myös rekisteröitymisnäkymässä. Ensimmäiseen versioon varsinaista rekisteröitymissivua ei vielä tehty, vaan testikäyttäjille luotiin tunnukset valmiiksi.

Kaikki sivulla näkyvä teksti haetaan resurssikansion strings.xml-tiedostosta, ja ne kääntyvät automaattisesti puhelimen käyttöjärjestelmän kielelle.



Kuva 13. Sovelluksen päänäköymä

Onnistuneen sisäänkirjautumisen jälkeen käyttäjälle aukeaa sovelluksen etusivu (kuva 13). Sovellus kertoo tällä sivulla käyttäjälle, millä tunnuksella hän on kirjautunut sisään, ja ainoa sivun varsinainen toiminnallisuus on uloskirjautuminen. Sovellus muistaa kirjautuneen käyttäjän niin kauan, kunnes hän itse kirjautuu sovelluksesta manuaalisesti ulos, eikä kirjautumissivua enää uudelleen käynnistäessä näytetä.



Kuva 14. Ilmoitukseen reagoitaessa avautuva näkymä

Kun käyttäjä painaa sovelluksen lähettämää ilmoitusta, hänelle avautuu kuvaa 14 vastaava näkymä. Tässä näkymässä asiakas näkee muistutuksen otsikon, kuvan, ja kolme painiketta. Vihreä painike merkitsee askareen suoritetuksi ja antaa käyttäjälle positiivisen palauteviestin, punainen painike merkitsee askareen ohitetuksi ja antaa käyttäjälle kannustavan palauteviestin, ja keltaisesta painikkeesta käyttäjä saa tarvittaessa lisäohjeita askareen suorittamsta varten. Jokaisesta painalluksesta tallennetaan tieto tietokantaan, joten käyttäjien käyttäytymistä sovelluksessa voidaan analysoida.

Tukihenkilö syöttää tehtäville otsikot, kuvat, ja ohjetekstit tehtävän luontivaiheessa, joten näkymän sisältö vaihtuu sen mukaan, mitä ilmoitusta on painettu.



Kuva 15. Lisäohjeet kertova dialogi

Tehtävien lisäohjeet näytetään käyttäjälle AlertDialog-elementtinä, joka on nähtävissä kuvassa 15. Lisäopastus sisältää lyhyen ohjetekstin, ja lisäkuvan avuksi tehtävän suoritusta varten. Käyttäjä pääsee takaisin ilmoitusnäkymään joko painamalla lisäohjelaatikon ok-painiketta, tai painamalla näyttöä laatikon ulkopuolelta.

Myös vihreän ja punaisen painikkeen palautetekstit näytetään käyttäjälle tällä tavalla. Näissä tapauksissa sovellus sulkeutuu ok-painikkeen painamisen jälkeen.

Android-sovellus on kehitetty Android Studio -ohjelmointiympäristössä, ja sen ohjelmakoodi on kirjoitettu Java-ohjelmointikielellä. Käyttöliittymät on toteutettu skaalautuviksi, joten ne toimivat lähes kaikilla puhelinmalleilla näyttökoosta riippumatta. Sovelluksen kirjautumisnäkymän ja etusivun taustakuvissa on käytetty ilmaista Whoosh! Illustration Kit -nimistä kuvituspakettia ja ilmoituksissa näkyvät kuvat ovat papunetin kuvapankista.

3.11.4 Retrofit

```
1  package com.example.functionapp.retrofitapi;
2
3  import ...
4
5
6
7
8
9
10 public class RetrofitBuilder {
11     private static final String BASE_URL = "http://10.0.2.2:5000/api/";
12     private static final String AZURE_BASE_URL = "http://_____.azurewebsites.net/api/";
13
14     private static Retrofit retrofit = null;
15
16     public static Retrofit getApi() {
17         if (retrofit == null) {
18             retrofit = new Retrofit.Builder()
19                 .baseUrl(AZURE_BASE_URL)
20                 .addConverterFactory(GsonConverterFactory.create(
21                     new GsonBuilder()
22                         .setFieldNamingPolicy(FieldNamingPolicy.UPPER_CAMEL_CASE)
23                         .setDateFormat("yyyy-MM-dd'T'HH:mm:ss").create()))
24                 .build();
25         }
26         return retrofit;
27     }
28 }
29
```

Kuva 16. RetrofitBuilder

Natiivisovellus keskustelee taustajärjestelmän kanssa Retrofit-kirjaston avulla. Retrofit on Square Inc:n kehittämä kirjasto, jonka avulla natiivisovellus keskustelee taustajärjestelmän kanssa. Kuvassa 16 on kuvattuna kirjaston käyttöönotto natiivisovelluksessa Java-koodina. Kyseinen koodinpätkä luo uuden Retrofit-instanssin, jos se ei ole jo olemassa, ja kertoo taustajärjestelmän URL-osoitteen sekä ottaa käyttöön GSON-kääntäjän. (Retrofit)

```

20 public interface RetrofitService {
21
22     // GET
23
24     @GET("{path}")
25     Call<ArrayList> getArrayList(String path);
26
27     @GET("calendar")
28     Call<ArrayList<CustomerCalendar>> getCalendars();
29
30
31     @GET("calendarevents")
32     Call<ArrayList<BaseEvent>> getCalendarEvents();
33
34     @GET("customer")
35     Call<ArrayList<Customer>> getCustomers();
36
37     @GET("customer/{id}")
38     Call<Customer> getCustomer(@Path("id") int id);
39
40     @GET("customer/data/{id}")
41     Call<Customer> getCustomerData(@Path("id") int id);
42
43     @GET("customerreaction")
44     Call<ArrayList<CustomerReaction>> getCustomerReactions();
45
46     @GET("repeatedevent")
47     Call<ArrayList<RepeatedEvent>> getRepeatedEvents();
48
49     @GET("task")
50     Call<ArrayList<Task>> getTasks();
51
52     // POST
53
54     @POST("customerreaction")
55     Call<CustomerReaction> createCustomerReaction(@Body CustomerReaction customerReaction);
56
57     @POST("customer/login")
58     Call<Customer> customerLogin(@Body CustomerCredentials customerCredentials);
59
60 }

```

Kuva 17. Retrofitin rajapinta

Kuvassa 17 on rajapinta sovellukseen toteutetuista komennoista, joilla voidaan hakea tai lähettää dataa. Natiivisovelluksen keskustelu taustajärjestelmän kanssa on hyvin hakupainotteista, eli sovellus hakee paljon enemmän dataa kuin lähettää takaisin. Natiivisovelluksen taustajärjestelmästä saama vastaus tallennetaan puhelimen muistiin sen käsittelyn ajaksi. Käsittelyn jälkeen vastauksen sisältämä tieto säilyy puhelimen muistissa, jolloin sitä ei tarvitse kokoajan olla hakemassa uudestaan.

