Projektimallin kehittäminen web-sovelluksen muuttamisesta natiivi iOS-sovellukseksi

Timi Voutilainen

2019 Laurea

Laurea-ammattikorkeakoulu

Projektimallin kehittäminen web-sovelluksen muuttamisesta natiivi iOS-sovellukseksi

Timi Voutilainen
Tietojenkäsittely
Opinnäytetyö
Toukokuu, 2019

Laurea-ammattikorkeakoulu **Tiivistelmä**
Tietojenkäsittelyn koulutusohjelma
Tradenomi (AMK)

Timi Voutilainen

Projektimallin kehittäminen web-sovelluksen muuttamisesta natiivi iOS-sovellukseksi

Vuosi 2019 Sivumäärä 33

Opinnäytetyönä kehitettiin projektimalli web-sovelluksen siirtämisestä natiiviksi iOS-sovellukseksi. Työn toimeksiantaja oli Profit Software Oy, joka halusi lisätä ymmärrystään web-sovelluksen ja natiivisovelluksen arkkitehtuurisista eroista luomalla projektimallin prosessia vasten.

Ennen kehitystyötä projektille tehtiin suunnitelma. Suunnitelman luonnin jälkeen projekti toteutettiin käyttäen mukautettua Scrum-mallia.

Työn teoreettinen viitekehys sisältää opinnäytetyöhön liittyviä teknologioita sekä käytettyjä tutkimusmenetelmiä. Teoriaosuudessa kuvataan myös projektinhallintaa sekä projektin kannalta oleellisia ketteriä menetelmiä.

Tietoperustana työssä käytettiin alalta löytyvää kirjallisuutta, teknologioiden kehittäjien dokumentaatioita sekä artikkeleita. Toiminnallisessa osassa tietoperustana käytettiin opinnäytetyöntekijän omaa osaamista sekä teknologioiden dokumentaatiot.

Kehitystyön tuloksena syntyi projektimalli, jonka avulla web-sovellus voidaan siirtää natiivisovellukseksi mahdollisimman ketterästi. Työn tuloksena lisättiin myös iOS-sovelluskehitysosaamista yrityksen sisäisesti.

Asiasanat: Projektimalli, React, iOS

Laurea University of Applied Sciences **Abstract**
Degree Programme in Business Information Technology
Bechelor's Thesis

Timi Voutilainen

Developing a project model for transferring web application to native iOS application

Year 2019 Pages 33

The objective of this bachelor's thesis was to develop a project model for transferring a web application to a native iOS application. The thesis was initiated by Profit Software Oy which wanted to increase their knowledge about the architectural differences of a web application and a native application by creating a project model.

In the thesis a plan was created before the development work. After the initial plan the project was implemented with custom Scrum model.

The theoretical part of the thesis contained the relevant technologies and used research methods. The theoretical part included project management and relevant agile methods as well. Information in the theoretical part was based on electronic sources and documenting technologies. The functional part was based on the author's own knowledge of Swift and iOS ecosystem and documentations on technologies.

The output of the thesis was a project model which can be used to transfer a web application to a native application as agilely as possible. As a result, the client company also increased its knowledge of iOS.

Keywords: Project model, React, iOS

Sisällys

1 Johdanto

Tässä toiminnallisessa opinnäytetyössä kehitettiin Profit Software Oy:lle projektimalli web-sovelluksen siirtämisestä natiivi iOS-sovellukseksi. Projektimallin ohella tuotettiin natiivi iOS-sovellus valmiina olevasta web-sovelluksesta, jotta projektimallia voitaisiin kokeilla ja kehittää ketterämmäksi.

Profit Softwaren pääasiallinen web-teknologia on React ja tällä tekniikalla tehtyjä sovelluksia on olemassa yli kymmenen. Profit Softwaren tavoitteena on kuitenkin toteuttaa natiiveja iOS- ja Android-sovelluksia. Alentaakseen asiakkaiden natiivien mobiilisovelluksien ostamisen kynnystä, tulisi yrityksen sisäistä osaamista lisätä sekä tutkia kuinka arkkitehtuurit eroavat ja kuinka kehitys toteutetaan ketterästi.

