SAVONIA

OPINNÄYTETYÖ - **AMMATTIKORKEAKOULUTUTKINTO**
TEKNIIKAN JA LIIKENTEEN ALA

ANDROID-SOVELLUS (KOTLIN) KUUKAUSIMAKSUJEN SEURANTAAN KÄYTTÄEN PSD2 OPEN BANKING RAJAPINTOJA

TEKIJÄ: Riku Wikman

SAVONIA-AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Tiivistelmä

Koulutusala Tekniikan ja liikenteen ala	
Koulutusohjelma/Tutkinto-ohjelma Tietotekniikan koulutusohjelma	
Työn tekijä(t) Riku Wikman	
Työn nimi Android-sovellus (Kotlin) kuukausimaksujen seurantaan käyttäen PSD2 Open Banking rajapintoja	
Päiväys 05.06.2018	Sivumäärä/Liitteet 38/0
Ohjaaja(t) Jussi Koistinen ja Mikko Pääkkönen	
Toimeksiantaja/Yhteistyökumppani(t) Mikko Pääkkönen	

Tiivistelmä

Opinnäytetyön tarkoituksena oli luoda demosovellus olemassa olevasta yritysideasta. Yritysidea itsessään syntyi Savonia-ammattikorkeakoulun tarjoamalla yPolku-kurssilla, jossa tarkoituksena on kehittää yritysidea alusta loppuun. Päätarkoituksena sovelluksella on seurata käyttäjän kuukausimaksuja ja näyttää niistä käyttäjälle informatiivista tietoa.

Android-sovellus kehitettiin käyttäen Kotlinia kehityskielenä ja Android Studiota editorina. Sovelluksen backend kehitettiin käyttäen C#-kieltä ja mikropalvelu arkkitehtuuria Docker-konttien kanssa. Backend-kehityksen editorina toimi Visual Studio.

Opinnäytetyön tuloksena valmistui toimiva Android-sovellus sekä toimiva backend rajapinta, joiden ominaisuuksiin kuuluivat kuukausimaksujen manuaalinen lisäys sekä automaattinen haku pankkitileiltä. Luotu sovellus hakee pankkitilien tietoja PSD2 Open Banking rajapintojen kautta päätelläkseen mitä kuukausimaksuja käyttäjällä on.

Avainsanat
Android, Kotlin, Material Design, Docker

SAVONIA UNIVERSITY OF APPLIED SCIENCES

THESIS
Abstract

Field of Study Technology, Communication and Transport			
Degree Programme Degree Programme in Information Technology			
Author(s) Riku Wikman			
Title of Thesis Android app (Kotlin) to track subscriptions using PSD2 Open Bandking APIs			
Date	5 June 2018	Pages/Appendices	38/0
Supervisor(s) Mr Jussi Koistinen, Senior Lecturer and Mr Mikko Pääkkönen, RDI Specialist			
Client Organisation /Partners Mikko Pääkkönen			

Abstract

The purpose of this thesis was to create a demo app for an existing business idea. The business idea itself was developed during the yPolku course provided by Savonia University of Applied Sciences, where the purpose is to create a business idea from start to finish. The main purpose of the app is to track the subscriptions of the user and to show meaningful information about them for the user.

The application was developed for Android using Kotlin as the development language and Android Studio as the editor. The backend was developed using C# and microservice architecture with Docker containers. The editor for backend development was Visual Studio.

As a result of this thesis a working Android application and a working backend API were created with features consisting of adding subscriptions manually and retrieving subscriptions from bank accounts. The completed application fetches bank account information through PSD2 Open Banking APIs to decide what subscriptions the application user has.

Keywords
Android, Kotlin, Material Design, Docker

SISÄLTÖ

1 JOHDANTO

Työn tilaajana toimii Savonia-ammattikorkeakoulun TKI-asiantuntija Mikko Pääkkönen. Sovelluksen idea on keksitty opinnäytetyön tekijän Savonian yPolku -opintojen aikana ja sovelluksen liiketoimintapuolta on käyty siellä läpi. Tämän opinnäytetyön tarkoituksena on luoda niin kutsuttu MVP eli Minimum Viable Product, jonka avulla ideaa voidaan demota.

Omien ja tuttujen ihmisten kokemuksien mukaan kuluttajilla on nykyään niin paljon toistuvia maksuja, että kaikkia maksuja on vaikea muistaa. Monesti se on aiheuttanut sen, että maksetaan palveluista joita ei enää käytetä ja palvelun muistaa vasta kun huomaa kuukausimaksun lähteneen tililtä.

Tämän opinnäytetyön aiheena on Android-sovellus, jolla kuluttajat voivat seurata omia toistuvia maksujaan. Android on suosittu mobiilikäyttöjärjestelmä ja vuonna 2017 67%:ssa mobiililaitteista käyttöjärjestelmänä toimi Android (Statista 2017). Android-sovellus on tehty käyttäen Kotlin-ohjelmointikieltä, joka on kohtuullisen uusi JetBrainsin kehittämä kieli. Uutta ohjelmointikieltä käyttäen Android-sovelluksen luominen tuo omat haasteensa ja tämä opinnäytetyö tutkii sovelluksen toteuttamista Kotlin-kielellä.

Käyttäjä voi lisätä Android-sovellukseen itse omia tilauksiaan tai käyttää jo sovelluksesta löytyviä yleisiä tilauksia, kuten vaikka Spotify Premium. Tämän lisäksi käyttäjä voi yhdistää suomalaisen pankkitilinsä sovellukseen, jolloin sovellus hakee tunnistetut toistuvat maksut automaattisesti. Sovellus näyttää tilastoja maksuista ja rahankäytöstä, sekä tarjouksia tietyiltä palveluntarjoajilta.

Opinnäytetyöhön sisältyy ns. backend eli käyttäjälle näkymätön osa sovellusta, yleensä palvelimella tai pilvessä pyörivä ohjelmisto. Tässä tapauksessa backend on luotu käyttäen C#-kieltä ja .NET Core 2 -ohjelmistokehystä. Koko backend pyörii Amazonin AWS pilvipalvelun päällä. Se mahdollistaa sen, että kehittäjän ei tarvitse luoda tai ylläpitää juuri sovellusta varten luotuja palvelimia. Backendin päätarkoitus tässä sovelluksessa on tallentaa käyttäjän tilaukset, jotta käyttäjä voi esimerkiksi puhelinta vaihtaessaan jatkaa sovelluksen käyttöä ilman vanhojen tilausten uudelleenlisäystä.

1.1 Lyhenteet ja määritelmät

Android = Googlen kehittämä käyttöjärjestelmä mobiililaitteille. Käytetään eri valmistajien laitteissa.

iOS = Applen kehittämä käyttöjärjestelmä mobiililaitteille. Käytetään vain Applen itse kehittämissä laitteissa.

Kotlin = JetBrains nimisen yrityksen kehittämä nykyaikainen ohjelmointikieli, jota käytetään mm. Android-sovellusten kehittämiseen.

Backend = Käyttäjälle näkymätön osa sovellusta, joka toimii sovelluksen datan kanssa esimerkiksi palvelimella.

Ohjelmistokehys = Muodostaa rungon sen päälle kehitettävälle ohjelmalle. Apuväline jonka tarkoituksena on nopeuttaa uusien tuotteiden valmistusta. Tarjoaa ohjelman osia, joita ei tarvitse kirjoittaa uudelleen kehityksen aikana.

Microservice = Myös suomeksi mikropalvelu, perinteisesti yhdestä isosta projektista muodostunut iso toteutus voi olla pilkottuna pienempiin mikropalveluihin, jotka hoitavat vain omia tarkemmin määrättyjä tehtäviään

Kontti = Kevyt, erillinen ja suoritettava ohjelmistopaketti

AWS = Amazon Web Services, Amazonin pilvipalvelu

2 VASTAAVIA SOVELLUKSIA

Seuraavassa käydään läpi kokeillut samankaltaiset sovellukset. Androidin ja iOS:n sovelluskaupoista löytyy jo samaan käyttötarkoitukseen luotuja sovelluksia. Nämä sovellukset kuitenkin edellyttävät, että käyttäjä lisää kaikki tilauksensa itse eivätkä ne yleensä ilmoita lähestyvistä maksuista. Usein olemassa olevat sovellukset eivät tallenna tilauksia palvelimelle, joten puhelinta vaihtaessa tilaukset häviävät. Sovellukset näyttävät yleensä vain vähän statistiikkaa, useimmiten vain tämänhetkisen kuukauden kokonaiskulut.

2.1 Bobby iOS-sovellus

Bobby iOS-sovellus on yksinkertainen, tehokas ja tyylikäs sovellus toistuvien maksujen hallintaan. Maksuja voi valita olemassa olevista maksuista, joita voi muokata tai lisätä uuden. Maksulle voi antaa muitakin maksuvälejä kuin kerran kuukaudessa veloitus. Maksut näkyvät eri väreillä päänäkymässä. Lisäksi maksuja voi järjestää omalla haluamallaan tavalla, sekä koko käyttöliittymän voi muuttaa tummaksi halutessaan. Maksut voi myös jaotella valuutan mukaan eli sovelluksessa voi käyttää useita eri valuuttavaihtoehtoja. Tilastoja sovelluksessa ei näytetä muuta kuin kuukauden kokonaiskulut.

Sovellukseen voi asettaa salasanan, jotta maksuja ei näe kukaan muu, tai käyttöön voi ottaa Touch ID:n. Touch ID tarkoittaa Applen sormenjälkitunnistusta ja sitä käyttämällä vain oikean sormenjäljen omaava henkilö pääsee tarkastelemaan toistuvia maksuja. (Bobby 2018)

2.2 Subby – The Subscription Manager Android-sovellus

Subby Android-sovellus on yksinkertainen ja helppokäyttöinen sovellus toistuvien maksujen hallintaan, jonka ominaisuuksiin kuuluu yli 300 valmista tilauspohjaa, yli 160 eri valuuttaa, tumma tai vaalea teema sekä pro-käyttäjille Google Drive varmuuskopiointi. Subby:llä ei voi siis luoda omia tilauksia, vaan mikäli valmiin 300:n tilauspohjan joukosta ei löydy omaa tilausta, joutuu käyttäjä tekemään pyynnön sovelluksen kehittäjälle tilauspohjan lisäämiseksi. Kirjoitushetkellä tilauspohjan pyytäminen onnistuu vain pro-käyttäjiltä. Pro-käyttäjäksi pääsee 1,99€ hintaisella maksulla ja vastineeksi saa varmuuskopioinnin Google Drive palveluun sekä tilauspohjien pyytämisen.

Sovelluksen voi asettaa lähettämään ilmoituksen lähestyvästä maksusta ja sen voi ajoittaa haluamalleen tunnille. Kyseisestä sovelluksesta puuttuu tilausten automaattinen hakeminen tililtä, sekä tarjousten näyttäminen kokonaan. Tämäkään sovellus ei näytä tilastoja muuta kuin sen hetkisen kuukauden kokonaiskulut pienessä tekstissä päänäkymän alaosassa. (Google Play 2018a)

2.2.1 Billy: subscription manager (beta) Android-sovellus

Billy on Androidille luotu sovellus, jolla voi seurata kuukausi- sekä vuosittaistilauksia. Billy on ainakin nimensä mukaan vielä beta-vaiheessa, mutta sovelluksen ominaisuuksista löytyy jo ainakin tilauksen

lisääminen valmiista pohjasta tai sitten oman lisäämällä. Tilaukselle voi myös asettaa oman väritunnisteen. Kuukausittaisen tai vuosittaisen maksuvälin lisäksi sovellukseen on uutuutena tullut puolen vuoden maksuväli.

Sovellus on hyvin pelkistetyn näköinen eikä tästäkään sovelluksesta löydy minkäänlaisia tilastoja maksuista. Sen hetkisen kuukauden maksujen kokonaissumma näkyy päänäkymässä tekstillä. Tilauksia ei voi myöskään tallentaa palvelimelle vaan puhelinta vaihtaessa tilaukset on syötettävä uudelleen. (Google Play 2018b)

2.3 TrackMySubs.com

TrackMySubs.com on verkkopalvelu, jonka avulla on mahdollista seurata tilausmaksujaan. Palvelulla voi seurata erilaisia tilauksia sekä vastaanottaa ilmoituksia lähestyvistä maksuista. Palvelu tarjoaa myös yksityiskohtaisempia tilastoja maksuista sekä rahankäytöstä.