Sovellus lähettää taustajärjestelmälle oikeastaan vain kirjautumiseen käytetyt tiedot sekä käyttäjän antamat reaktiot ilmoituksiin. Näiden lähettämiseen käytetään "POST"-metodeja *customerLogin* sekä *createCustomerReaction*, jotka voi havaita kuvasta 17. Kutsujen sisältämä tieto lähetetään kutsun body-osassa.

```

58
59 // login customer
60 service.customerLogin(credentials).enqueue(new Callback<Customer>() {
61     @Override
62     public void onResponse(Call<Customer> call, Response<Customer> response) {
63
64         if (response.isSuccessful()) {
65             Intent mainActivity = new Intent(getApplicationContext(), MainActivity.class);
66             if (response.body() != null) {
67
68                 MainActivity.putExtra(name: "logged_in_customer", (Customer) response.body());
69
70                 // set customerId to sharedPrefs
71                 SharedPreferences sharedPreferences = getSharedPreferences(name: "function_app_sharedPrefs", Context.MODE_PRIVATE);
72                 SharedPreferences.Editor spEditor = sharedPreferences.edit();
73                 spEditor.putInt("customerId", response.body().getId());
74                 spEditor.apply();
75
76                 startActivity(mainActivity);
77             }
78         } else {
79             Log.i(tag "retrofit", response.toString());
80         }
81     }
82
83     @Override
84     public void onFailure(Call<Customer> call, Throwable t) {
85         Log.e(tag "retrofit", msg "" + call.toString() + t.getMessage());
86     }
87 });
88
89

```

Kuva 18. Sisäänkirjautumisessa käytetty funktio

Kuvassa 18 näkyvää funktiota käytetään kirjautuessa sovellukseen. Tämä funktio lukee käyttäjän syöttämän käyttäjätunnuksen sekä salasanan kirjautumissivulla olevista kentistä, jonka jälkeen funktio lähettää ne pyynnön mukana taustajärjestelmään. Taustajärjestelmän takaisin lähettämä vastaus otetaan sen jälkeen kiinni callback-funktiossa, jossa luetaan vastauksen sisältö. Jos vastauksen vastauskoodi on 200 eli onnistunut ja vastauksessa on sisältö, voidaan siirtyä sovelluksen päänäkömään. Samalla tallennetaan puhelimen muistiin käyttäjän ID myöhempiä käyttöä varten. Tiedot tallennetaan "PRIVATE"-moodissa, jolloin puhelimen muut sovellukset eivät pääse lukemaan tätä tietoa (Save key-value data 2020).

Jos vastauksen koodi oli jotain muuta kuin 200, oli kutsu epäonnistunut eikä käyttäjää voida päästää sisään sovelluksen päänäkömään. Epäonnistuneen kirjautumisyhteyden jälkeen sovellus ilmoittaa kirjautumisen epäonnistuneen kirjautumissivulla punaisella tekstillä.

3.11.5 Notifikaatiot

Notifikaatiot toteutettiin Androidin omalla NotificationManagerilla. Sovellus luo lo-kaaleja notifikaatioita sekä huolehtii itse niiden esittämisestä. Halutut muistutukset on syötetty järjestelmän nettisovelluksen kautta, jonka jälkeen ne voidaan hakea natiivisovellukseen taustajärjestelmän avulla. Sovellus luo notifikaatioita

taustajärjestelmältä saamansa tiedon mukaan. Muistutukset luodaan kirjautumisen jälkeen ja sovellus päivittää itsensä sekä muistutukset tilanteen tasalle tietyin väliajoin tai käyttäjän päivittäessä ne manuaalisesti.

Katsotaan seuraavaksi läpi, miten natiivisovelluksessa näytetään simppeli notifi-
kaatio, jossa on ikoni, teksti sekä sisältöä. Kirjastojen taikka riippuvuuksien käyt-
töönottoa ei tässä erikseen kuvata.

```
26
27 public class NotificationHelper extends ContextWrapper {
28
29     private static String TAG = "NotificationHelper";
30
31     NotificationManager notificationManager;
32     SharedPreferences sharedPreferences = getSharedPreferences("function_app", MODE_PRIVATE);
33
34     public NotificationHelper(Context base) {
35         super(base);
36         createNotificationChannel();
37     }
38
39     private void createNotificationChannel() {
40
41         // Create the NotificationChannel, but only on API 26+ because the NotificationChannel class is new and not in the support library
42         if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
43             int importance = NotificationManager.IMPORTANCE_HIGH;
44             NotificationChannel channel = new NotificationChannel("function_app_notification_channel", "Function App", importance);
45             channel.setDescription("function_app_notification_channel");
46
47             // Register the channel with the system; you can't change the importance or other notification behaviors after this
48             notificationManager = getManager();
49             notificationManager.createNotificationChannel(channel);
50         }
51     }
52
53     // get notificationManager to send notifications, if notification manager is not constructed, => construct it
54     private NotificationManager getManager() {
55         if (notificationManager == null) {
56             notificationManager = (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
57         }
58         return notificationManager;
59     }
60 }
```

Kuva 19. NotificationManagerin konfigurointi sekä instanssin luominen

Ennen kuin notifiikaatioita pystyy lähettämään, on rekisteröitävä kanava, jolle lähettää notifiikaatioita. Kanavan pystyy perustamaan *createNotificationChannel*-metodilla (kuva 19). Kanava täytyy luoda ennen kuin sovellus yrittää lähettää notifiikaatioita. Kanavaa muodostettaessa sen rakentajalle täytyy välittää parametrimä *importance*, eli kanavan tärkeys. Vaikka notifiikaatioiden tärkeys on asetettu, ei se kuitenkaan takaa sitä, että ne aina ilmestyisivät sen mukaan. Käyttäjä voi kuitenkin aina muuttaa sovelluksen notifiikaatioiden tärkeyttä, joten sovelluksen tarvitsemat ”tunkeilevat” notifiikaatiot täytyi toteuttaa muilla keinoilla.

```

60
61 // builds the notification from constructed notification body
62 public void notify(int id, NotificationCompat.Builder notification) {
63     getManager().notify(id, notification.build());
64 }
65
66 // dismiss notification
67 public void cancel(int id) {
68     getManager().cancel(id);
69 }
70

```

Kuva 20. Notifikaation esittämiseen sekä notifikaation poistamiseen käytetyt funktiot