Opinnäytetyö tehtiin talven sekä kevään 2019 aikana. Työssä kuvataan arkkitehtuurisia eroja, projektimallia, projektimallin toteutusta sekä esitellään työssä käytettäviä teknisiä välineitä ja viitekehyksiä, joita ovat iOS ja React sekä niiden rajapinnat. Projektimallin toteutuksessa kuvataan kehitystyötä ja siihen liittyvää dokumentointia yleisellä tasolla sekä vastaan tulleita haasteita.

Teoriaosuudessa kuvataan tutkimusmenetelmiä, toimintatutkimusta ja tapaustutkimusta. Projektinhallinnasta kuvataan ketteriä menetelmiä, joista käydään tarkemmin läpi Scrumia. Web-sovellusta ja natiivisovellusta kuvataan yleisellä tasolla sekä sovelluksissa käytettyjä teknologioita.

Työssä käytetty web-sovellus oli yrityksen sisäisesti rakennettu soveltuvuusselvitys-sovellus, joka oli alun perin rakennettu asiakkaiden tarpeista. Opinnäytetyön tekijä oli mukana tekemässä web-sovellusta ja se oli tekijälle tuttu, mutta yleiseen arkkitehtuuriin tekijä ei voinut vaikuttaa.

2 Toimeksiantaja ja työn tausta

Työn toimeksiantaja on Profit Software Oy. Opinnäytetyön tekijä työskentelee yrityksessä it-konsulttina. Profit Software on perustettu vuonna 1992 ja se on erikoistunut finanssialan ohjelmistoratkaisuihin ja tehtävään konsultointiin vakuutusyhtiöille ja pankeille. (Profit Software 2019.)

Opinnäytetyön tekijällä ei ollut aikaisempaa kokemusta natiivi iOS-sovelluskehityksestä ja tarkoituksena oli lisätä yrityksen sisäisesti osaamista. Yrityksellä oli myös halua saada projektimalli, kuinka web-sovellus muutetaan natiiviksi iOS-sovellukseksi.

Työssä käytettävä web-sovellus oli yrityksen sisäisesti rakennettu soveltuvuusselvitys, joka rakennettiin asiakkaiden mahdollisista tarpeista. Sovelluksen tarkoituksena oli, että siitä

tehtäisiin sekä web- että mobiilisovellukset, jotta sovellusta pystyttäisiin käyttämään millä tahansa laitteella.

2.1 Aihe ja tavoite

Työssä laadittiin projektimalli web-sovelluksen muuttamisesta iOS-natiivisovellukseksi, jolla saataisiin mahdollisimman tehokkaasti käytettyä olemassa olevaa koodia ja arkkitehtuuria hyödyksi. Projektimallin tulisi olla sellainen, että sitä voitaisiin käyttää ja tarjota asiakkaille.

Yrityksen sisällä ei ole tällä hetkellä natiivia iOS-osaamista, mutta Android-osaamista löytyi, joten oli luonnollista, että projektimalli tehtäisiin iOS:lle, jotta samalla lisättäisiin yrityksen iOS-osaamista. Arkkitehtuurit ovat melko samankaltaisia sekä iOS-ympäristössä että Android-ympäristössä, mikä mahdollistaa sen, että niitä voidaan helposti soveltaa käyttämään toista alustaa.

Tärkeänä osana projektimallissa on se, että voidaan määrittää, miten tuotetaan arkkitehtuurisesti hyvä web-sovellus, jotta se olisi mahdollisimman helposti siirrettävissä täysin toiseen arkkitehtuuriin. Tulevaisuudessa voi olla, että tulee uusi käytettävä web-sovelluskirjasto ja on oleellista, että opinnäytetyön tekijä osaa miettiä arkkitehtuuria yleisenä kokonaisuutena eikä pelkästään kyseisen web-sovelluskirjaston kannalta.