Palvelulla ei ole kuitenkaan minkäänlaista mobiilisovellusta ja ilmainen palvelu on pelkistetty. Ilmaisella käyttäjällä voi seurata vain kymmentä tilausta. Maksamalla kuusi dollaria kuukaudessa, saa seurattua 50:ntä tilausta ja kahdeksalla dollarilla kuukaudessa voi seurata niin montaa tilausta kuin haluaa. (TrackMySubs 2018)

2.4 Hiatus iOS-sovellus

Hiatus on tällä hetkellä vain iOS:lle löytyvä tilausten- ja taloudenhallinta sovellus. Hiatus on ehkä kaikista vastaavista sovelluksista edistynein. Sovellukseen voi kirjautua pankkitunnuksilla ja sovellus hakee tällöin automaattisesti käyttäjän maksut. Sovellus lähettää lähestyvistä maksuista ilmoituksen ja sillä voi myös perua tilauksia suoraan sovelluksesta.

Sovellus voi myös kertoa, mikäli käyttäjä maksaa jostakin tilauksesta sen mielestä liikaa. Tämä on ominaisuus, joka erottaa sovelluksen muista vastaavista sovelluksista. Kun käyttäjä maksaa sovelluksen mielestä liikaa tilauksesta, neuvottelee sovellus automaattisesti alemman hinnan palveluntarjoajan kanssa. Lisäksi sovellus näyttää hieman parempia tilastoja kuin ainoastaan kuukauden kulut. (Hiatus 2018)

3 TOISTUVAT MAKSUT

Toistuviin maksuihin perustuvat tilaukset ovat olleet määrältään jo kauan nousussa. Viimeisen vuosikymmenen aikana mukaan on tullut erilaiset suoratoistopalvelut ja esimerkiksi musiikin suoratoistopalvelujen tilaajamäärät ovat nousseet suuresti. Vuonna 2014 musiikin suoratoistopalveluilla oli Yhdysvalloissa noin 7,7 miljoonaa tilaajaa kun se oli vuoteen 2017 mennessä noussut 35,3 miljoonaan tilaajaan. (Recording Industry Association of America 2018) Videoiden suoratoistopalveluiden käyttäjämäärät ovat myös olleet kovassa nousussa. Vuonna 2016 kehittyneissä maissa oli videoiden suoratoistopalveluiden käyttäjiä hieman yli 200 miljoonaa ja sen on ennustettu tuplautuvan vuoteen 2022 mennessä. (Statista 2018a)

Edellä mainituista luvuista huomataan, että yhä useammalla käyttäjällä on erilaisia tilauksia yhä enemmän ja enemmän. Tämä aiheuttaa sen, että käyttäjä ei usein enää edes muista kaikkia tilauksiaan ja maksaa usein turhaan palveluista, joita hän ei edes käytä. Isossa-Britanniassa ihmiset maksoivat yhteensä 448 miljoonaa puntaa kuukaudessa tilauksista, joita eivät he eivät käyttäneet. (Moneywise 2017) Tämä saattaa johtua siitä, että tilauspalveluiden käyttäjät eivät yleensä näe kokonaiskuvaa tilauksistaan mistään, eivätkä voi hallinnoida tilauksiaan keskitetysti.

Tilauspalveluita on monia erilaisia ja tilauksilla on erilaisia maksuvälejä. Yksi yleisimmistä maksuväleistä on kerran kuukaudessa maksu (Fusebill 2018, Statista 2018b). Muita vaihtoehtoisia maksuvälejä ovat useamman kuukauden välein perittävät maksut, viikoittain tai useiden viikkojen välein perittävät maksut sekä vuosittaismaksut. Tämän takia toistuvia maksuja seuraavassa sovelluksessa pitää olla mahdollisuus asettaa tilaukselle mikä tahansa maksuväli, päivästä kymmeneen vuoteen.

4 KÄYTETYT TEKNIIKAT

Android-sovelluskehitystä ei ole lukittu millekään tietylle käyttöjärjestelmälle kuten esimerkiksi iOS-sovelluskehitys, jota voi periaatteessa tehdä vain macOS-laitteilla. Helpointa kehitystä on tehdä virallisella Android-kehitysympäristöllä, Android Studiolla. Myös muita vaihtoehtoja löytyy ja esimerkiksi ennen Android Studion olemassaoloa Eclipse-kehitysympäristö oli suosittu.

Vastaavasti myös .NET Core 2 kehitystä voi tehdä millä tahansa käyttöjärjestelmällä. Suosittu kehitysympäristö C#:lle ja .NET Corelle on Microsoftin oma työkalu, Visual Studio. Visual Studio löytyy niin Windowsille kuin myös macOS:lle. Myös Linuxilla voi kehittää .NET Core sovelluksia, mutta Visual Studiota ei löydy Linuxille vaan kehittäjän täytyy käyttää jotain muuta kehitysympäristöä ja ajaa .NET Corea esimerkiksi komentoriviltä.

4.1 Android Studio

Android Studio on Googlen ja JetBrainsin yhdessä JetBrainsin IntelliJ-kehitysympäristön päälle kehittämä Android-kehitysympäristö. (Ducrohet Xavier, Norbye Tor ja Chou Katherine 2013) Google listaa seuraavat ominaisuudet Android Studion keskeisimmiksi:

- Joustava Gradleen perustuva ohjelman kääntäminen
- Nopea ja paljon ominaisuuksia sisältävä emulaattori
- Yhdistetty ympäristö, jolla voi kehittää kaikille eri Android versioille
- Instant Run, jolla voi puskea ohjelman muutokset käynnissä olevaan ohjelmaan ilman kääntämistä
- Kooditemplaatteja ja GitHub integraatio yleisien ominaisuuksien tekoon ja esimerkkikoodien hakemiseen
- Kattavat testaustyökalut ja ohjelmistokehykset
- Koodin tarkistustyökalut suorituskykyongelmien, versioiden yhteensopivuuksien ja muiden ongelmien automaattiseen löytämiseen
- C++ ja NDK tuki
- Sisäänrakennettu tuki Google Cloud Platformille

LUETTELO 1. (Google 2018a)

Android Studiolla kehittäjä voi siis kehittää sovellustaan alusta loppuun, testata sitä mukana tulevalla emulaattorilla ja julkaista sovelluksen Googlen Play Kauppaan.

4.2 Visual Studio

Visual Studio on Microsoftin kehittämä kehitysympäristö, jossa voi käyttää monia eri kieliä ja sovelluskehyksiä. Esimerkkejä suoraan tuetuista kielistä ovat C#, C++, F#, Python ja JavaScript. (Microsoft 2018a) Visual Studiosta on olemassa eri versioita, joista osa on maksullisia. Ilmainen ja tässä

opinnäytetyössä käytetty versio on nimeltään Community Edition. Se on tarkoitettu yksittäisille kehittäjille, jotka voivat käyttää sitä niin ilmaisten kuin maksullisien sovelluksien luomiseen. Yrityksille on taas olemassa Professional ja Enterprise versiot, jotka maksavat. (Microsoft 2018b)

Visual Studiolla voi debugata eli virheenkorjata sovelluksia editorista löytyvällä debuggerilla. Se tapahtuu asettamalla niin kutsuttu breakpoint eli pysäytyspiste, johon ohjelman suoritus pysähtyy. Kehittäjä voi sitten suorittaa ohjelmaa rivi riviltä ja Visual Studio näyttää muun muassa eri muuttujien arvot sekä tietoja suorituskyvystä. (Microsoft 2018c)

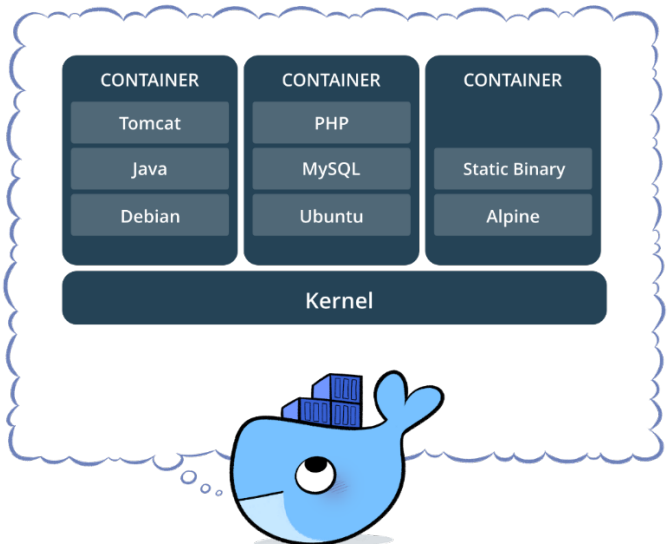
4.3 Amazon Web Services

Amazon Web Services eli AWS on Amazonin tarjoama pilvipalvelualusta. AWS tarjoaa monia erilaisia palveluja kehittäjille ja yrityksille sovelluksiensa skaalaamiseen. Näitä palveluja yhdistelemällä kehittäjä voi luoda sovelluksensa kokonaan pilveen. Esimerkkejä näistä palveluista ovat esimerkiksi EC2, jota voi ajatella ns. virutaalikoneina pilvessä joihin kehittäjä voi laittaa sovelluksensa samoin kuten palvelimelle. Toinen AWS:n tarjoama palvelu on tietokantapalvelu pilvessä, RDS. Kummassakin palvelussa Amazon voi skaalata tehoja ja tallennustilaa automaattisesti, mikäli sovellus sitä vaatii. (Amazon Web Services 2018a)

Laskutusmalli kaikissa AWS:n palveluissa eroaa perinteisistä vuokrattavista palvelimista siinä, että vuokrattava palvelin maksaa aina saman verran esimerkiksi kuukaudessa, mutta AWS laskuttaa vain käytetystä laskenta-ajasta. Näin ollen, jos sovellus ei ole päällä, ei se kerrytä laskua. (Amazon Web Services 2018b)

4.4 Docker

Docker on yritys ja ns. konttialusta, joka on ollut yleistämässä kontteja kehitysmaailmassa. Kontti on kevyt, erillinen ja suoritettava ohjelmistopaketti, joka sisältää kaiken mitä sovellus tarvitsee: koodin, ajoympäristön, järjestelmätyökalut ja asetukset. Kontit eristävät sovelluksen ympäristöstään ja tekevät näin helpoksi julkaisun ja kehittämisen eri alustoilla. (Docker 2018)



KUVA 1. Docker 2018

Docker esiteltiin yleisölle ensimmäistä kertaa vuonna 2013 PyCon tilaisuudessa (Laura Bernheim 2017). Avoimelle lähdekoodille Docker julkaistiin saman vuoden maaliskuussa (Abel Avram 2013).

4.5 Android

Android on Googlen kehittämä käyttöjärjestelmä mobiililaitteille. Käyttöjärjestelmää käytetään useiden eri valmistajien laitteissa. Android on maailman suosituin mobiilikäyttöjärjestelmä ja esimerkiksi Yhdysvalloissa vuonna 2017 noin 61% mobiililaitteista käytti Androidia. (Statista 2017) Android perustuu Linux-käyttöjärjestelmään ja on avoimen lähdekoodin ohjelmisto.

Vaikka Android onkin avoimen lähdekoodin ohjelmisto, se ei ole yhteisön kehittämä kuten yleensä, vaan pääasiassa Google ja Open Handset Alliance kehittävät Androidia. (Android.com 2018) Käytännössä tämä on tarkoittanut sitä, että Google on pitkälti päättänyt mitä Androidissa on ja miten sitä kehitetään.

4.6 Kotlin

Kotlin on staattisesti tyypitetty avoimen lähdekoodin moderni ohjelmointikieli. Sen on kehittänyt JetBrains, joka löytyy myös Android Studion taustalta. Kotlin toimii JVM:n päällä, mutta kääntyy myös JavaScriptiksi tai voi käyttää LLVM-teknologiaa. (Kotlinlang 2018)

Kotlin on vuodesta 2017 ollut yksi virallisista Android kehityskielistä. Kotlinin syntaksi ei ole suoraan yhteensopivaa Javan kanssa, mutta Kotlinilla voi käyttää Javan kirjastoja, sillä ne molemmat kääntyvät JVM:lle. JetBrainsin mukaan Kotlinin käyttäminen vähentää koodirivien määrää noin 40%:lla. Lisäksi Kotlin vähentää muun muassa Java-ohjelmille tuttuja null pointer virheitä, sillä Kotlin pakottaa kehittäjän käsittelemään null arvojen mahdollisuuden, eikä käännä ohjelmaa, ellei kehittäjä tee niin. (Shafirov 2017)

4.7 ASP.NET Core 2 ja .NET Core 2

ASP.NET Core 2 on Microsoftin kehittämä ilmainen ja avoimen lähdekoodin verkkosovellusten sovelluskehys, joka toimii niin täydessä .NET Framework:ssä kuin monen alustan .NET Core 2:ssa. Se voidaan nähdä jatkeena ASP.NET sovelluskehykselle.

Ero .NET Frameworkin ja .NET Coren välillä on se, että .NET Framework sisältää enemmän ominaisuuksia, mutta toimii vain Windowsilla. .NET Core taas on ilmainen ja avoimen lähdekoodin alusta, joka toimii niin Linuxilla, macOS:lla kuin Windowsillakin. (Microsoft 2018d)

4.8 SQL Server

SQL Server on Microsoftin kehittämä relaatiotietokanta. Relaatiotietokannassa tieto on jaettu tauluihin, jotka ovat yhteyksissä toisiinsa. Useimmiten yhdistämiseen käytetään ns. pää -ja vierasavaimia.