Notify-metodilla voidaan näyttää käyttäjälle notifikaatio, joka voidaan puolestaan poistaa *cancel*-metodilla. *Notify*-metodi rakentaa sille parametrina annetun notifikaation ja ohjelma tunnistaa notifikaation sille asetetulla tunnisteella. (Kuva 20)

```

70
71 // construct notification
72 public NotificationCompat.Builder buildNotification(String title, String body) {
73     return new NotificationCompat.Builder(this, "function_app_notification_channel")
74         .setSmallIcon(R.drawable.to_predict)
75         .setContentTitle(title)
76         .setContentText(body);
77 }
78

```

Kuva 21. Yksinkertaisen notifikaation rakentaminen

Kuvassa 21 on kuvattu yksinkertaisen notifikaation rakentamiseen tarvittavat ohjeet, jotka annetaan *NotificationCompat*-luokan *Builder*-oliolle. *Builder*-olio haluaa parametreina kontekstin sekä notifikaatioita esittävän kanavan tunnisteeseen. Notifikaatiolle annetaan ikoni, otsikko ja haluttu tekstisisältö. Kuvassa 21 olevasta rakentajasta on hyvä huomata, että sille annetaan notifikaation sisältö parametreina, jolloin kyseisellä metodilla voidaan tehdä sisällöltään erilaisia notifikaatiota.

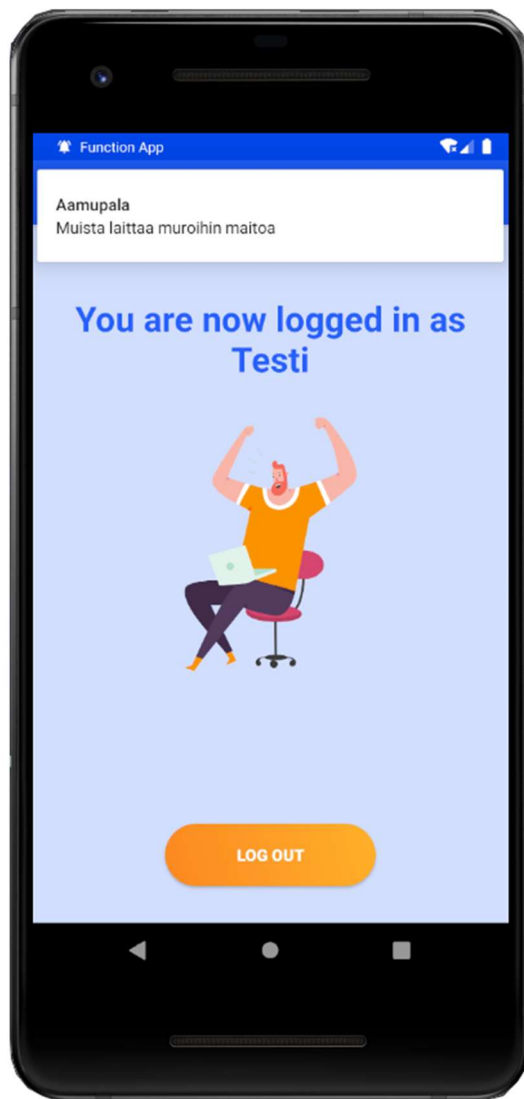
```

79 // construct notification with actions
80 public NotificationCompat.Builder buildNotificationWithActions(String title, String body, int taskId, String imageUrl, String avUrl) {
81
82     Intent startNotificationActivity = new Intent(this, NotificationActivity.class);
83
84     // open notification activity
85     Intent openNotificationActivity = new Intent(this, NotificationActivity.class);
86     openNotificationActivity.putExtra("task title", title);
87     openNotificationActivity.putExtra("task message", body);
88     openNotificationActivity.putExtra("task id", taskId);
89     openNotificationActivity.putExtra("image url", imageUrl);
90     openNotificationActivity.putExtra("av url", avUrl);
91     PendingIntent pendingNotificationActivity = PendingIntent.getActivity(this, taskId, openNotificationActivity, PendingIntent.FLAG_UPDATE_CURRENT);
92
93     // get ringtone
94     Uri alarmSound = RingtoneManager.getDefaultUri(RingtoneManager.TYPE_NOTIFICATION);
95
96     // play notification sound
97     try {
98         Ringtone r = RingtoneManager.getRingtone(getApplicationContext(), alarmSound);
99         r.play();
100     } catch (Exception e) {
101         e.printStackTrace();
102     }
103
104     // create notification with buttons (only api 26 or higher)
105     return new NotificationCompat.Builder(this, "function_app_notification_channel")
106         .setFullScreenIntent(PendingIntent.getActivity(this, taskId, startNotificationActivity, PendingIntent.FLAG_ONE_SHOT), true)
107         .setAutoCancel(true)
108         .setSmallIcon(R.drawable.test_notification_icon_bell)
109         .setContentTitle(title)
110         .setContentText(body)
111         .setContentIntent(pendingNotificationActivity);
112 }
113
114

```

Kuva 22. NotificationActivityyn käynnistävä notifikaatio

Kuvassa 22 on kuvattu sovelluksen yleisesti käyttämän notifikaation rakentaja. Tämä notifikaatio avaa painettaessa natiivisovelluksen *NotificationActivity*-näytteen, jossa käyttäjä voi reagoita tehtäviin. Käyttäjälle ilmestyvän notifikaation voi havaita kuvasta 23.



Kuva 23. Sovelluksen lähettämä notifikaatio

Nämä notifikaatioita käsittelevät kappaleet viittasivat kokonaisuudessaan Google Developersin Android-dokumentaatioon ja projektin lähdekoodiin, jota on näkyvillä kuvissa 19 – 22.

3.11.6 Testaus, luvat ja asennus

Testaajien oli tarkoitus asentaa sovellus Google Play –sovelluskaupan kautta, mutta julkaisu viivästyi eikä tämä ollut testijaksolla mahdollista. Päädyimme jakamaan asennuspaketin testaajille Microsoft OneDrive –jakopalvelun kautta. Toimiakseen sovellus vaatii käyttäjältä luvan internet-yhteyteen ja muihin suojattuihin järjestelmäominaisuuksiin, jotka sovellus pyytää ensimmäisellä käynnistyskerralla.

3.12 Angular web-käyttöliittymä

Toteutimme sovellukselle yksinkertaisen web-käyttöliittymän Angular 7 -sovelluskehystä käyttäen. Web-käyttöliittymän avulla tukihenkilöt voivat lisätä asiakkailleen tehtävälistoja, joiden tehtävistä asiakkaat saavat ilmoituksia natiivisovellukseensa.