2.2 Rajaus

Opinnäytetyön ulkopuolelle jätetään projektimallin aikana toteutettu ohjelmointityö ja sen dokumentointi. Tarkempaa työn aikana toteutunutta projektinhallintaa ei käydä läpi, koska se saattaisi kasvattaa opinnäytetyön kokoa.

Työn dokumentoinnin kannalta esitellään arkkitehtuurisia ja iOS-sovelluskehitykseen liittyviä kuvioita yleisellä tasolla. Toiminallisessa osassa esitellään esimerkkejä toteutuksessa käytettävistä Swift ohjelmointikoodista. Projektimallia ei käydä läpi mahdollisen asiakkaan näkökulmasta, koska se saattaisi kasvattaa opinnäytetyön kokoa. Kaikki koodiesimerkit, joita opinnäytetyössä on käytetty, ovat yleisluontoisia eivätkä sisällä arkaluontoisia asioita. Kooditasoisia yksityiskohtia ei voida esittää, koska Profit Software Oy omistaa lähdekoodin.

2.3 Käsitteet

Android: Android on Googlen kehittämä käyttöjärjestelmä, joka on suunniteltu ensisijaisesti kosketusnäyttö puhelimille ja tableteille.

JavaScript: JavaScript on ohjelmointikieli ja sen pääasiallinen käyttö tapahtuu selaimessa, jotta web-sivuille on mahdollista lisätä toiminnallisuutta.

JSON: JSON (JavaScript Object Notation) on tiedostomalli, jota käytetään datan tallentamiseen ja lähettämiseen. Siinä data sijaitsee objektissa, jossa on nimi – arvo pareja.

Ohjelmointirajapinta: Ohjelmointirajapinta määritellee, miten ohjelmat voivat tehdä pyyntöjä keskenään. Rajapinta voi olla pelkkä datarajapinta, jonka kautta voidaan saada toisen järjestelmän dataa, tai toiminnallinen rajapinta, joka tarjoaa valmiiksi olevia toiminnallisuuksia.

REST: REST (Representational State Transfer) on sovellusarkkitehtuurimalli ohjelmointirajapintojen kehittämiseen. REST-rajapintaan tehdään kutsu ja vastauksena saadaan dataa esimerkiksi JSON-muodossa.

Sovelluskirjasto: Sovelluskirjasto yleisesti sisältää valmiiksi kirjoitettua koodia, jonka avulla sovelluskehittäjä voi lisätä toiminnallisuutta tai automatisoida prosessin kirjoittamatta koodia.

3 Projektinhallinta

Projektinhallinnalla tarkoitetaan resurssien suunnittelua ja hallintaa, jotta projektissa saavutetaan vaatimukset aikataulujen puitteissa. Päärajoitteet projekteilla ovat vaatimukset, aika, laatu ja budjetti. Projektihallinnan tarkoituksena on tuottaa asiakkaalle valmis projekti asiakkaan tavoitteiden mukaan. (Kerzner 2017.)

Projektinhallintaa voidaan käyttää missä tahansa projekteissa, mutta yleensä sitä muokataan tarkoin alojen vaatimuksien mukaan. It-alalle on muodostunut oman tyylinen projektinhallinta, jossa erikoistutaan projektin eri elinkaariin, kuten määrittelyyn, suunnitteluun, toteutukseen sekä testaukseen. (Kerzner 2017.)

3.1 Ketterät menetelmät

Ketterät menetelmät minimoivat riskiä kehittämällä sovellusta pienissä sykleissä tai julkaisuvaiheissa toisin kuin vesiputousmallissa, jossa tuotetaan kokonainen sovellus kerralla. Kun sovellusta tuotetaan pienissä osissa tai sykleissä, voidaan vaikeuttaviin tai helpottaviin asioihin reagoida välittömästi, minkä ansiosta projektin aikataulua on helpompi pitää silmällä. Sovelluksen muuttuviin vaatimuksiin voidaan puuttua kesken kehityksen, mikä on yleistä sovelluskehitys projekteissa. Ketterät menetelmät sopivat hyvin projekteihin, joissa määritelmät ovat epäselviä tai nopeasti muuttuvia. (Dawson 2009.)