SQL Serveristä on olemassa eri versioita, joista osa on maksullisia ja osa ilmaisia. Tässä opinnäytetyössä on käytetty ilmaista SQL Server Express -versiota. Muita versioita ovat esimerkiksi Enterprise ja Standard versiot. (Microsoft 2018e)

4.9 Open Banking API

Euroopan parlamentin ja neuvoston direktiivin (EU) 2015/2366 mukaan tilinpitäjäpankkien pitää mahdollistaa kolmansille osapuolille pääsy asiakkaidensa tileille asiakkaiden suostumuksen mukaan. Direktiivi tuli voimaan 13.1.2018. (Finanssivalvonta 2018a)

Käytännössä direktiivi on tehnyt sen, että pankit ovat tehneet ns. Open Banking rajapintoja, joilla kolmannen osapuolen kehittäjät pääsevät tilitietoihin käsiksi. Rajapinnat ovat useimmiten REST-rajapintoja. Suomessa toimivista pankeista opinnäytetyön tekoaikaan (helmikuu 2018 – toukokuu 2018) toimivat rajapinnat löytyivät ainoastaan Nordealta ja OP:lta.

5 ONGELMAT KEHITYKSESSÄ

Sovelluksen kehitysvaiheessa tuli vastaan monia ongelmia sekä haasteita. Suurimmaksi osaksi ongelmat kohdistuivat kehitysalustoihin ja niiden toimintaan. Mikään haasteista ei kuitenkaan aiheuttanut ylitsepääsemättömiä esteitä tai tilanteita, joissa sovellustoteutusta olisi pitänyt muuttaa.

5.1 Dockerin ja Android Studion yhteensopivuus

Docker Windowsille vaatii toimiakseen Hyper-V:n, joka on Microsoftin virtualisointiohjelma joka luo ja hallitsee virtuaalikoneita. Android Studion emulaattori taas vaatii, että Hyper-V on pois päältä. Näin ollen kumpiakin ei voi ajaa samaan aikaan samalla tietokoneella. Jos Hyper-V:n haluaa kytkeä päälle tai pois päältä, täytyy käyttäjän käynnistää tietokone uudelleen. Tämä aiheutti kehityksessä paljon päänvaivaa eikä hyvää ratkaisua löytynyt.

5.2 Open Banking rajapintojen rajoitteet

Pankkien rajapintojen dokumentaatiossa on paljon parannettavaa. Pääasiassa dokumentaatio näyttää generoidulta eikä monista päätteistä ole minkäänlaisia esimerkkejä. Tämä jättää paljon kehittäjän arvailun sekä kokeilun varaan.

Toinen ongelma on, että joissain rajapinnoissa on vielä bugeja rajapinnan vastauksissa. Esimerkiksi opinnäytetyön tekovaiheessa Nordean rajapinta antoi tilitapahtumat sivutettuna JSON:na, mutta linkki seuraavaan ”sivuun” oli väärässä muodossa eikä näin ollen toiminut, vaan kehittäjän piti itse muokata linkki oikeaksi.

5.3 Kotlin ja Android Studio

Kotlin ja Android Studio päivittyvät nopeaan tahtiin ja miltei joka kerta kun toinen niistä päivittyi, rikkoi se jonkin osan kääntöprosessista, joka johti pitkään ratkaisun hakemiseen hakukoneilla. Yksi esimerkki kyseisestä ongelmasta on, kun Kotlin päivittyi versioon 1.2.40 ja rikkoi yhteensopivuuden Googlen oman Room-kirjaston kanssa. Ratkaistakseen ongelman, piti kääntöprosessia muokata ja lisätä ehtoja vain Kotlinin takia.

```
subprojects {
  configurations.all {
    resolutionStrategy {
      eachDependency {
        if (requested.name == "kotlin-compiler-embeddable") {
          useVersion("1.2.30")
        }
      }
    }
  }
}
```

KUVA 2. Kotlin 1.2.40 Room-kirjaston yhteensopivuusongelman korjaus

5.4 Amazon RDS Security Groups

Amazon RDS pilvipalvelussa on turvallisuuden vuoksi asetettu IP:t, joista voi ottaa yhteyden tietokantaan. Tietokantaa luodessa Amazon lisäsi automaattisesti oman IP:ni luotettuihin osoitteisiin, joista saa yhteyden tietokantaan. Tämän lisäksi IP-osoitteita voi itse lisätä yksittäin tai antamalla joukon luotettuja osoitteita (Amazon Web Services 2018c).

Kesken kehityksen oma IP-osoitteeni muuttui ja yhteys tietokantaan ei enää onnistunut kehityskoneelta. Kun virheviesti oli niin heikko, oli vaikea yhdistää syitä siihen miksi tietokantayhteys ei enää toiminutkaan. Tämän monien tuntien vianselvittelyn voisi välttää selvällä virheviestillä ”yhteys tästä IP-osoitteesta ei ole sallittu”.

6 PSD2 OPEN BANKING RAJAPINNAT

Open Banking rajapinnoilla tarkoitetaan Euroopan parlamentin ja neuvoston PSD2-direktiivin mukaisia pankkien toteuttamia rajapintoja. PSD2-direktiivin tarkoituksena on mahdollistaa pankkien asiakkaalle maksaminen ja tiedon saaminen omista tileistään myös muiden kuin omien pankkiensa kautta. Direktiiviin perustuvat lakimuutokset tulevat voimaan 13.1.2018, mutta kaikki muutokset eivät kuitenkaan tule käyttöön samaan aikaan. Rajapinnoissa toimiluvan saaneilla kolmannen osapuolen sovelluskehittäjillä on pääsy asiakkaan pankkitileille, mikäli asiakas antaa suostumuksensa. Direktiivi tarkoittaa myös, että kolmannen osapuolen palveluntarjoajan ei tarvitse tehdä erillistä sopimusta tilinpitäjäpankin kanssa, vaan kaikilla luvan saaneilla palveluntarjoajilla on pääsy rajapintoihin. Tässä opinnäytetyössä käytettiin Suomessa toimivien pankkien rajapintoja. (Finanssiala ry 2018)

Toteutusvaiheessa toimivat rajapinnat löytyivät ainoastaan Nordealta ja OP:lta. Molemmat rajapinnat olivat ainakin ulkopuolisen kehittäjän silmään hyvin samanlaisia, mikä johtuu pitkälti myös EU direktiivistä. Rajapintoihin täytyy rekisteröidä oma sovellus ja sovellus pitää rekisteröidä vielä erikseen joko tilitapahtumien tarkastelua varten tai maksujen suorittamista varten. Näin asiakas tietää myös itse mihin on antanut oikeudet kolmannen osapuolen sovellukselle, eikä esimerkiksi voi käydä tilannetta, jossa asiakas päästää tietämättään jonkun suorittamaan maksuja puolestaan. Ainakin molempiin käytettyihin rajapintoihin tunnistaudutaan OAuth2-tunnistautumisella, esimerkiksi Nordean tapauksessa Tunnusluvut-sovelluksella. Kun tunnistautuminen onnistuu, saadaan salaiset avaimet, jotka täytyy liittää rajapintakutsujen headereihin.

Rajapinnat ovat REST-rajapintoja, joissa palautetaan tiedot JSON-muodossa. Kun salainen avain on pyynnön headerissä, voidaan kysellä kyseisen käyttäjän tietoja. Vaihtoehtoja on joko kysyä käyttäjän eri tilit, yksityiskohtaisempaa tietoa yhdestä tilistä tai yhden tilin kaikki tilitapahtumat. Tilitapahtumat palautetaan sivutettuna JSON:na, sillä niitä saattaa olla tuhansia. Käytännössä sivutetussa JSON:ssa on linkki aina seuraavaan sivuun, joka on vain kutsu rajapintaan uusilla parametreilla.

Rajapintoihin rekisteröidyttäessä kehittäjällä on pääsy vain demokäyttäjään. Tämä tarkoittaa, että esimerkiksi OAuth2-tunnistautuminen ei toimi, vaan demokäyttäjälle on annettu valmiiksi avaimet. Mikäli kehittäjä haluaa pääsyn oikeisiin pankkitietoihin, täytyy hänen hakea paikallista FSA-lisenssiä. Suomessa lisenssi haetaan Finanssivalvonnalta. (Finanssivalvonta 2018b)

Endpoint	Suported HTTP Methods
/accounts	GET
/accounts/{ACCOUNT_ID}	GET
/accounts/{ACCOUNT_ID}/transactions	GET

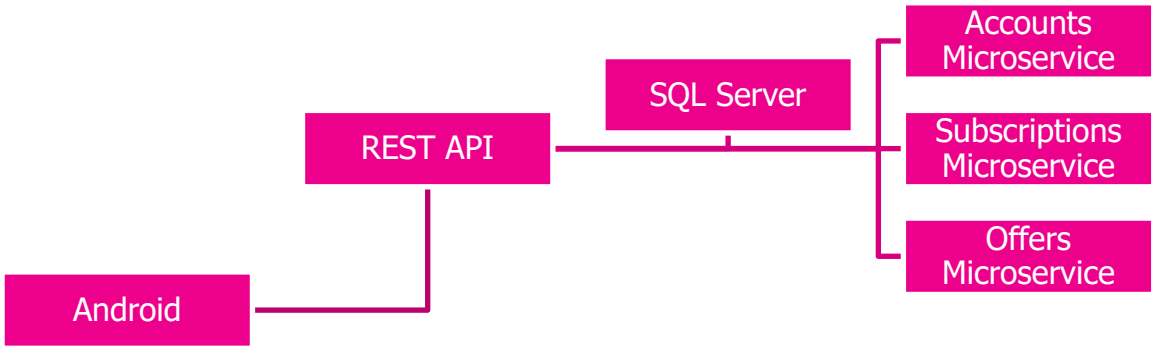
KUVA 3. Nordean Open Banking rajapinnan Accounts API:n pääteet. (Nordea 2018)

7 ARKKITEHTUURI

Haluttuna lopputuotoksena oli Android mobiilisovellus, joka toimii demona yritysidealle. Demoon haluttuja ominaisuuksia olivat:

- Kuukausimaksujen automaattinen haku pankkitililtä
- Kuukausimaksujen manuaalinen lisäys
- Yleistieto kuukauden menoista
- Tarjousten lähettäminen / vastaanottaminen

Kokonaisuus toteutettiin tekemällä backend rajapinta, josta Android-sovellus hakee maksut, tarjoukset ja käyttäjän pankkitilin kuukausimaksut. Backend itse on jaettu pienempiin niin kutsuttuihin microserviceihin, joka mahdollistaa tietyn osan skaalaamisen tarvittaessa.



KUVA 4. Sovelluksen arkkitehtuuri

7.1 Android-sovellus

Android-sovellus on käyttäjälle näkyvä osa sovelluskokonaisuutta. Sovellus tallentaa joitain tietoja omaan paikalliseen SQLite kantaan. Pääasiassa sovellus kuitenkin hakee tiedot pilvessä sijaitsevalta tietokantapalvelimelta. Android-sovellus ei kommunikoi suoraan tietokantapalvelimen kanssa, vaan pyytää tietoja REST-rajapinnalta. Samoin sovellus ei myöskään keskustele suoraan mikropalveluiden kanssa, vaan lähettää pyynnöt REST-rajapintaan.

7.2 REST-rajapinta

REST-rajapinta toimii sovelluksen käskynjakajana. Rajapinta päättelee Android-sovellukselta tulleista pyynnöistä, mitä mikropalveluja pyynnöt tarvitsevat ja kutsuu niitä HTTP:n avulla. Rajapinnan vastuulla on myös käyttäjän kirjautuminen ja autentikaatio.

7.3 Mikropalvelut

Sovelluksen eri osat on jaettu mikropalveluihin, joka mahdollistaa niiden skaalaamisen tarvittaessa. Esimerkkinä tilanne, jossa monet käyttäjät haluavat löytää kuukausimaksut tilitapahtumistaan samanaikaisesti. Kyseisessä tilanteessa tilitapahtumien mikropalvelu, Accounts Microservice, käyttää paljon prosessointiaikaa sekä muistia. Jos backendiä ei olisi jaettu mikropalveluihin, jouduttaisiin koko backendiä skaalaamaan pyyntöjen käsittelyn vuoksi joka tulisi kalliiksi ja turhia resursseja käytettäisiin. Mikropalveluarkkitehtuurissa voidaan skaalata pelkkää tilitapahtumien mikropalvelua ja pitää muut palvelut samoillaan.

Mikropalvelut on jaettu kolmeen: tilitapahtumia käsittelevä mikropalvelu, kuukausimaksuja käsittelevä mikropalvelu sekä tarjouksia käsittelevä mikropalvelu. Kukin näistä hoitaa vain oman vastuualueensa tehtäviä.