The screenshot shows a web application interface for adding tasks. At the top, there is a dropdown menu labeled 'Valitse kalenteri' (Select calendar) with 'Kaisan kalenteri' (Kaisa's calendar) selected. Below this is a section titled 'Lisää eventti' (Add event). It contains three input fields: 'Eventin nimi' (Event name) with the value 'Iltatoimet', 'Eventin ID' (Event ID) with the value '30', and 'Alkuajankohta' (Start time) with the value '15.04.2020 22.22'. Below these fields is a section titled 'Tehtävät' (Tasks). It contains three rows of task entries. Each row has three columns: 'Tehtävän nimi' (Task name), 'Aputeksti' (Description), and 'Tehtävän kesto minuutteina' (Task duration in minutes). The first row has 'Iltapala' (Dinner), 'Syö leipää' (Eat bread), and '10'. The second row has 'Pese hampaat' (Brush teeth), 'Käytä sähköhammasharjaa' (Use electric toothbrush), and '5'. The third row has 'Käy nukkumaan' (Go to bed), 'öitä' (hours), and '1'. To the right of each row, there are three radio buttons for priority: 'Korkea' (High), 'Keskitaso' (Medium), and 'Matala' (Low). The first row has 'Matala' selected, the second row has 'Korkea' selected, and the third row has 'Korkea' selected. At the bottom of the task list, there are two buttons: 'Lisää rivi' (Add row) and 'Tallenna' (Save).

| Tehtävän nimi | Aputeksti | Tehtävän kesto minuutteina | Korkea | Keskitaso | Matala |
|---------------|-------------------------|----------------------------|----------------------------------|-----------------------|----------------------------------|
| Iltapala | Syö leipää | 10 | <input type="radio"/> | <input type="radio"/> | <input checked="" type="radio"/> |
| Pese hampaat | Käytä sähköhammasharjaa | 5 | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> |
| Käy nukkumaan | öitä | 1 | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> |

Kuva 24. Nettisovelluksen käyttöliittymä

Kuvassa 24 näkyvässä käyttöliittymässä avustaja valitsee ensimmäisenä, kenen asiakkaan kalenteriin hän haluaa tehtävälistan luoda. Tämän jälkeen tehtävälis- talle annetaan nimi, kuten esimerkiksi aamurutiinit ja alkuajankohta, jolloin asia- kas saa tehtävälistan ensimmäisestä tehtävästä ilmoituksen.

Tehtävälista koostuu tehtävistä, kuten esimerkiksi aamurutiinien tapauksessa he- rätyksestä, hampaiden pesusta, aamupalan syömisestä, pukemisesta ja kouluun lähtemisestä. Tehtävälis- talle lisätään jokaiselle tehtävälle oma rivi ja riville täyte- tään tehtävän nimi, apu- tai ohjeteksti, tehtävälle varattu aika minuutteina, tehtä- vän prioriteettitaso, ja kuvat sekä itse ilmoitukselle että lisäohjeelle. Kaikista tär- keimmille tehtäville käytetään korkeaa prioriteettia, jolloin ilmoitus on tehok- kaampi ja järjestelmä lähettää reagoimattomille ilmoituksille varailmoituksia. Ma- talan prioriteetin tehtävät eivät ole tehtävälistan tai päivän kulun kannalta kriittisiä.

Kun tehtävälista on valmis avustaja tallentaa sen relaatiotietokantaan käyttöliitty- män Tallenna-painiketta painamalla.

Web-käyttöliittymän toteutukseen käytettiin Angular 7 -ohjelmistokehystä. Angular on TypeScript-pohjainen avoimen lähdekoodin ohjelmistokehys, jota käytetään modernien web-sovelluksien kehittämiseen HTML ja TypeScript -ohjelmointikielien avulla. Ulkoasussa on käytetty Googlen Angular Material UI-komponenttikirjastoa, joka on kokoelma laadukkaita, testattuja ja moderneja suunnitteluperiaatteita noudattavia tyylimääriä.


3.13 Julkaisu

Järjestelmän eri osat on julkaistava, jotta niitä voitaisiin käyttää. Natiivisovellus on ladattava laitteeseen, joten se julkaistaan sovelluskauppaan. Testijakson aikana sovelluksen sai kuitenkin ladattua erillisen linkin kautta kehittäjien käytössä olleesta pilvitallennustilasta.

Järjestelmän muut osa-alueet luotiin Microsoftin Azure-pilvipalveluun. Tietokannalle luotiin pilvipalvelussa oma SQL Server-instanssi, jonne tietokanta saatiin pystytettyä. Taustajärjestelmä sekä Angular-nettisovellus julkaistiin App Service-instansseina.

4 Julkaisu sovelluskauppaan

Natiivisovellus oli tarkoitus jakaa testiryhmän käyttöön Google Play-sovelluskaupan kautta. Julkaisu saatiin tehtyä, mutta sovellus hylättiin tuntemattomasta syystä, eikä asiaa ehditty korjaamaan tiukan aikataulun vuoksi. Sovellus päädyttiin jakamaan testiryhmälle suoraan .apk-asennuspakettina. Tässä luvussa käsitellään kuitenkin sovelluskaupan vaatimia pakollisia tietoja julkaisua varten.

**Versio: Function App 1.5** [Muokkaa](#)

15. toukokuuta 0.24: Täysi käyttöönotto

Käyttöönottohistoria
15. toukokuuta 0.24: Täysi käyttöönotto

Mitä uutta tässä julkaisussa on?
Oletus – Suomi – fi-FI
Muutoksia ilmoitusten latautumiseen, portrait-mode only

[Käännetty yhdelle kielelle](#) [Muokkaa julkaisutietoja](#)

Android-sovelluspaketit ja APK:t
[Tämä julkaisu täyttää Google Playn 64-bittisyyden vaatimuksen. Lue lisää](#)

Laajenna kaikki

| Tyyppi | Versiokoodi | Ominaisuudet | Lähetetyt | Sovelluksen latauskoko | Asennukset aktiivisille laitteille |
|--------------------------|-------------|--------------|---------------------|------------------------|------------------------------------|
| 1 APK lisätty | | | | | |
| APK | 5 | 2 | 15. toukokuuta 0.21 | 2,40 Mt | Ei tietoja |
| 1 APK poistettu käytöstä | | | | | |