Sovelluskehityksessä on tärkeää reagoida nopeasti mahdollisiin aikataulumuutoksiin, koska myöhästynyttä projektia ei saada aikaisemmin valmiiksi lisäämällä siihen tekijöitä. Tästä on olemassa Brooksin laki, joka on yksinkertaistettuna se, että lisättäessä henkilö projektiin, projekti vie entistä enemmän aikaa. Tähän vaikuttavat muun muassa se, että projektin uudelle jäsenelle joudutaan opettamaan järjestelmää, joka vie mahdollisesti muilta kehittäjiltä

tuottavaa aikaa. Projektin uudella jäsenellä kestää jonkin aikaa tulla tuottavaksi, koska uudet jäsenet voivat alussa tuottaa negatiivista työpanosta projekteihin, jos heidän työnsä sisältää esimerkiksi virheitä, jotka hankaloittavat kehitystä ja joiden selvitys vie aikaa. (Brooks, 1975.)

3.2 Scrum

Scrum on kahdeksan yksilön tiimi rugbyssa, jossa jokainen ryhmittyneenä koittaa viedä palloa eteenpäin. Jokaisella yksilöllä on oma määritelty roolinsa kohdistettuna yhteiseen tavoitteeseen. Scrum nimi on lainattu rugbysta painottaakseen tiimien tärkeyttä monimutkaisessa tuotekehityksessä. Scrum projektinhallinnan viitekehyksenä on tarkoitettu pienille tiimeille. Kehitystiimeissä jokaisen täytyy ymmärtää oma roolinsa. (Verheyen 2014.)

Ennen projektin alkua määritellään pääarkkitehti, joka kehittää sovellusarkkitehtuurin. Tiimin on oltava valmiina muuttamaan arkkitehtuuria projektin edetessä, mutta projektin alussa pääarkkitehti luo projektisuunnitelman, joka toteutetaan arkkitehtuurin perusteella ja pääarkkitehti pitää huolta, että projektin visio perustuu arkkitehtuuriin ja pysyy johdonmukaisena koko projektin ajan. (Rising & Janoff 2000.)

Alustavan suunnitelman jälkeen luodaan lyhyitä kehitysvaiheita, joita kutsutaan sprinteiksi, joiden aikana tuotetta kehitetään osissa. Sprintin yleinen kesto on yhdestä viikosta neljään viikkoon. Lopetusvaihe yleensä lopettaa tuotekehityksen, minkä jälkeen on mahdollista tehdä ylläpito- tai jatkokehitysprojekteja. Tiimi seuraa sprinttien aikana kaikkia tehtäviä, jotka laitetaan ylös backlogille. Backlogi määrittelee tiimin tekemistä. Ennen jokaista sprinttiä tiimi päivittää backlogin ja priorisoi jokaisen tehtävän. Tiimin jäsenet arvioivat tehtävien mahdolliset kestot ja ottavat sprintille mukaan sen verran tehtäviä, joita ajatellaan saatavan valmiiksi. Isompien tehtävien kestoa on vaikeaa määritellä, joten niille täytyy antaa aikaa tarpeeksi. Tehtäviä on mahdollista siirtää seuraavalle sprintille, jos ne jäävät kesken tai sprintille voidaan ottaa mukaan lisää tehtäviä, jotka ovat lisättynä backlogille. Sprintin suunnittelussa pitää ottaa huomioon myös se, että tehtävistä saattaa syntyä uusia tehtäviä sprintin aikana, kuten esimerkiksi tehtävistä syntyvien virheiden myötä. Tehtävien määritelmät voivat myös muuttua kesken sprintin, kun huomataan jotakin, joka ei ollut ennen toteuttamista tullut ilmi. (Rising & Janoff 2000.)

Sprinttien aikana tiimi pitää usein palavereita, yleensä päivittäin, joissa käydään läpi mitä kaikki tiimin jäsenet ovat tehneet ja mitä tehdään seuraavaksi. Näiden palavereiden tarkoituksena on rakentaa ryhmähenkeä tiimin sisällä ja tehdä jokaisen jäsenen työ näkyväksi muille jäsenille. Näiden ansiosta jokaisella tiimin jäsenellä on parempi näkemys sprintin kulusta ja he pystyvät mukauttamaan omaa tekemistään muiden tekemisen perusteella. (Rising & Janoff, 2000.)