Accounts Microservice eli tilitapahtumien mikropalvelu pyytää käyttäjän tilitapahtumat pankkien rajapinnoista, tutkii niistä tunnetut kuukausimaksut ja palauttaa tiedon käyttäjän maksuista. Mikropalvelu osaa tarvittaessa kutsua eri pankkien erilaisia rajapintoja.

Subscriptions Microservice eli kuukausimaksujen mikropalvelu käsittelee käyttäjän maksuja ja tallentaa niistä tietoja tietokantaan. Näin mikropalvelu osaa myös palauttaa käyttäjän maksut, mikäli käyttäjä on hukannut ne sovelluksestaan.

Offers Microservice eli tilausten mikropalvelu käsittelee yritysten tarjoamia tarjouksia ja huolehtii niiden näyttämisestä oikeille käyttäjille. Tämä mikropalvelu ei kuitenkaan ole yhteydessä valmiiseen demosovellukseen johtuen siitä, että tarjouksien rakenteesta ei päästy täydelliseen selvyyteen kehitysvaiheessa.

Kaikki mikropalvelut ovat Docker kontteja ja siis täysin itsenäisiä paketteja. Docker kontit sijaitsevat Amazonin EC2-palvelussa, joka on Amazonin virtuaalikoneiden pilvipalvelu. EC2 voidaan asettaa skaalaamaan kontteja automaattisesti liikenteen mukaan. (Amazon Web Services 2018d)

7.4 SQL Server

Sovelluksen käyttämät tiedot, mm. käyttäjän kuukausimaksut, tallennetaan tietokantaan. Tietokantana toimii Microsoftin SQL Server, joka sijaitsee Amazonin RDS-palvelussa. RDS on Amazonin pilvipalvelu tietokannoille, jota on helppo hallita web-käyttöliittymän kautta. SQL Serverin tilausmalleista opinnäytetyössä on käytössä SQL Server Express, eli ilmainen versio.

Tietokantapalvelimelle on erikseen asetettu IP-osoitteet, joista sille voi tehdä pyyntöjä. Käytännössä se tarkoittaa, että vain REST-rajapinnan sekä mikropalveluiden IP-osoitteet ovat sallittuja. Kehityksessä myös kehitystietokoneen IP-osoite oli sallittujen listalla.

8 ANDROID TOTEUTUS

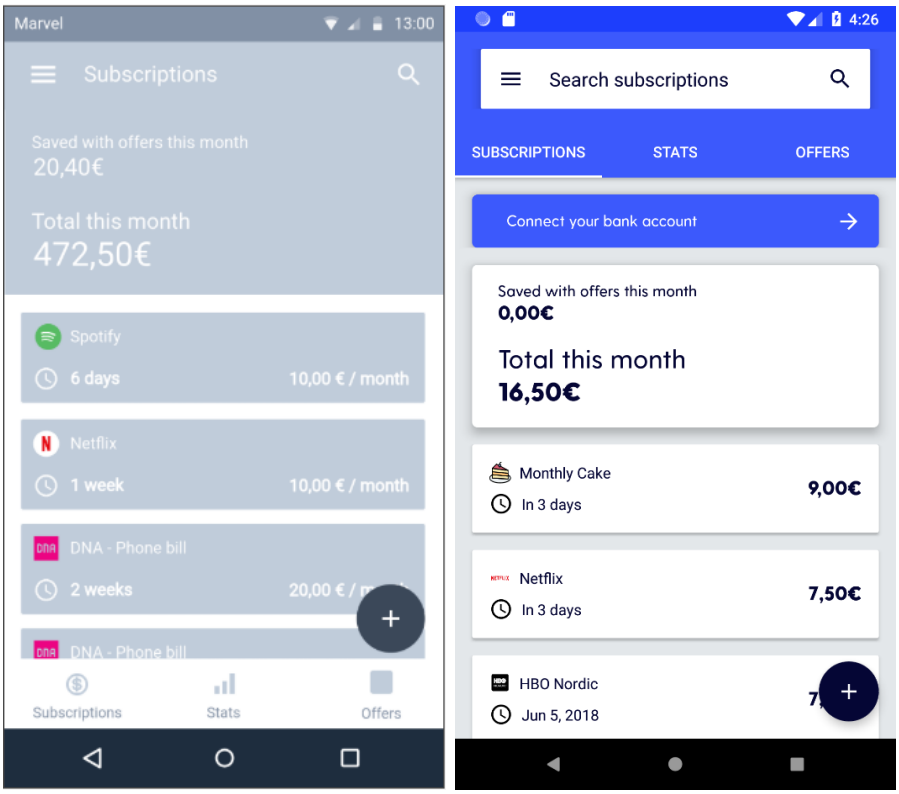
Android-sovelluksen oli tarkoitus tulla demokäyttöön ja sen ei tarvinnut toimia kaikilla mahdollisilla Android-versioilla. Tästä syystä sovelluksen vaatima minimiversio Androidille on 8.0. Tämä mahdollistaa demoamisen uusilla laitteilla ja samalla miltei uusimpien ominaisuuksien käytön demossa.

Sovelluksesta jäi pois esimerkiksi statistiikan näyttäminen ja tarjouksien näyttäminen monesta eri syystä. Tarjouksien näyttämisen käyttöliittymää ei tehty, koska ei ollut kehitysvaiheessa vielä selvää kuvaa siitä, miten tarjoukset tulevat toimimaan liiketoiminnan kannalta. Statistiikka taas päätettiin jättää pois, jotta muut ominaisuudet saadaan varmasti valmiiksi.

8.1 Käyttöliittymä ja sovelluslogiikka

Sovelluksen käyttöliittymä on suunniteltu Googlen Material Design -tyylin mukaan. Material Design määrittelee minkälaisia elementtejä, värejä, fontteja sekä animaatioita sovelluksen käyttöliittymä sisältää. Se hakee inspiraatiota oikeasta maailmasta ja siitä, miten asiat heijastavat valoa sekä luovat varjoja alleen. (Google 2018b) Material Designin tarkoituksena on myös yhtenäistää Android-sovelluksien käyttöliittymiä, niin että eri sovelluksien käyttöliittymät olisivat tuttuja käyttäjille.

Käyttöliittymän suunnittelussa käytettiin aluksi Marvel nimistä käyttöliittymän suunnitteluohjelmaa. Kyseisessä ohjelmassa oli karkeita ensimmäisiä mielikuvia siitä, miltä sovelluksen käyttöliittymän pitäisi näyttää. Tässä vaiheessa käyttöliittymään ei suunniteltu vielä värejä.



KUVA 5. Käyttöliittymän ensimmäisiä suunnitelmia.

KUVA 6. Sovelluksen päänäyttö

Kun sovelluksesta oli jonkinlaisia mielikuvia suunniteltuna, vaihdettiin Android Studioon ja oikean käyttöliittymän tekoon. Ensimmäiset mielikuvat ja suunnitelmat käyttöliittymästä muuttuivat nopeasti, kun oikeaa käyttöliittymää alkoi rakentaa. Lisäksi sovelluksen ns. business-logiikka muutti käyttöliittymää hieman.

8.1.1 RecyclerView

Päänäytöllä tilaukset, kokonaissummat ja esimerkiksi pankkitilin yhdistämisilmoitus elävät kaikki samassa RecyclerView-komponentissa. RecyclerView on kuten ListView eli komponentti, jonka avulla voidaan näyttää suuri määrä tietoja tai elementtejä pienemmässä näytössä. RecyclerView hoitaa vierittämisen tarpeen tullen. RecyclerViewillä kuten ListViewilläkin täytyy aina olla niin kutsuttu adapteri, joka kertoo sille mitä listassa pitäisi olla ja minkälaisia layout-tiedostoja käytetään listajäsenten näyttämiseen. (Google 2018d) RecyclerViewin ja ListViewin ero on siinä, että RecyclerView pakottaa käyttämään ViewHoldereita, joissa pidetään yllä viittauksia listan jäseniin ja näin ollen tekee listan vierittämisestä paljon tehokkaampaa.

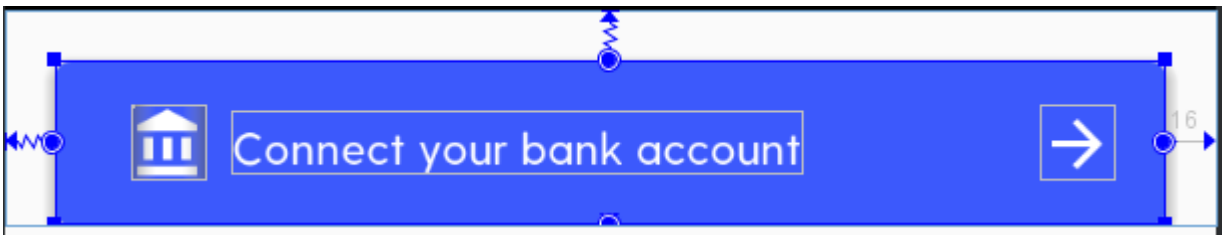
Päänäytön RecyclerViewissä on kolmea eri tyyppistä listan jäsentä: muistutus, yleisnäkymä ja kuukausimaksu. Näillä kaikilla kolmella tyypillä on erilainen näkymä ja se saadaan toteutettua RecyclerViewissä helposti. Normaalisti RecyclerViewissä on vain yksi ViewHolder, jossa on viittaukset esimerkiksi listajäsenen tekstikenttiin. Tälle ViewHolderille annetaan sitten joku XML layout-tiedosto, jossa on kuvattu minkälainen listajäsenen pitäisi olla. Kolmen tyypin tapauksessa tehdään kuitenkin kolme erilaista ViewHolderia, joille kaikille annetaan erilainen layout-tiedosto. RecyclerViewin adapterista kerrotaan *getItemViewType*-metodia käyttäen, mitä ViewHolderia kuuluu käyttää. Tässä sovelluksessa tarkoitus on, että ensimmäinen jäsen on aina muistutus pankkitilistä, toinen jäsen on yleisnäkymä ja kaikki muut ovat kuukausimaksuja. Alla on kuva *getItemViewType*-metodin toteutuksesta.

```
override fun getItemViewType(position: Int): Int {
    return when {
        position > 1 → SUBSCRIPTION_TYPE
        position == 1 → GLANCEVIEW_TYPE
        else → BANK_ACCOUNT_REMINDER_TYPE
    }
}
```

KUVA 7. getItemViewType-metodi

8.1.2 ConstraintLayout

Käyttöliittymä on rakennettu käyttäen ConstraintLayout-komponentteja, mikä tarkoittaa, että näytöllä näkyvät elementit ovat yhteydessä toisiinsa ja sijoittuvat toistensa mukaan. Kehittäessä komponentit yhdistetään toisiinsa nuolia vetämällä tai vaihtoehtoisesti näkymän XML-tiedostoon merkiten.



KUVA 8. ConstraintLayout ja yhdistäminen

ConstraintLayoutia käyttäessä elementeille kerrotaan kuinka paljon niiden pitäisi jättää tilaa toiseen elementtiin. Yllä olevassa kuvassa (KUVA 8.) siniselle elementille on kerrottu, että sen täytyy jättää oikealle puolelle 16 dp:tä (Density-independent Pixel) väliin.