Alueellinen saatavuus: Beta

Maat, joissa ei saatavilla
150
ja muu maailma

Maat, joissa saatavilla
1
synkronoitu tuotannon kanssa

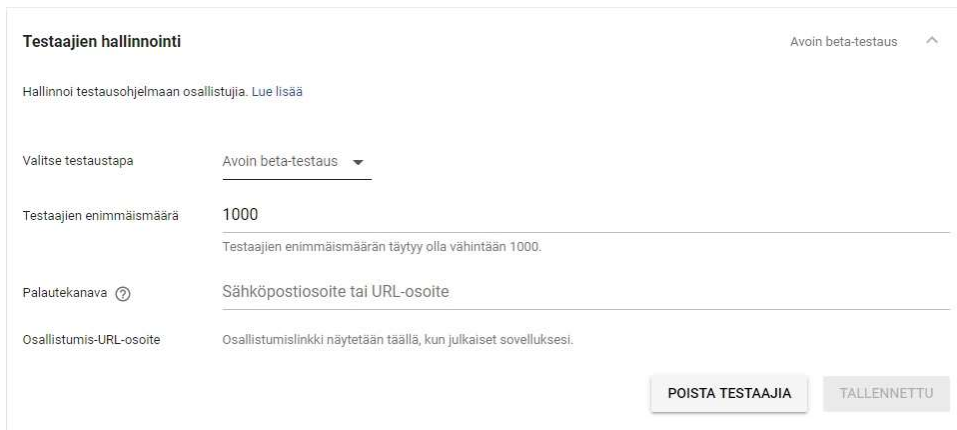
Hallinnoi alueellista saatavuutta

Kuva 25. Julkaisu avoimelle kanavalle

Google Play –sovelluskaupassa on sovelluksille neljä erilaista julkaisukanavaa. Julkaisukanava vaikuttaa sovelluksen näkyvyyteen sovelluskaupassa, ja se valitaan käyttötarkoituksen mukaan. Tuotantokanava on tarkoitettu valmiille sovellusversioille, ja tällä kanavalla tehty julkaisu näkyy kaikille sovelluskaupan käyttäjille. Avoin, suljettu, ja sisäinen julkaisukanava ovat tarkoitettu sovelluksen testaamiseen. Sisäinen kanava on pienimmälle testiryhmälle tarkoitettu, ja tällä kanavalla sovellusta pääsee yleensä testaamaan vain sovelluksen kehittäjät. Sisäistä kanavaa käytetään, kun testiryhmää halutaan laajentaa hieman, ja mu-

kaan voidaan ottaa esimerkiksi muita työntekijöitä tai luotettuja käyttäjiä. Avoimelle kanavalle tehtyä sovellusjulkaisua voi käyttää kuka tahansa kenellä on osoite sovelluksen julkaisusivulle, tai on saanut kutsun sähköpostissa. (Kuva 25)

Valitsimme julkaisukanavaksi suoraan avoimen testikanavan, ja tarkoituksena oli jakaa sovellusjulkaisun osoite testiryhmälle.



Kuva 26. Testaajien hallinnointi

Avoimella julkaisukanavalla testaajien määrää voidaan rajata asettamalla sovellukselle alueellinen saatavuus, ja testaajille enimmäismäärä. Rajasimme sovelluksen näkymään vain suomen sovelluskaupassa asettamalla alueelliseksi saatavuudeksi suomen, ja valitsimme testaajien enimmäismääräksi 1000, joka oli minimimäärä. Testaamiseen osallistuminen suunniteltiin tapahtuvaksi jaetun osallistumislinkin kautta. (Kuva 26)

Tuotetiedot

SUOMI - FI-FI Hallinnoi käännöksiä

Täytä ennen julkaisua kentät, joiden kohdalle on merkitty *.

Nimi *
Suomi - fi-FI

Function App

12/50

Lyhyt kuvaus *
Suomi - fi-FI

Function App aikatauluttaa ja ohjaa toimintaa muistutusten ja hälytysten kautta

79/80

Koko kuvaus *
Suomi - fi-FI

Järjestelmän käyttötarkoitus on helpottaa arjen hallintaa. Järjestelmä muistuttaa käyttäjää äänen, kuvan ja tekstin avulla ennalta määrättyistä tehtävistä.

153/4000

Vältä yleisimmät sovelluksen sisällönkuvausrikkomukset tutustumalla sisällönkuvauskäytäntöön. Muista myös tutustua kaikkiin muihin ohjelmasaäntöihin ennen sovelluksen lähettämistä.

Jos sovelluksesi tai tietosivusi on oikeutettu ennakkoilmoituksen lähettämiseen Google Playn sovellusten tarkistustimille, ota meihin yhteyttä ennen julkaisemista.

Kuva 27. Sovelluksesta vaadittavat tiedot sovelluskauppaan julkaistaessa

Ennen julkaisua sovellukselle täytyi antaa pakollisia tuotetietoja, joita olivat muun muassa nimi, lyhyt, maksimissaan 80 merkkiä pitkä kuvaus sovelluksen toiminnasta tai käyttötarkoituksesta, ja pidempi, maksimissaan 4000 merkkiä pitkä kuvaus sovelluksesta (kuva 27).

Koska sovellus oli tarkoitus jakaa testiryhmälle, joka sai erillisen opastuksen sovelluksen käytöstä, oli yksinkertainen ja lyhyt kuvaus tässä vaiheessa riittävä.

Graafinen sisältö

Ylläpidä sovelluskuvaketta, kuvakaappauksia ja videoita mainostaaksesi sovellustasi Google Playssa. Lue lisää

Tutustu toisena esiintymistä ja immateriaalimaisuutta koskevaan käyttöön ennen uusien kuvien lähettämistä. Jos lisäät käännöksiä tietosivullesi ilman lokalisoituja kuvia, oletustietosivun kuvia käytetään.

Korkean resoluution kuvake *

Oletus – Suomi – fi-FI

512 x 512

32-bittinen PNG

Google Playn kuvakemuoto

Google Play lisää pyöristetyt reunat ja varjostuksen kuvakkeeseen dynaamisesti. Läpinäkyviä taustakuvia ei enää sallita. Lue lisää



Miltä kuvakkeesi näyttää Google Playssa

Kuva 28. Sovelluksen kuvake

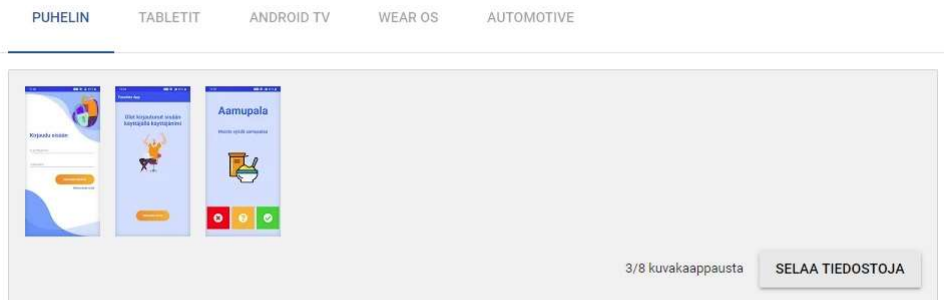
Sovellukselle piti antaa erilaista graafista sisältöä, kuten esimerkiksi sovelluskuvake. Virallista sovelluskuvaketta ei vielä tässä vaiheessa ollut, joten teimme itse väliaikaisen kelloa muistuttavan kuvakkeen (kuva 28). Kuvakkeella oli tiettyjä vaatimuksia, kuten minimiresoluutio ja hyväksytyt tiedostomuodot.