Scrum tiimissä on kolme eri roolia, joita ovat Scrum master, tuotteen omistaja ja kehitystiimi. Scrum master on vastuussa palavereista sekä backlogin määrittelystä sprintille. Scrum master pitää huolta, että kehitystiimi edistyy tehtävissään ja kirjaa ylös päätökset, joita tehdään palavereissa. Tuotteen omistajan tehtävänä on määritellä tehtävät, joita sovellukseen tarvitaan ja niiden priorisointi. Tuotteen omistaja on käytännössä vastuussa sovelluksen ominaisuuksien määrittelystä ja siitä, että toteutetut ominaisuudet täyttävät määritellyt vaatimukset. Kehitystiimi tekee tehtäviä, joita Scrum master on määrittänyt jokaiselle sprintille. (Rising & Janoff 2000.)

4 Web-sovellus ja mobiilisovellus

Web-sovelluksia voidaan rakentaa käyttäen useita web-kirjastoja, joista käytetyimpiä ovat Facebookin luoma React-kirjasto, Googlen luoma Angular-kirjasto sekä Vue-kirjasto. Käytännössä kaikilla kirjastoilla on mahdollista luoda samoilla toiminnoilla pyörivä web-sivu, mutta kirjastoissa arkkitehtuurit eroavat toisistaan. Kirjastojen valinta yleensä perustuu kehittäjän omiin taitoihin tai organisaation päätöksiin.

Työssä käytetty web-sovellus oli alkuperäisesti asiakkaan tarpeisiin rakennettu soveltuvuusselvitys. Sovelluksen käyttötarkoitus oli se, että yrityksen työntekijät voivat antaa myynti- sekä rekryvinkkejä. Web-sovellus oli rakennettu käyttämällä React-webkirjastoa sekä TypeScriptiä ohjelmointikielenä. Web-sovellus oli valmiina ennen tämän opinnäytetyön aloittamista. iOS-natiivisovelluksen kehittämisessä on käytetty MacBookia, koska siinä on käytössä Mac OS -käyttöjärjestelmä, Applen Xcode-sovellusta sekä ohjelmointikielenä Swiftiä.

4.1 React

React on Facebookin luoma JavaScript-kirjasto ja se julkaistiin toukokuussa 2013. React on tarkoitettu käyttöliittymien tekemiseen ja yhtenä sen tärkeänä ominaisuutena tähän on pienten erillisten komponenttien uudelleenkäyttämistä rakentaakseen käyttöliittymiänäkymiä. Toisin kuin muissa suosituissa JavaScript-kirjastoissa Reactissa käytetään HTML:n sijaan JSX-syntaksia, joka muuttuu käyttäjän selaimeen HTML:ksi. Monimutkaiset React-sovellukset vaativat ylimääräisiä kirjastoja käsittelemään tilanhallintaa, reititykseen ja rajapintakutsujen tekemiseen. (Facebook 2019.)

React on tällä hetkellä suosituin JavaScript-webkirjasto. Maaliskuussa 2019 Reactia ladattiin noin 6,2 miljoonaa kertaa viikossa, kun taas sen kilpailijoita Vueta 1 miljoonaa kertaa ja Angularia 0,4 miljoonaa kertaa. (Potter 2019.)

4.2 TypeScript

TypeScript on Microsoftin kehittämä ja ylläpitämä ohjelmointikieli. Käytännössä se on JavaScriptiä tyypitettynä. JavaScriptin yhtenä ongelmana pidetään sitä, että se ei ole tyypitettyä, joka tarkoittaa sitä, että muuttujille ei ole määritetty tyyppejä. TypeScriptillä koitetaan

poistaa kehitysvaiheessa ongelmia, joita tulee, kun yritetään käsitellä muuttujaa väärän tyyppisenä. TypeScript kääntyy suorituksen aikana JavaScriptiksi ja se myös samalla tarkastaa tyyppivirheet, eikä virheiden ollessa käänny, vaan antaa kehittäjälle ilmoituksen virheestä. (Microsoft 2019.)