8.1.3 Fontit ja värit

Sovelluksen yleisilmettä parantamaan hankittiin maksullinen fontti. Fontiksi valikoitui Greycliff -niminen sans-serif fontti. Sen voi nähdä esimerkiksi päänäytöllä (KUVA 6.) kuukausimaksujen hinnoissa. Android-sovellukseen fonttien lisääminen on helppoa ja se vaatii vain fonttitiedostojen siirtämisen projektin alle *assets/fonts* kansioon. Joskus tämä kansio pitää ensin luoda. Tämän jälkeen fonttia voi käyttää esimerkiksi näkymien XML-tiedostoissa.

```
android:fontFamily="@font/greycliff_cf_medium"
android:text="Connect your bank account"
```

KUVA 9. Oman fontin käyttö XML-tiedostoissa

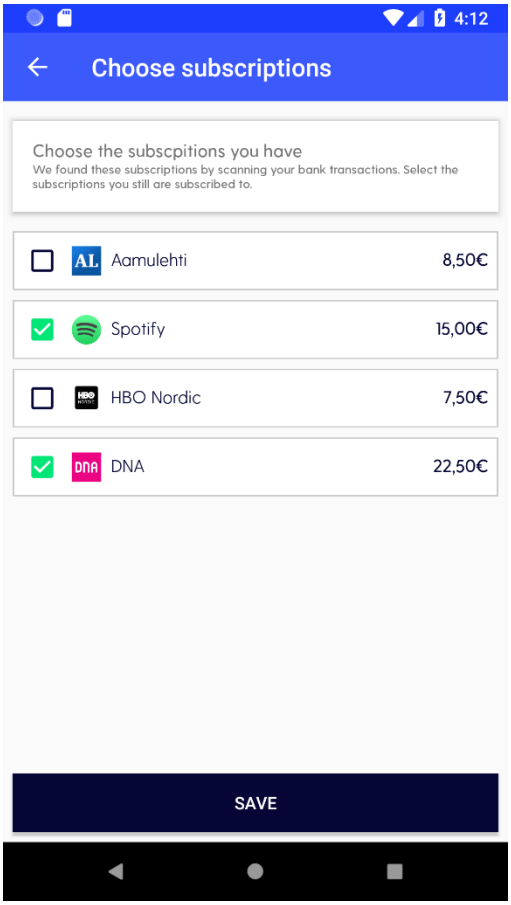
Sovelluksessa on valittu omat värit, jotka toistuvat joka puolella sovellusta. Nämä värit ovat tallennettuna *values/colors.xml* tiedostossa, jolloin niitä voi käyttää näkymien layout-tiedostoissa ilman, että muistaa aina kyseisen värikoodin. Tämä myös helpottaa tilannetta, jossa värit halutaankin muuttaa. Tällöin värit täytyy vaihtaa ainoastaan *colors.xml* tiedostossa. Värejä on käytössä kolme, pääväri, tummempi pääväri ja korostusväri. Päänäytöllä (KUVA 6.) on näkyvissä kaikki kolme; ohut yläpalkki edustaa tummempaa pääväriä, kun taas sen alla oleva isompi palkki edustaa pääväriä. Korostusväri löytyy esimerkiksi tekstistä ja tilauksen lisäysnapista.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#3C59FC</color>
    <color name="colorPrimaryDark">#1d3efc</color>
    <color name="colorAccent">#050635</color>
    <color name="textColor">#050635</color>
</resources>
```

KUVA 10. Värit määriteltynä omassa tiedostossaan

8.1.4 Maksujen hakeminen ja luominen

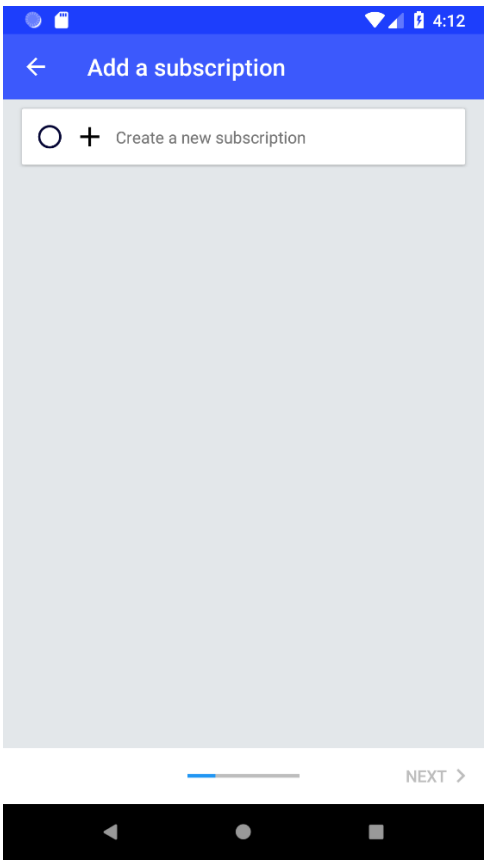
Kun käyttäjä hakee omat kuukausimaksunsa pankkitililtään, näyttää sovellus hänelle listan tunnistetuista maksuista. Tältä listalta käyttäjä voi sitten valita maksut, jotka hän haluaa lisätä sovellukseen. Lista on toteutettu RecyclerView-komponentilla ja se pitää yllä tietoa valituista listan jäsenistä.

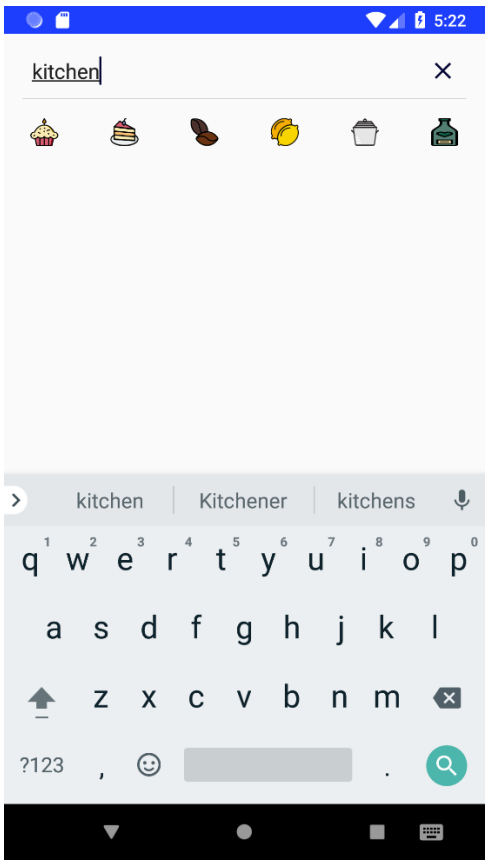


KUVA 11. Pankkitililtä tunnistetut maksut

Koska kaikkia maksuja ei kuitenkaan voida tunnistaa automaattisesti, täytyy käyttäjän pystyä lisäämään maksuja myös manuaalisesti. Manuaalinen maksujen lisäys tapahtuu niin kutsutussa stepperissä, missä käyttäjällä on eri askelia joissa hän täyttää eri tietoja. Stepper on toteutettu käyttäen Android Material Stepper -nimistä avoimen lähdekoodin kirjastoa. (StepStone Tech 2018) Kirjaston avulla pystyy luomaan valmiin stepperin, jossa jokainen näyttö on oma fragmenttinsä. Fragmentillä tarkoitetaan pienempää osaa Activityä, jolla taas tarkoitetaan näytöllä näkyvää näkymää. (Google 2018c)

Manuaalisesti lisättäessä käyttäjä pääsee myös valitsemaan ikonin lisäämälleen tilaukselle. Ikonit tulevat sovelluksen mukana ja ne ovat fontin lailla ostettu sovellusta varten. Käyttämällä maksullisia ikoneja saadaan paremman laadun tunne sovellusta käyttäessä. Ikoneja voi hakea löytääkseen oman mieleisen ikonin kuvaamaan maksua.

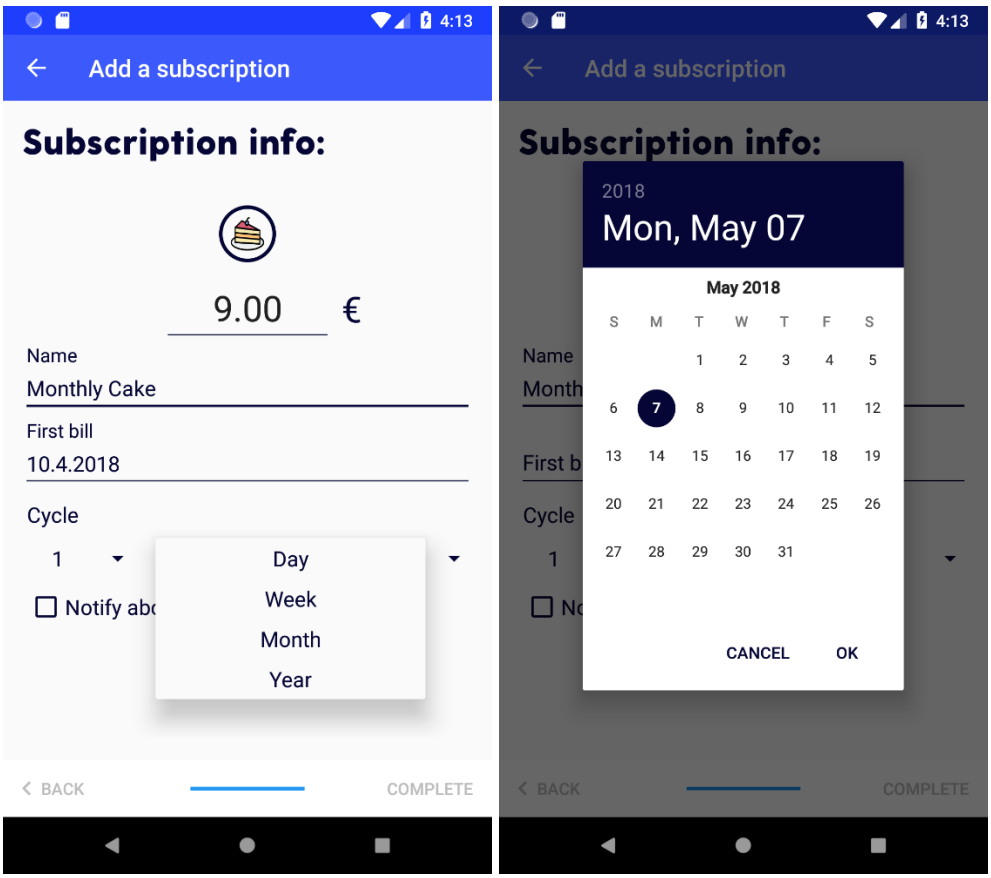




KUVA 12. Manuaalinen maksun lisäys

KUVA 13. Ikoneja lisäämässä

Lopuksi käyttäjä asettaa vaaditut tiedot maksuun. Päivämääriä valitessa käyttäjälle näytetään kalenterinäkymä, josta voi valita haluamansa päivän helposti. Lisäksi käyttäjä valitsee kahdesta alasvetovalikosta haluamansa aikavälin, jolla maksu toistuu. Ensimmäisestä alasvetovalikosta käyttäjä valitsee lukumäärän ja toisesta aikavälin jolla maksu toistuu. Näin voidaan valita esimerkiksi kolmen päivän tai kahden kuukauden välein.



KUVA 14. Maksuvälin valitseminen

KUVA 15. Päivämäärän valitseminen

Maksun maksuväli perustuu Javan *Calendar*-luokkaan siten, että päivän arvoksi alasvetovalikkoon on asetettu *Calendar.DAY_OF_MONTH*-vakioarvo, viikon arvoksi *Calendar.WEEK_OF_YEAR*-vakioarvo ja niin edelleen. Jotta näille arvoille on saatu myös teksti alasvetovalikkoon, on alasvetovalikon listajäseneksi asetettu oma luokka SubscriptionScheduleType, joka on Kotlinin *enum class*-tyyppiä. *Enum class* on vain Kotlinin tyyli esittää monista ohjelmointikielistä tuttu *enum*-käsite.

```
enum class SubscriptionScheduleType(val type: Int, val text: String) {
    DAY(Calendar.DAY_OF_MONTH, text: "Day"),
    WEEK(Calendar.WEEK_OF_YEAR, text: "Week"),
    MONTH(Calendar.MONTH, text: "Month"),
    YEAR(Calendar.YEAR, text: "Year")
}
```

KUVA 16. Maksuvälin enum class

8.1.5 NetworkImageView ja paikalliset kuvat

Päänäytöllä (KUVA 6.) näkyvien tilausten ikonit saattavat tulla joko verkosta tai paikallisesti puhelimesta. Jotta ikoni voidaan näyttää sen tullessa verkosta, käytetään ikonin näyttämiseen Googlen Volley-kirjastosta löytyvää NetworkImageView-komponenttia. NetworkImageView hakee kuvan annetusta osoitteesta, lataa ja lopuksi näyttää kuvan. NetworkImageViewiin voidaan asettaa myös ns. placeholder kuva, joka näkyy ennen kuin verkosta tullut kuva on ladattu.

NetworkImageView ei tue suoraan paikallisten kuvien näyttämistä, joten tämä ongelma on kierretty käyttämällä paikallisia kuvia placeholder kuvina ja verkosta tulevia kuvia varsinaisina NetworkImageViewin kuvina. Maksua manuaalisesti lisättäessä kuva tallennetaan maksuun lisäämällä teksti *"localDrawable:///"* kuvan sijaintiin. Täten ollen, kun kuva pitäisi näyttää, asetetaan kuva placeholder kuvaksi mikäli "localDrawable" teksti löytyy kuvan sijainnista ja muutoin haetaan se netistä.