Kuvakaappaukset *

Oletus – Suomi – fi-FI

JPEG tai 24-bittinen PNG (ei-alfa). Sivun vähimmäispituus: 320 kuvapistettä. Sivun enimmäispituus: 3 840 kuvapistettä.

Lisää yhteensä vähintään kaksi kuvakaappausta. Tyyppiä kohden voi lisätä korkeintaan kahdeksan kuvakaappausta. Vaihda järjestystä tai siirrä kuvia tyyppien välillä vetämällä.



Ominaisuuskuva *

Oletus – Suomi – fi-FI

1024 lev. x 500 kork.

JPG tai 24-bittinen PNG (ei alfa)

Mainoskuva

Oletus – Suomi – fi-FI

180 lev. x 120 kork.

JPG tai 24-bittinen PNG (ei alfa)

TV-banneri

Oletus – Suomi – fi-FI

1 280 lev. x 720 kork.

JPG tai 24-bittinen PNG (ei alfa)



Lisää mainoskuva
Pudota kuva tähän



Lisää TV-banneri
Pudota kuva tähän

Kuva 29. Sovelluksesta lisättävä graafinen materiaali

Sovelluksen sisällöstä vaadittiin vähintään 2 kuvakaappausta. Lisäsimme puhelinkäyttöliittymästä kuvakaappaukset jokaisesta sovelluksen näkymästä, eli kirjautumisruudusta, sovelluksen etusivusta, ja ilmoitusruudusta (kuva 29).

Jos sovellus hyödyntäisi muita Android-käyttöympäristöjä, kuten esimerkiksi tabletteja, Android TV:tä, tai älykelloja, voitaisiin niille lisätä erilliset kuvakaappaukset. Tällä hetkellä näin ei kuitenkaan ollut, joten lisäsimme kuvakaappaukset vain puhelinkäyttöliittymästä. Kuvakaappaukset näkyvät sovelluksen julkaisusivulla.

Luokittelu

Sovelluksen tyyppi *
Sovellukset

Luokka *
Elämäntapa

Tagit

Lisää tageja, jotka kuvaavat sovelluksen sisältöä ja toimintoja. Tagit voivat vaikuttaa siihen, missä sovelluksesi näkyy Google Playssa, ja vertailuryhmiin, joihin sinua verrataan. Lue lisää

MUOKKAA TAGEJA

Kuva 30. Sovelluksen luokittelu

Sovelluskaupassa on kahden tyyppisiä sovelluksia, pelejä ja sovelluksia. Annoimme sovellukselle tyyppiksi "Sovellukset", ja sovellusluokaksi valikoitui "Elämäntapa". Sovellukselle on myös mahdollista antaa erilaisia tageja, jotka helpottavat sovelluksen löytymistä sovelluskaupasta, mutta emme lisänneet niitä vielä tässä vaiheessa. (Kuva 30)

Sisällön ikärajoitus *

NYKYINEN IKÄRAJOITUS
Lähetetty: 14. toukokuuta 21.10
Näytä tiedot Lue lisää

L E 3 0 3+ 3

Kuva 31. Sovelluksen ikärajoitus

Vastasimme Google Play –sovelluskaupan ikärajoitus-kyselyyn, ja sovelluksen suomalaisiksi ikärajaksi muodostui kolme vuotta (kuva 31). Eri mailla on hieman

eri käytännöt, ja esimerkiksi Saksassa ja Pohjois-Amerikassa sovellus on sallittu kaikenikäisille, kun taas Euroopassa, Venäjällä, Etelä-Koreassa ja muualla maailmassa käyttäjän tulee olla vähintään kolme vuotias. Julkaisu rajattiin alueellisesti vain Suomen sovelluskauppaan, joten muiden maiden ikärajoilla ei tässä tapauksessa ollut merkitystä.

Yhteystiedot

Sivusto

Sähköposti *

Anna sähköpostiosoite, jonka kautta sinuun voi ottaa yhteyttä. Osoite näytetään julkisesti sovelluksesi yhteydessä.

Puhelin

Kuva 32. Tekijöitä vaadittavat yhteystiedot

Tekijöiden yhteystiedoista ainut pakollinen kenttä oli sähköposti, joten jätimme kotisivun ja puhelinnumeron täyttämättä. (Kuva 32)

Tietosuojakäytäntö

✓ Lähetetty - Tietosuojakäytäntö lisätty

Kun lisäät sovelluksesi tietosivulle tietosuojakäytännön, käyttäjäsi saavat paremman kuvan siitä, miten käsittelet arkaluontoisia käyttäjä- ja laitetietoja.

Piilota yhteenveto ☐ [Muuta](#)

Tietosuojakäytäntö <https://functionappprivacypolicy.imfast.io/>

Kuva 33. Tietosuojakäytäntö

Sovellukselle täytyi toimittaa tietosuojakäytäntö erillisenä verkkosivuna. Tietosuojakäytännöstä käyttäjät saavat paremman kuvan siitä, miten arkaluontoista

käyttäjä- ja laitetietoa käsitellään, ja minkälaista tietoa käyttäjistä kerätään. (Kuva 33)

Mainokset

✓ Lähetetty • Sovellus ei sisällä mainoksia

Sinun on ilmoitettava meille, sisältääkö sovelluksesi mainoksia. Sisältää mainoksia -tunniste näkyy mainoksia sisältävien sovellusten kohdalla Google Playssa. Varmista, että tiedot ovat oikein ja ajan tasalla.

Piilota yhteenveto ^ Muuta

Mainokset Olet ilmoittanut, että sovelluksesi ei sisällä mainoksia.

Sovellusten käyttöoikeudet

✓ Lähetetty • Kaikki toiminnot käytettävissä ilman erityisiä pääsyoikeuksia

Jos sovelluksesi osia on rajoitettu kirjautumistietojen, jäsenyyden, sijainnin tai muun todennustavan perusteella, kerro, miten niihin pääsee.

Piilota yhteenveto ^ Muuta

Ohjeet Kaikki toiminnot ovat käytettävissä ilman erityisiä käyttöoikeuksia

Kohdeyleisö ja sisältö

✓ Lähetetty

Sinun on ilmoitettava meille sovelluksesi kohdeikäryhmä ja muut tiedot sen sisällöstä. Näin voimme varmistaa, että lapsille suunnitellut sovellukset ovat heille turvallisia ja sopivia.

Piilota yhteenveto ^ Muuta

Kohdeikä Sovelluksesi kohdeikäryhmä on: 9–12, 13–15, 16–17, 18-vuotiaat ja vanhemmat

Designed for Families Sovelluksesi ei lähde mukaan Designed for Families -ohjelmaan

Kuva 34. Muut pakolliset tiedot

Muita pakollisia tietoja olivat sisältääkö sovellus mainoksia, vaativatko jotkin ominaisuudet tai toiminnallisuudet erillisiä käyttöoikeuksia, ja minkä ikäiselle yleisölle sovellus ja sen sisältö on suunnattu. Sovelluksemme ei sisältänyt mainoksia tai erillisiä käyttöoikeuksia vaativia toiminnallisuuksia, ja sovellus on suunnattu kaikenikäisille noin yhdeksänvuotiaasta ylöspäin. (Kuva 34)

5 Testaus

Sovelluksen testaus rajoittui kehittäessä tapahtuneeseen käsin toteutettuun testaamiseen sekä myöhemmässä vaiheessa asiakkailla tapahtuneeseen testaukseen. Sovelluksessa ei havaittu kriittisiä ongelmia ja se voitiin luovuttaa asiakkailla testattavaksi.

5.1 Asiakkaat

Sovellusta testattiin asiakkailla kehitystyön loppuvaiheessa. Testaajat saivat linkin .apk-tiedostoon, josta sovelluksen pystyi asentamaan puhelimeen. Testaaja sovelluksen asennettuaan pystyi kirjautumaan sovellukseen hänelle ennalta luoduilla tunnuksilla, joiden takaa löytyi hänen viikko-ohjelmansa. Testaajat eivät itse päässeet syöttämään omaa viikko-ohjelmaansa, vaan se oli kysytty heiltä etukäteen. Tämän viikko-ohjelma oli sitten valmiiksi laadittu testitunnukselle, joka luovutettiin testaajalle testijakson alkaessa.

Sovelluksen testaus tapahtui sen jälkeen vaatimusmäärittelyn mukaisesti sekä testaajilta kerättiin palaute testijakson lopulla. Sovellukseen ei testijakson aikana tehty päivityksiä, jotka olisivat saattaneet vaikuttaa palautteeseen myönteisesti tai kielteisesti.

5.2 Käyttäjäkokemukset ja jatkokehitys

Tässä kappaleessa kuvataan sovelluksen testausvaiheessa saatuja käyttäjäkokemuksia sekä sovelluksen mahdollisia jatkokehityssuunnitelmia.

5.2.1 Käyttäjäkokemukset

Saimme järjestelmää kesäkuussa testanneelta testiryhmältä vaihtelevaa palautetta. Suurin osa piti ideaa hyvänä ja hyödyllisenä, mutta selviä puutteitakin löytyi. Testikäyttäjät pitivät muun muassa sovelluksen helppokäyttöisyydestä, ja asennus onnistui kaikilta myös vaivatta. Joillain puhelinmalleilla oli kuitenkin ilmoituksien kanssa hieman ongelmia, eivätkä ilmoitukset olleet käyttäjien mielestä tarpeeksi huomiota herättäviä.

Testaus aloitettiin samoihin aikoihin kuin koronapandemia puhkesi ensimmäistä kertaa, joka aiheutti myös omat hankaluutensa rajallisten lähitapaamisten vuoksi. Vaikka testiryhmältä saatiin jo nyt muutamia hyviä jatkokehitysideoita, tarvitsee järjestelmä ehdottomasti lisää testausta.

5.2.2 Jatkokehitys

Ylivoimaisesti selkein jatkokehityskohde natiivisovelluksessa koskee sovelluksen käyttäjälle lähettämiä ilmoituksia. Ilmoitukset täytyisi saada jollain tapaa kaappaamaan puhelin niin, ettei niitä voi olla huomaamatta tai ohittaa, samaan tapaan kuin puhelimesta löytyvä herätyskello. Tällä hetkellä sovellus lähettää ilmoitukset yksinkertaisina push-notifikaatioina, joihin käyttäjän ei tarvitse välttämättä reagoida jos hän ei niin halua. Tämä mahdollistaa esimerkiksi sen, että käyttäjältä saattaa jäädä ilmoitus huomaamatta, tai esimerkiksi mobiilipeliin uppoutunut käyttäjä saattaa yksinkertaisesti jättää ilmoituksen huomioimatta.

Web-sovelluksessa on myös paljon mahdollisuuksia jatkokehitykselle. Terapeuttien käyttökokemusta parantaisi esimerkiksi huomattavasti se, jos tehtävälistoista voisi tallentaa valmiita pohjia, ettei saman tyyppisiä listoja, kuten esimerkiksi aamurutiineja tarvitsisi kirjoittaa aina alusta alkaen. Toinen hyvä ominaisuus olisi interaktiivisen kalenterin integroiminen, jonka avulla käyttäjä voisi asettaa tehtävälistojen alku- ja loppuajat helpommin.

Koska jokainen loppukäyttäjän sovelluksessa tekemä toiminto kirjataan tietokantaan, olisi natiivisovelluksen käyttäjistä helppo luoda hyödyllistä analytiikkaa. Esimerkiksi web-sovellukseen voisi tehdä erillisen näkymän, josta terapeutti voisi seurata asiakkaidensa käyttäytymistä sovelluksessa, ja mahdollista kehitystä rutiinien hallitsemisessa. Työkaluna analytiikan luomiseen voisi käyttää esimerkiksi Microsoftin Power BI –raportointi- ja analysointipalvelua.

6 Pohdinta

Projekti onnistui kaiken kaikkiaan melko hyvin, vaikka koronapandemia ja lupaasiat vaikeuttivatkin testauksen aloittamista. Sovellus saatiin testikuntoon joitain puhelinmalleja lukuun ottamatta, google play-sovelluskauppaan julkaiseminen jäi kesken, testaus oli suht farssi.