```
if (subscriptionIconSrc != null && subscriptionIconSrc.startsWith( prefix: "localDrawable")) {
    holder.iconView.setDefaultImageResId(subscriptionIconSrc.substring( startIndex: 16).toInt())
} else if (subscriptionIconSrc != null) {
    holder.iconView.setImageUrl(subscriptionIconSrc, imgLoader)
} else {
    holder.iconView.setDefaultImageResId(R.drawable.ic_crop_original_grey_24dp)
}
```

KUVA 17. NetworkImageView ja paikalliset kuvat

8.2 RxJava ja RxKotlin

RxJava on kirjasto, jonka avulla voi tehdä asynkronisia ja tapahtumiin perustuvia ohjelmia. (RxJava Contributors 2017) Asynkronisella ja tapahtumaan perustuvalla tarkoitetaan sitä, että kun tehdään jotain, esimerkiksi päivitetään tietokantaa, jossa saattaa kestää eikä valmistumisen ajankohtaa voida tietää. Tällöin ei kuitenkaan haluta, että koodi jaa odottamaan tämän kyselyn valmistumista, joka aiheuttaisi esimerkiksi käyttöliittymän jäätymisen. Kun kysely valmistuu, siitä syntyy tapahtuma, jonka perusteella toimitaan. Tämä on RxJavan peruskäsite. Käyttämällä tapahtumapohjaista tapaa, voidaan jättää pois esimerkiksi sotkuiset ja helposti sekaisin menevät *callback*-metodit.

RxKotlin on tehty helpottamaan RxJavan käyttöä Kotlinilla. RxJava toimii kyllä sellaisenaan myös Kotlinilla, mutta RxKotlinissa on Kotlinille erityisiä ominaisuuksia, jotka tekevät käytöstä vielä helpompaa. Tässä sovelluksessa RxJavaa on käytetty esimerkiksi tietokantakyselyissä. Kun toistuvat maksut haetaan kannasta, palautetaan RxJavan *Flowable*-tyyppinen olio. Flowable saattaa sisältää tässä tapauksessa monia toistuvia maksuja listassa ja se saattaa päivittyä ajan myötä. Tätä niin kutsuttua streamia ja sen muutoksia voidaan sitten kuunnella.

```
@Query( value: "SELECT * FROM subscriptions s JOIN providers p ON p.pid = s.providerId")
fun getAll(): Flowable<List<SubscriptionWithProvider>>
```

KUVA 18. Flowable-olion palautus

```
fun getSubscriptionsFromDb() {
    mDb?.subscriptionDao()?.getAll()?.subscribeOn(Schedulers.io())
            ?.observeOn(AndroidSchedulers.mainThread())
            ?.subscribe {
                listOfSubscriptions → bindSubscriptions(listOfSubscriptions)
            }
}
```

KUVA 19. Flowable-streamin kuuntelu

RxJava helpottaa Android-kehittämisen ongelmaa, jossa esimerkiksi tietokantakyselyt täytyy toteuttaa omassa säikeessä. RxJavassa on mahdollista käyttää niin kutsuttuja vuorottajia (scheduler),

jotka voivat päättää missä säikeessä koodia suoritetaan. Toistuvien maksujen haussa (KUVA 19.) on käytetty *Schedulers.io()*-vuorottajaa. Tämä tarkoittaa, että kehittäjän ei tarvitse itse luoda uutta säiettä missä tietokantakyselyn vastaus hoidetaan.

8.3 Room ORM

8.3.1 ORM

Kirjainyhdistelmä ORM tulee sanoista *Object Relational Mapping*, jolla tarkoitetaan ohjelmointitekniikkaa joka yhteensovittaa olio- ja relaatiomalleja tiedon tallentamisessa. Sen tarkoituksena on helpottaa kehittämistä poistamalla SQL-kielen kirjoittamisen ja ymmärtämisen tarpeen. Sen sijaan se laittaa kehittäjän käyttämään oliopohjaista kieltä kuten esimerkiksi Javaa.

ORM:ää käytettäessä kehittäjän ei tarvitse miettiä miten tallentaa tietoa ja kuinka muuttaa olioirakenteita SQL-lauseiksi, vaan mitä ja koska tallennetaan. Tällöin ORM huolehtii olioiden tallentamisesta ja SQL-lauseiden muodostamisesta. Esimerkkejä ORM tuotteista ovat Hibernate ja Oracle TopLink. (Christopher Ireland, David Bowers, Michael Newton, and Kevin Waugh 2009)

8.3.2 Room yleisesti

Room on Googlen kirjasto, joka on luotu helpottamaan tietokannan käsittelyä Android-kehityksessä. (Google 2018e) Google ei missään mainitse Roomin olevan kirjaimellisesti ORM, mutta kun sitä tarkastelee huomaa erittäin paljon yhtenäisyyksiä eri ORM:ien kuten Hibernaten kanssa. Kuten muitakin ORM:iä käytettäessä, tarkoitus on, että ei tarvitse itse olla niin paljoa tekemisissä SQLiten kanssa, vaan että Room tekisi mahdollisimman paljon kehittäjän puolesta.

Tyylinä Roomia käytettäessä on, että luodaan luokkia, jotka kuvaavat tietokannassa tallessa olevaa tietoa. Yleensä yksi luokka kuvaa yhtä taulua. Näiden luokkien lisäksi on niin kutsuttuja *Dao*-luokkia, jotka suorittavat lisäyksiä tietokantaan tai hakuja tietokannasta. Sovellus ei siis tee suoria kyselyitä kantaan, vaan käyttää näitä Dao-luokkia.

```
@Entity(tableName = "subscriptions", foreignKeys = [(ForeignKey(entity = Provider::class,
        parentColumns = ["pid"],
        childColumns = ["providerId"],
        onDelete = ForeignKey.CASCADE))])
open class Subscription(@PrimaryKey(autoGenerate = true) var id: Long?,
```

KUVA 20. Subscription-luokan Room kuvauksia

8.3.3 Room ja relaatiot

Roomia käyttäessä relaatiot täytyy ottaa huomioon omalla tavallaan. Käytännössä *foreign key*-määrittelyjen lisäksi täytyy lisätä *@Embedded*-annotaatiot. Embedded-annotaatio tarkoittaa, että luokalla voi olla relaatiokenttiä ja Room osaa käyttää niitä. Eli kun kysely palauttaa kaikki tarvittavat tiedot

niin Subscriptionille kuin Providerille, joka on varustettu Embedded-annotaatiolla, osaa Room yhdistää tiedot ja palauttaa oikeanlaisen olion.

Tässä sovelluksessa on luotu erikseen oma SubscriptionWithProvider-luokka, jossa Subscription on valmiiksi yhdistettynä Provideriin. Tämä johtuu siitä, että aina ei haluta palauttaa tietoa Providerista, jolloin pelkkä Subscription-luokka riittää.

```
class SubscriptionWithProvider {
    @Embedded lateinit var subscription: Subscription
    @Embedded lateinit var provider: Provider
}
```

KUVA 21. Embedded-annotaatio

8.3.4 Room ja tyypinmuunnokset

Joskus halutaan tallentaa tietokantaan tietoa, jota Room ei osaa automaattisesti tallentaa. Tämmöisiä ovat esimerkiksi päivämäärät tai opinnäytetyössä käytetty oma luokka Money, joka kuvastaa rahaa. Tällöin täytyy tehdä niin kutsuttu *type converter*. Type converterille kerrotaan miksi tyypiksi jokin tietty tyyppi täytyy muuntaa.

Tyypinmuutoksien käyttöä varten täytyy ne ottaa käyttöön Room-tietokannan määritysluokassa. Tämä tapahtuu antamalla määritysluokalle annotaatio *@TypeConverters(Luokannimi::class)*, joka kertoo Roomille mistä luokasta tyypinmuutokset löytyvät.

```
class RoomConverters {
    @TypeConverter
    fun fromInt(value: Int): Money {
        return Money(value, Currency.EURO)
    }

    @TypeConverter
    fun moneyToInt(value: Money): Int {
        return value.amount
    }

    @TypeConverter
    fun fromTimestamp(value: Long?): Date? {
        return if (value == null) null else Date(value)
    }
```

KUVA 22. Tyyppimuunnokset

```
@Database(entities = [Provider::class, Subscription::class], version = 1)
@TypeConverters(RoomConverters::class)
abstract class SubscriptionsDatabase : RoomDatabase() {
```

KUVA 23. Tyyppimuunnoksien käyttöönotto

8.4 Kotlin Android kehityksessä

Kotlin vähentää koodirivien määrää monessa paikassa verrattuna Javaan ja samalla tekee koodista helpompilukuista. Monesti koodirivien vähentyminen johtuu siitä, että Kotlinissa ei tarvitse kirjoittaa niin sanottua ”boilerplate”-koodia, joka ei muutu, vaikka sitä kirjoitetaan eri paikoissa.

Yksi tapaus, jossa vähempi rivien määrä näkyy, on monen if-lauseen toteutus. Esimerkkinä erään RecyclerView-komponentin adapterin metodi, jossa metodin täytyy palauttaa tietty vakioarvo riippuen sijainnista. Javalla olisi tarvinnut tehdä if – else rakenne ja palauttaa vakioarvo eri if-lauseen osista. Kotlinilla tallaisessa tilanteessa käytetään *when*-lausetta ja palautus kirjoitetaan vain kerran. Alla kuvat eri toteutuksista.

```java
public int getItemViewType(int position) {
    if (position > 1) {
        return SUBSCRIPTION_TYPE;
    } else if (position == 1) {
        return GLANCEVIEW_TYPE;
    } else {
        return BANK_ACCOUNT_REMINDER_TYPE;
    }
}
```

KUVA 24. Java toteutus getItemViewType metodista

```kotlin
override fun getItemViewType(position: Int): Int {
    return when {
        position > 1 -> SUBSCRIPTION_TYPE
        position == 1 -> GLANCEVIEW_TYPE
        else -> BANK_ACCOUNT_REMINDER_TYPE
    }
}
```

KUVA 25. Kotlin toteutus getItemViewType metodista

Toinen esimerkki Kotlinin yksinkertaisuudesta on listan järjestäminen, joka onnistuu yhdellä rivillä. Vaikka järjestämiseen haluaisi käyttää omaa vertausfunktiota, voi sen antaa lambda-funktiona.

```kotlin
val sortedSubscriptions = subscriptionsMut
        .sortedWith(compareBy({ it.subscription.getNextPaymentDate() }))
```

KUVA 26. Listan järjestäminen

Viimeisenä esimerkkinä rivien vähentymisestä on tilanne, jossa täytyy käydä lista läpi ja lisätä yhteen listassa olevien tilauksien hinnat. Esimerkissä (KUVA 27.) näkyy myös yksi omista murheenaiheista Kotlinissa, eli niin kutsutun *ternary*-operaattorin puuttuminen. Kotlinissa *ternary*-operaattoria ei ole, vaan joudutaan aina kirjoittamaan normaali if-lause.

```kotlin
return listCopy.map { it: SubscriptionWithProvider

    if (it.subscription.isDueThisMonth()) {
        ^map it.subscription.price
    } else {
        ^map Money( amount: 0, Currency.EURO)
    }

}.fold(Money( amount: 0, Currency.EURO), { total, next -> total + next })
```

KUVA 27. Listan mappaus

Yksi Kotlinin erikoisuuksista Javaan verrattuna on *operator overloading*. Sillä tarkoitetaan esimerkiksi plus-operaattorin ylikirjoittamista tarkoittamaan eri asiaa eri parametreillä. Sovelluksessa sitä on käytetty sovelluksen sisäisen *Money*-luokan pluslaskujen suorittamiseen. *Money*-luokalla pidetään muistissa ja suoritetaan laskuja, jotka liittyvät rahankäsittelyyn. Rahankäsittelyssä ei voida käyttää normaaleja float-arvoja, koska float-arvoilla laskeminen aiheuttaa pyöristysvirheitä.

```
operator fun plus(b: Money): Money {
    return Money( amount: amount + b.amount, currency)
}
```

KUVA 28. Operator overloading

9 MUU TOTEUTUS

REST-rajapinnassa tekniikalla ei ollut niin väliä ja päädyin C#:iin ja ASP.NET Core 2:een siitä syystä, että olin opinnäytetyön aloittamisen aikaan paljon tekemissä kyseisten tekniikoiden kanssa. Lisäksi ASP.NET Core 2 tarjoaa modernin tavan luoda REST-rajapintoja. Mikropalvelut on luotu myös C#:lla ja ASP.NET Core 2:lla, sillä ne toimivat yhdessä REST-rajapinnan kanssa ja näin ollen oli loogista niputtaa ne käyttämään samoja tekniikoita.

SQL Server valikoitui tietokannaksi hyvin pitkälti samoista syistä kuin backend-tekniikat, sillä olin opinnäytetyön aloittamisen aikaan paljon tekemisissä kyseisen teknologian kanssa ja olin vakuuttunut sen käytettävyydestä.

9.1 Backend

Sovelluksen käyttämä REST-rajapinta ja mikropalvelut on molemmat tehty käyttäen ASP.NET Core 2 ohjelmistokehystä C#:lla. Kaikki ne ovat käytännössä Web API tyyppisiä projekteja, koska ne keskustelevat keskenään HTTP:n avulla. Lisäksi projektit on luotu lisäämällä Docker-tuki heti luontivaiheessa, jotta projektista saadaan Docker-kontti helposti.

Koska projektit ovat Web API tyyppisiä, on HTTP-pyynnöt helppo käsitellä ja jakaa eri alueisiin ns. kontrollereiden ja reittien avulla. Kontrollereissa kerrotaan mikä reitti tarkoittaa, että kyseistä kontrolleria halutaan käyttää ja kontrollerin eri metodeille kerrotaan kutsun tyyppi. REST-rajapinnoissa yleensä GET-kutsulla halutaan pyytää resursseja, POST-kutsulla halutaan luoda uusi resurssi, PUT-kutsulla päivitetään ja DELETE-kutsulla poistetaan resurssi.

```
[Produces("application/json")]
[Route("api/NordeaTransactions")]
public class NordeaTransactionsController : Controller
{
```

KUVA 29. Reitin määrittäminen kontrollerille

```
[HttpGet("{account}")]
public async Task<JsonResult> GetSubscriptionsForAccount(string account)
{
```

KUVA 30. Kutsun tyypin (GET) määrittäminen kontrollerin metodille

9.1.1 Mikropalvelut

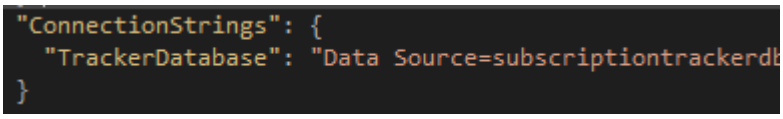
Accounts-mikropalvelussa on luotu omat luokat eri pankkien PSD2-rajapintojen toteutuksille, sillä rajapinnat eivät toimi täysin samalla tavalla eivätkä palauta samanlaista tietoa. Luokilla on tiedot rajapintojen päätteistä, ne osaavat hakea rajapintojen tunnukset ympäristömuuttujista sekä tietävät missä muodossa tieto tulee rajapinnasta. Näin ollen ne osaavat myös muuttaa rajapinnasta tulevan JSON:n C# olioiksi käyttäen Newtonsoft.Json -nimistä kirjastoa.

Esimerkiksi mikropalvelun toiminnasta voidaan ottaa vaikkapa accounts-mikropalvelun toiminta käyttäjän tilitapahtumia haettaessa. Kun pyyntö REST-rajapinnasta tulee mikropalveluun, reitittyy se oikeaan kontrolleriin ja metodiin osoitteen mukaan. Sen jälkeen kyseisen pankin API-luokka tekee kyselyn pankin rajapintaan. Tilitapahtumia haettaessa tapahtumia voi olla tuhansia ja tästä syystä tapahtumat ovat sivutettuna. Kaikki sovelluksessa käytettyjen pankkien rajapinnat vastaavat JSON-muodossa. Sivutetusta JSON:sta sitten luetaan tapahtumat C# olioiksi ja lisätään listaan. Jos JSON:ssa oli linkki seuraavaan sivuun, muokataan linkkiä ja haetaan seuraava sivu tapahtumia. Samalla pidetään koko ajan yllä listaa tapahtumista. Kun linkkiä seuraavaan sivuun ei enää ole, on kaikki tapahtumat saatu haettua. Tämän jälkeen täytyy tarkistaa listassa olevat tilitapahtumat ja verrata niitä tietokannassa oleviin tunnettuihin maksunsaajiin. Mikropalvelu ottaa viisitoista tapahtumaa kerrallaan ja kutsuu tietokannassa olevaa funktiota, joka palauttaa jokaista kohtaan tiedon onko tapahtuman maksunsaaja tunnettu vai ei. Kun tapahtumat on saatu käytyä läpi, palauttaa mikropalvelu vastauksen JSON muodossa REST-rajapinnalle, joka taas palauttaa tiedon Android-sovellukselle.

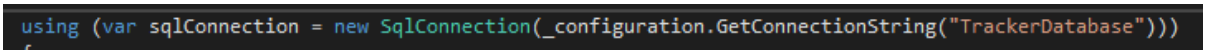
Kaikki edellä mainitut toimenpiteet olisi voinut suorittaa suoraan REST-rajapinnasta ilman mikropalvelujen tekemistä, mutta kuten esimerkistä huomataan, se käyttää paljon resursseja ja sen valmistumisessa saattaa kestää kauan. Tästä syystä se on jaettu omaksi mikropalvelukseen ja AWS voi skaalata sitä automaattisesti, mikäli se huomaa resurssien olevan vähissä.

9.1.2 Tietokantayhteydet

Jokaisessa backendin osassa, niin mikropalveluissa kuin REST-rajapinnassakin on yhteys tietokantaan. Ne hakevat käyttäjätunnukset konfiguraatiotiedostosta ja käyttävät normaalia SqlClient:iä.



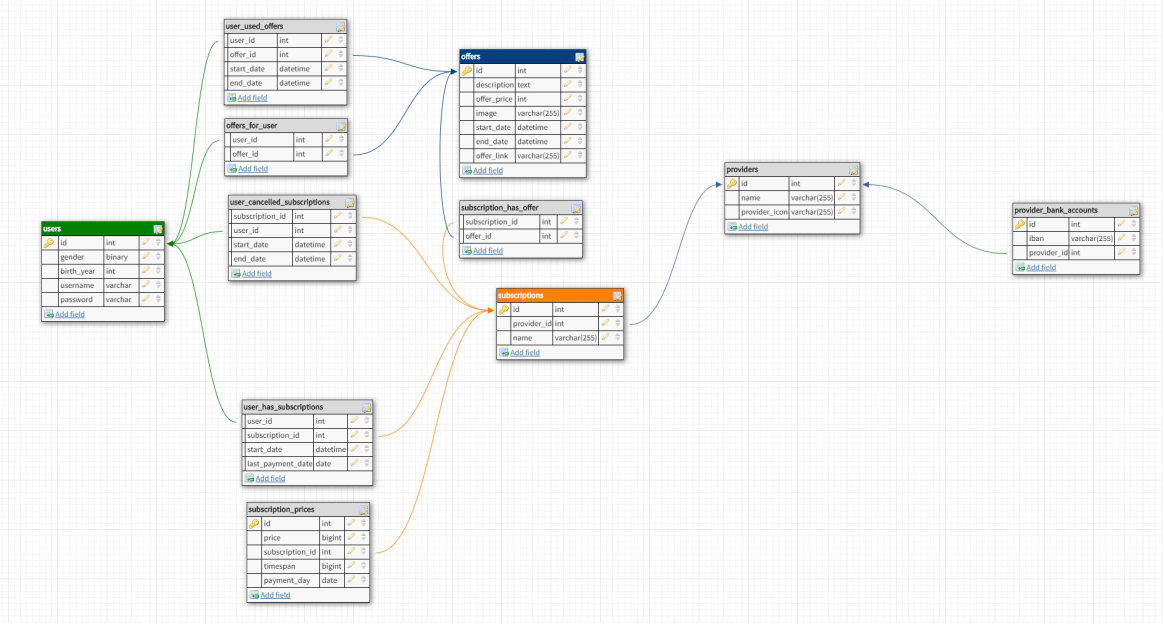
KUVA 31. Tietokantayhteyden konfiguraatio



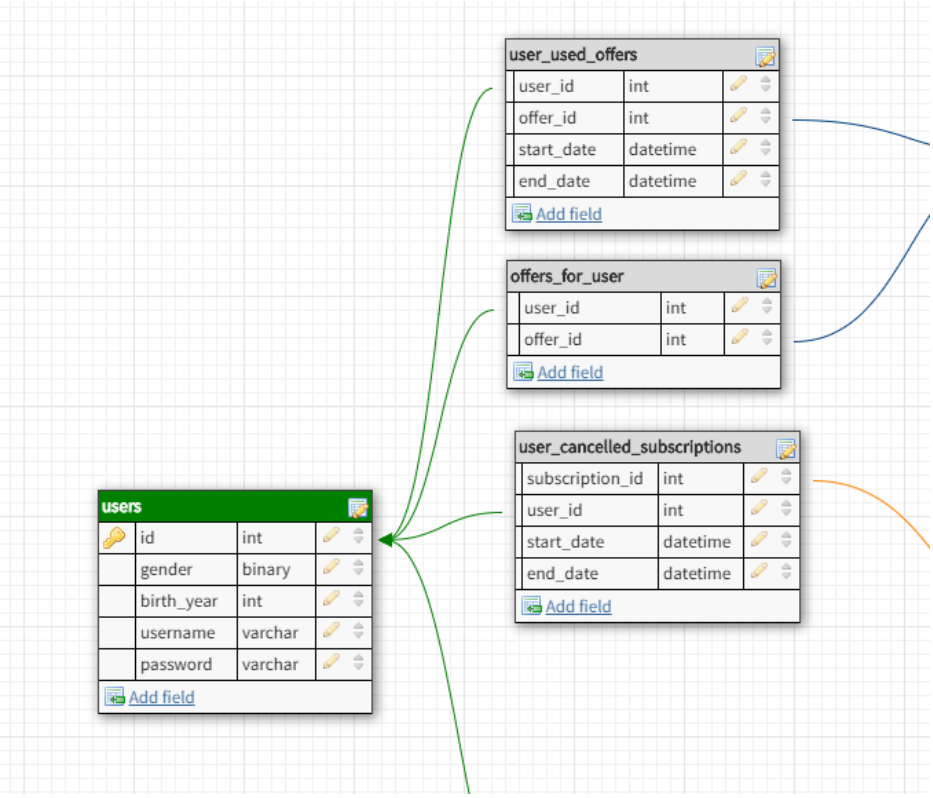
KUVA 32. Tietokantayhteyden käyttäminen

9.2 SQL Server

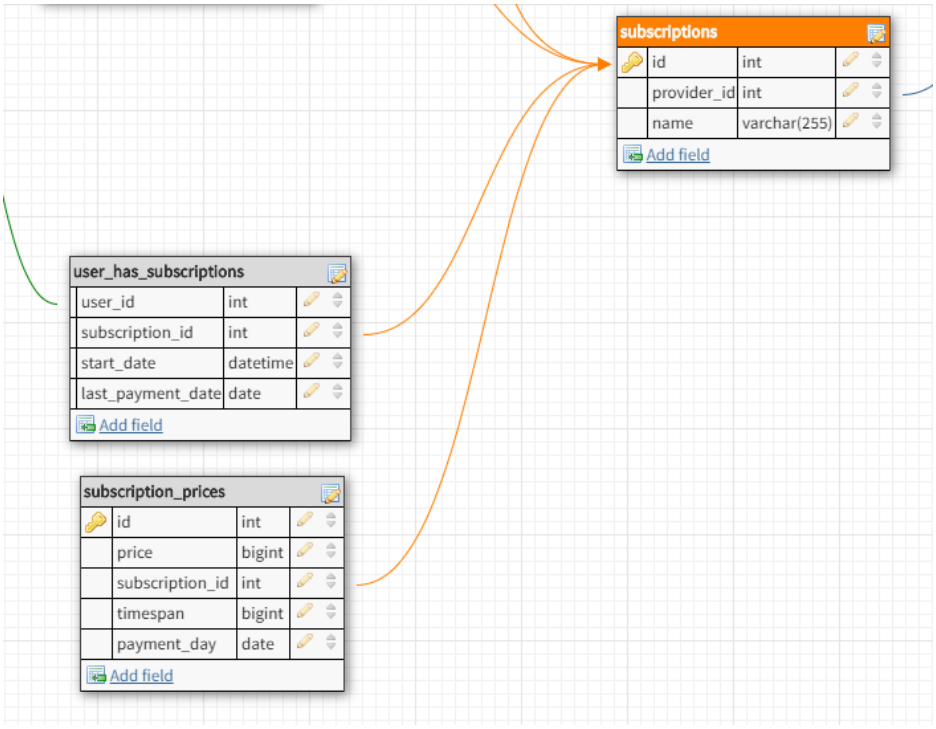
Sovelluksen tietokanta on toteutettu käyttäen Microsoftin SQL Server Express -tietokantaa. Tietokanta sijaitsee RDS -nimisessä pilvipalvelussa, josta se saa yhden julkisen IP-osoitteen. Tietokannalle on määritelty mistä ulkoisista IP-osoitteista siihen voi ottaa yhteyden.



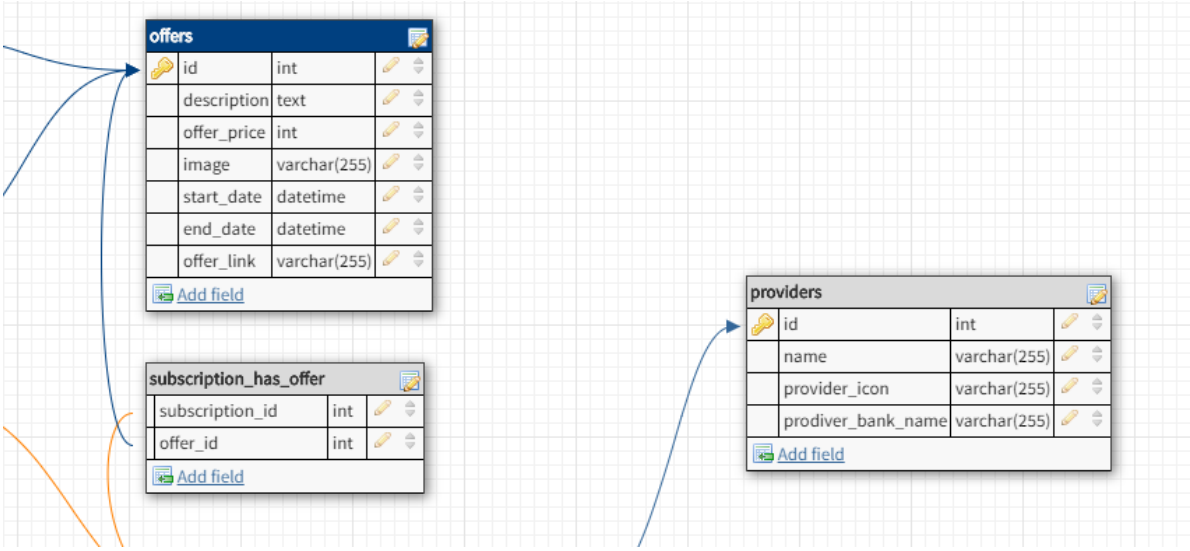
KUVA 33. Kokonaiskuva tietokannan skeemasta



KUVA 34. Tietokannan skeema osa 1.



KUVA 35. Tietokannan skeema osa 2.



KUVA 36. Tietokannan skeema osa 3.

Tietokannassa käyttäjään liittyy vain monen suhde moneen välitauluja. Tämä johtuu siitä, että käyttäjällä voi olla monia tilauksia, monia käytettyjä tarjouksia tai monia yksinomaan hänelle kohdistettuja tarjouksia.

Kuukausimaksujen tarjoajien taulussa on myös tieto heidän maksunsaajanimestään, tällä hetkellä demosovellusta varten on tuettuna vain yksi maksunsaaja / tarjoaja, mutta tulevaisuudessa tarkoituksena on lisätä mahdollisuus useille maksunsaajanimille. Maksunsaajanimi on indeksoitu nopeuttaakseen oikean tarjoajan etsimistä.

Tietokannassa on tallennettu taulun palauttava funktio maksunsaajien etsimiseksi. Tämä mahdollistaa sen, että jokaista maksutapahtumaa ei tarvitse kysyä erikseen kannalta, vaan sille lähetetään x määrä (tällä hetkellä 15) tapahtumaa, ja funktio palauttaa tiedon onko maksunsaaja tunnettu.