Järjestelmän vaatimusmäärittely oli vaikeaa järjestelmän laajuuden vuoksi sekä minimiversion eli MVP-järjestelmän rajausta oli hankalaa. Vaatimusmäärittelylle sallittu aika oli hyvin rajallinen tilaajan kiireen sekä kehittäjien ajanpuutteen takia.

Vaatusmääritys saatiin kuitenkin tehtyä sekä sen tarkkuus oli varsin mainio. Myös järjestelmän minimiversion rajausta hankaluudesta huolimatta onnistui.

Sovelluksen kehittäminen sai takapakkia siinä vaiheessa, kun selvisi että sovelluksen tarvitsema `SYSTEM_ALERT_WINDOW`-oikeus on tarkoitus poistaa tulevissa Android-versioissa, joten sovellus olisi ollut vanhentunut jo valmistuessaan. Tämän uhan vuoksi sovelluksen ilmoituksen päätettiin toteuttaa ilmoitusten sekä hälytysten avulla.

Tämän projektin ehdottomasti helpoin osuus oli loppujen lopuksi itse sovelluskehitys, vaikka osa käytetyistä teknologioista eivät olleetkaan kehittäjien vahvuuksia. Taustajärjestelmän kehitys onnistui hujauksessa heti kun järjestelmän tietomalli oli saatu laadittua. Tietomallin laatimiseen käytettiin hyvin runsaasti aikaa verrattuna saatavilla olleeseen aikatauluun sekä kiireeseen. Tietomallin ohella ensimmäisenä laadittiin tietokanta tälle järjestelmälle, jota vasten kehitystyö aloitettiin ”database-first” ajattelumallilla.

Tietomallien laatimisen jälkeen taustajärjestelmä voitiin kehittää melkein loppuun asti. Taustajärjestelmän ollessa valmis jakamaan tietoa eteenpäin pystyttiin aloittamaan käyttöliittymien kehitys kummankin käyttöliittymän osalta. Natiivikäyttöliittymän kehitys oli selvästi aikaa vievämpää Angularilla toteutettuun nettisovellukseen verrattuna.

Natiivikäyttöliittymän haasteista huolimatta se saatiin toteutettua varsin hyvin niin käytettävyydeltään kuin ulkoasultaan. Käyttöliittymässä ei havaittu testauksen aikana suurempia ongelmia tai muita haittakohtia. Nettisovelluksen kehitys onnistui ongelmitta sekä sen käytettävyys saatiin hyvälle mallille. Toiminnallisuus nettisovelluksessa on kunnossa eikä senkään käytettävyydessä olla havaittu ongelmia.

Kommunikointi tilaajan kanssa sujui hyvin sekä ongelmitta. Tilaajan kanssa pidettiin kehityksen aikana palavereja noin joka toinen viikko. Palavereista saatiin hyvin selville tilaajan kanta järjestelmän silloisesta tilasta sekä mihin suuntaan sitä tulisi kehittää.

Lopputuloksena järjestelmä saatiin hyvään vaiheeseen varsinkin, kun ajatellaan järjestelmän vaatimusmäärittelyyn sekä kehitykseen käytettyä aikaa. Järjestelmä

oli saatava nopeasti testausvaiheeseen sekä kehittäjien ajanpuutetta ajatellen järjestelmä saatiin hyvin aikaiseksi. Minimivaatimuksien osalta järjestelmä saavutti tilaajan esittämät vaatimukset niin sanottua MVP eli ”Minimum Viable Product”-versiota ajatellen.

Lähteet

Modern API Design with ASP.NET Core 2, Fanie Reynders, 2018

Get started with Swashbuckle and ASP.NET Core, Microsoft, <https://docs.microsoft.com/en-us/aspnet/core/tutorials/getting-started-with-swashbuckle?view=aspnetcore-3.1&tabs=visual-studio> Luettu 4.10.2020.

What Is OpenAPI?, <https://swagger.io/docs/specification/about/>, Luettu 4.10.2020.

Android SharedPreferences, <https://developer.android.com/training/data-storage/shared-preferences>. Luettu 10.10.2020.

Android Create a Notification, Google Developers, <https://developer.android.com/training/notify-user/build-notification#notify>. Luettu 10.10.2020.

Web API Design, Microsoft, <https://docs.microsoft.com/en-us/azure/architecture/best-practices/api-design>. Luettu 11.10.2020.

Pixel Perfect Precision Handbook 3, ustwo, <https://downloads.ctfassets.net/gw5wr8vzz44g/2bMTFo4agkUgmsSgeu8uik/9d2d7e4ee7d655ca847a009329e20976/PP3.pdf>. Luettu 12.10.2020.

Mitä eroa on sisäisellä, suljetulla ja avoimella testillä, Google, <https://support.google.com/googleplay/android-developer/answer/3131213?hl=fi>. Luettu 12.10.2020.

Build configuration files, <https://developer.android.com/studio/build#build-files>. Luettu 13.10.2020.

Add build dependencies, <https://developer.android.com/studio/build/dependencies> Luettu 13.10.2020

What's new in .NET Core 3.0, Microsoft, <https://docs.microsoft.com/en-us/dotnet/core/whats-new/dotnet-core-3-0>, Luettu 14.10.2020

Mikä on henkilötieto, Tietosuoja, <https://tietosuoja.fi/gdpr> Luettu 16.10.2020

Android Studio, Google Developers, <https://developer.android.com/studio/features>, Luettu 14.10.2020

WebStorm – The Smartest Javascript IDE, Atlassian, <https://confluence.jetbrains.com/display/WI/WebStorm+IDE> Luettu 14.10.2020

What is SQL Server Management Studio (SSMS). Microsoft, <https://docs.microsoft.com/en-us/sql/ssms/sql-server-management-studio-ssms?view=sql-server-ver15> Luettu 14.10.2020

App Manifest Overview, Google Developers, <https://developer.android.com/guide/topics/manifest/manifest-intro> Luettu 16.10.2020

Dynamic Data Masking, Microsoft, <https://docs.microsoft.com/en-us/azure/azure-sql/database/dynamic-data-masking-overview> Luettu 17.10.2020

Save key-value data, Google Developers <https://developer.android.com/training/data-storage/shared-preferences> Luettu 17.10.2020

Characteristics of Modern Web Applications, Microsoft, <https://docs.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/modern-web-applications-characteristics> Luettu 17.10.2020

Retrofit <https://square.github.io/retrofit/> Luettu 17.10.2020

Visual Studio 2019, Microsoft, <https://visualstudio.microsoft.com/vs/> Luettu 18.10.2020

Android Studio, Google Developers, <https://developer.android.com/studio> Luettu 18.10.2020