```
ALTER FUNCTION [dbo].[CheckForKnownBankAccounts]
(
    @account_1 varchar(255),
    @account_2 varchar(255),
    @account_3 varchar(255),
    @account_4 varchar(255),
    @account_5 varchar(255),
    @account_6 varchar(255),
    @account_7 varchar(255),
    @account_8 varchar(255),
    @account_9 varchar(255),
    @account_10 varchar(255),
    @account_11 varchar(255),
    @account_12 varchar(255),
    @account_13 varchar(255),
    @account_14 varchar(255),
    @account_15 varchar(255)
)
RETURNS @rtnTable TABLE
(
    account_name varchar(255),
    is_subscription int
)
AS
BEGIN
    DECLARE @account VARCHAR(255)

    IF (@account_1 IS NOT NULL)
        BEGIN
            SELECT @account = provider_bank_name FROM dbo.providers WHERE provider_bank_name LIKE @account_1;

            IF (@account IS NOT NULL)
                BEGIN
                    INSERT INTO @rtnTable(account_name, is_subscription) VALUES (@account, 1);
                    SET @account = NULL
                END
        END
```

KUVA 37. Maksunsaajan tunnistamisfunktio

Yllä olevassa kuvassa (KUVA 37.) näkyy osa maksunsaajien tunnistamisfunktiosta. Funktio tarkistaa jokaista parametriä kohtaan, löytyykö kyseisellä maksunsaajan nimellä kannasta palveluntarjoaja ja palauttaa tiedon kutsujalle.

10 YHTEENVETO

Ennen opinnäytetyötä olin tehnyt jonkin verran Android-kehitystä, mutta en ollut koskenutkaan Kotliniin. Opinnäytetyötä tehdessä Kotlin tuli kuitenkin kielenä hyvin tutuksi ja jopa tykästyin siihen. Loppuarvioksi Kotlinin käytettävyydestä Android-kehityksessä voisin sanoa, että oppimiskäyrä on melko matala ja kehitys on välillä jopa nopeampaa kuin Javalla. Olin myös pitkään halunnut olla tekemisissä AWS:n kanssa ja opinnäytetyö tarjosi hyvän pintaraapaisun AWS:n kanssa työskentelyyn.

Sovelluksessa jäi paljon jatkokehitettävää, vaikka kyseessä onkin vain MVP demo oikeasta sovelluksesta. Esimerkiksi tarjousten käsittely jäi hyvin vähäiseksi eikä niistä saatu tehtyä käyttöliittymää. Tämä johtui suureksi osaksi siitä, että ei ollut vielä tekovaiheessa tietoa siitä minkälaisia tarjoukset ovat.

Toinen tulevaisuuden jatkokehitys kohde on palveluntarjoajien monen maksunsaajanimen mahdollistaminen. Tällä hetkellä sovelluksessa on mahdollisuus vain yhteen maksunsaajanimeen per tarjoaja, koska liikeidean esittelyssä ei ole tarvetta monelle maksunsaajalle.

Opinnäytetyössä tehdyn sovelluksen on tarkoitus toimia pohjana varsinaiselle yritysidean sovellukselle ja sovellusta tullaan jatkokehittämään myös siltä pohjalta. Tässä vaiheessa jatkokehityskohteita ovat esimerkiksi parempi oman talouden hallinta ja kuvaajat kuluista ja maksuista.

LÄHTEET JA TUOTETUT AINEISTOT

AMAZON WEB SERVICES 2018a. [Viitattu 2018-04-19] Saatavissa: https://aws.amazon.com/what-is-aws/

AMAZON WEB SERVICES 2018b. [Viitattu 2018-04-19] Saatavissa: https://aws.amazon.com/pricing/

AMAZON WEB SERVICES 2018c. [Viitattu 2018-06-04] Saatavissa: https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_WorkingWithSecurityGroups.html

AMAZON WEB SERVICES 2018d. [Viitattu 2018-04-25] Saatavissa: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html

ANDROID.COM 2018. [Viitattu 2018-04-19] Saatavissa: https://source.android.com/

AVRAM Abel 2013. [Viitattu 2018-06-04] Saatavissa: https://www.infoq.com/news/2013/03/Docker

BERNHEIM Laura 2017. Docker's Tools of Mass Innovation: Explosive Growth From Open-Source Containers to Commercial Platform for Modernizing and Managing Apps. [Viitattu 2018-06-04] Saatavissa: https://www.hostingadvice.com/blog/dockers-explosive-growth-from-open-source-containers-to-commercial-platform/

BOBBY 2018. [Viitattu 2018-04-10] Saatavissa: http://www.bobbyapp.co/

DOCKER 2018. What is a container. [Viitattu 2018-04-19] Saatavissa: https://www.docker.com/what-container

DUCROHET Xavier, NORBYE Tor ja CHOU Katherine 2013. Android Studio: An IDE built for Android. [Viitattu 2018-04-19] Saatavissa: https://android-developers.googleblog.com/2013/05/android-studio-ide-built-for-android.html

FINANSSIALA RY, 2018. Kysymyksiä ja vastauksia toisesta maksupalveludirektiivistä (PSD2) [Viitattu 2018-06-04] Saatavissa: http://www.finanssiala.fi/uutismajakka/Sivut/QA-Toinen-maksupalveludirektiivi.aspx

FINANSSIVALVONTA, 2018a. Uusi maksupalveludirektiivi – Payment Services Directive, PSD2. [Viitattu 2018-04-19] Saatavissa: http://www.finanssivalvonta.fi/fi/Saantely/Saantelyhankkeet/PSD2/Pages/Default.aspx

FINANSSIVALVONTA 2018b. [Viitattu 2018-05-10] Saatavissa: http://www.finanssivalvonta.fi/en/Regulation/Interpretations/Pages/01_2018.aspx

FUSEBILL 2018. Billing Frequency is a Balancing Act. [Viitattu 2018-04-10] Saatavissa: https://blog.fusebill.com/what-is-the-best-billing-frequency-for-subscription-business

GOOGLE 2018a. Meet Android Studio. [Viitattu 2018-04-19] Saatavissa: https://developer.android.com/studio/intro/

GOOGLE 2018b. [Viitattu 2018-04-25] Saatavissa: https://material.io/design/introduction/#principles

GOOGLE 2018c. [Viitattu 2018-04-25] Saatavissa: https://developer.android.com/guide/components/fragments

GOOGLE 2018d. [Viitattu 2018-05-04] Saatavissa: https://developer.android.com/reference/android/support/v7/widget/RecyclerView

GOOGLE 2018e. [Viitattu 2018-05-10] Saatavissa: https://developer.android.com/topic/libraries/architecture/room

GOOGLE PLAY 2018a. [Viitattu 2018-04-10] Saatavissa: https://play.google.com/store/apps/details?id=com.slapp.subby

GOOGLE PLAY 2018b. [Viitattu 2018-04-10] Saatavissa: https://play.google.com/store/apps/details?id=vmax.billy

HIATUS 2018. [Viitattu 2018-04-10] Saatavissa: http://www.hiatusapp.com/

IRELAND Christopher, BOWERS David, NEWTON Michael ja WAUGH Kevin 2009. Understanding Object-Relational Mapping: A Framework Based Approach, 202 - 203 [Viitattu 2018-06-04] Saatavissa: https://pdfs.semanticscholar.org/e1da/0d17ed4acd847801fda7e0b5a4dc77b43df6.pdf

KOTLINLANG 2018. [Viitattu 2018-04-19] Saatavissa: https://kotlinlang.org/docs/reference/faq.html

MICROSOFT 2018a. [Viitattu 2018-04-19] Saatavissa: https://docs.microsoft.com/fi-fi/visualstudio/#pivot=languages&panel=languages1

MICROSOFT 2018b. [Viitattu 2018-06-04] Saatavissa: https://www.visualstudio.com/downloads/

MICROSOFT 2018c. [Viitattu 2018-06-04] Saatavissa: https://docs.microsoft.com/fi-fi/visualstudio/debugger/debugger-feature-tour

MICROSOFT 2018d. [Viitattu 2018-04-19] Saatavissa: https://www.microsoft.com/net/download/windows/build

MICROSOFT 2018e. [Viitattu 2018-04-19] Saatavissa: https://www.microsoft.com/fi-fi/sql-server/sql-server-2017-editions

MONEYWISE 2018. Brits waste £448 million a month on unused subscriptions. [Viitattu 2018-04-10] Saatavissa: https://www.moneywise.co.uk/news/2017-03-29/brits-waste-448-million-month-unused-subscriptions

NORDEA 2018. [Viitattu 2018-05-10] Saatavissa: https://developer.nordeaopenbanking.com/app/documentation?api=Account%20Information%20Services%20API&version=2.1#api_endpoints

PRICEWATERHOUSECOOPERS 2007. [Viitattu 2018-06-03] Saatavissa: https://www.pwc.com/us/en/technology-innovation-center/assets/softwarepricing_x.pdf

RECORDING INDUSTRY ASSOCIATION OF AMERICA 2018. News and Notes on 2017 RIAA Revenue Statistics. [Viitattu 2018-04-10] Saatavissa: https://www.riaa.com/reports/2017-riaa-shipment-revenue-statistics-riaa/

RXJAVA CONTRIBUTORS 2017. [Viitattu 2018-05-10] Saatavissa: https://github.com/ReactiveX/RxJava/wiki

SHAFIROV MAXIM 2017. [Viitattu 2018-04-19] Saatavissa: https://blog.jetbrains.com/kotlin/2017/05/kotlin-on-android-now-official/

STATISTA 2017. Distribution of smartphone operating systems used in the U.S. 2017. [Viitattu 2018-04-19] Saatavissa: https://www.statista.com/statistics/716053/most-popular-smartphone-operating-systems-in-us/

STATISTA 2018a. Digital Media Market Report. [Viitattu 2018-04-10] Saatavissa: https://www.statista.com/outlook/206/100/video-streaming--svod-/worldwide

STATISTA 2018b. Payment frequency of product subscriptions in the United States as of February 2017, by product category. [Viitattu 2018-04-10] Saatavissa: https://www.statista.com/statistics/721376/payment-frequency-of-product-subscriptions-by-product-category/

STEPSTONE TECH 2018. [Viitattu 2018-04-25] GitHub Repository: https://github.com/stepstone-tech/android-material-stepper

TRACKMYSUBS 2018. [Viitattu 2018-04-10] Saatavissa: https://www.trackmysubs.com/