Vaikka React on kehitetty käytettäväksi JavaScriptillä, voidaan sitä silti kehittää myös TypeScriptiä käyttäen. TypeScriptillä kirjoitettua koodia pidetään kehittäjän silmissä helpommin luettavana, koska tyypit ovat luettavissa, jolloin tiedetään tarkalleen mitä parametreja annetaan ja mitä palautetaan. Sovelluskehittimeen saa liitettyä lisäosan, joka näyttää virheet koodissa. Lisäosa parantaa koodin laatua pitämällä huolen siitä, että koodi noudattaa kaikkia määriteltyjä sääntöjä.

4.3 iOS-käyttöjärjestelmä

iOS-käyttöjärjestelmä on Applen vuonna 2007 julkaistu käyttöjärjestelmä, joka on käytössä kaikissa Applen puhelimissa ja tableteissa. Vuonna 2018 Hesburger-mobiilisovelluksen käyttäjistä noin 32 prosenttia käyttivät iOS-käyttöjärjestelmää. Tämä antaa hyvää osviittaa Suomen mobiililaitteiden käyttöjärjestelmistä, sillä Hesburger-sovellusta on ladattu reilusti yli miljoona kertaa. Tällä hetkellä käyttöjärjestelmästä on meneillään kahdestoista versio. (Vincit 2019.)

iOS-sovelluksia voidaan kehittää kahdella ohjelmointikielellä, jotka ovat Swift ja Objective-C. Nykyään uudet sovellukset pääsääntöisesti kehitetään käyttämällä Swiftiä, mikä johtuu siitä, että Apple kehittää jatkuvasti kieltä ja antaa sille täyden tuen. Swiftissä on ominaisuuksia, joita on myös muissakin moderneissa kielissä, kuten Kotlinissa, jota Google tukee nykyään Javan sijasta (Kolehmainen 2019).

4.4 Xcode ja Swift

Xcode on sovelluskehitin, jolla voidaan kehittää iOS-käyttöjärjestelmään sovelluksia. Xcode toimii pelkästään Mac OS -käyttöjärjestelmissä, eli Applen työpöytätietokoneissa sekä kannettavissa tietokoneissa. Xcode osaa kääntää Swiftiä, C, C++ ja Objective-C:tä, joista Swift on de facto ohjelmointikieli. Xcoden avulla voidaan käyttöjärjestelmässä pyörittää myös emulaattoreita, joka mahdollistaa esimerkiksi iPhonen käytön Applen tietokoneilla. Tämän ansiosta kehittäjällä ei ole tarvetta kehittää sovellusta käyttäen fyysistä laitetta. Myös fyysisellä laitteella voidaan ajaa Xcoden avulla kehitettyä sovellusta kehitysvaiheessa, mikä mahdollistaa testauksen ennen sen julkaisua. (Apple Inc 2019.)

Swift on Applen vuonna 2014 julkaisema ohjelmointikieli, jolla kehitetään iOS-sovelluksia. Swift-projektin tarkoituksena on luoda paras olemassa oleva ohjelmointikieli työpöytä- ja mobiilisovelluksien kehittämiseen. Swift on kehitetty korjaamaan C-pohjaiset ohjelmointikielet, kuten Objective-C: n. Swift on turvallisempi kielenä kuin C-pohjaiset kielet. Muuttujat ovat

aina alustettu ennen käyttöä, taulukot ja kokonaisluvut tarkistetaan ylivuotojen varalta sekä muistia hallitaan automaattisesti. Swiftissä on myös innovatiivinen ominaisuus, valinnaiset muuttujat, joka tarkoittaa sitä, että muuttuja voi olla tyhjä. Käyttäessään väärin näitä muuttujia ohjelma ei käänny ja kehittäjälle annetaan varoitus väärinkäytetystä muuttujasta. (Apple Inc 2019.)

4.5 Material Design

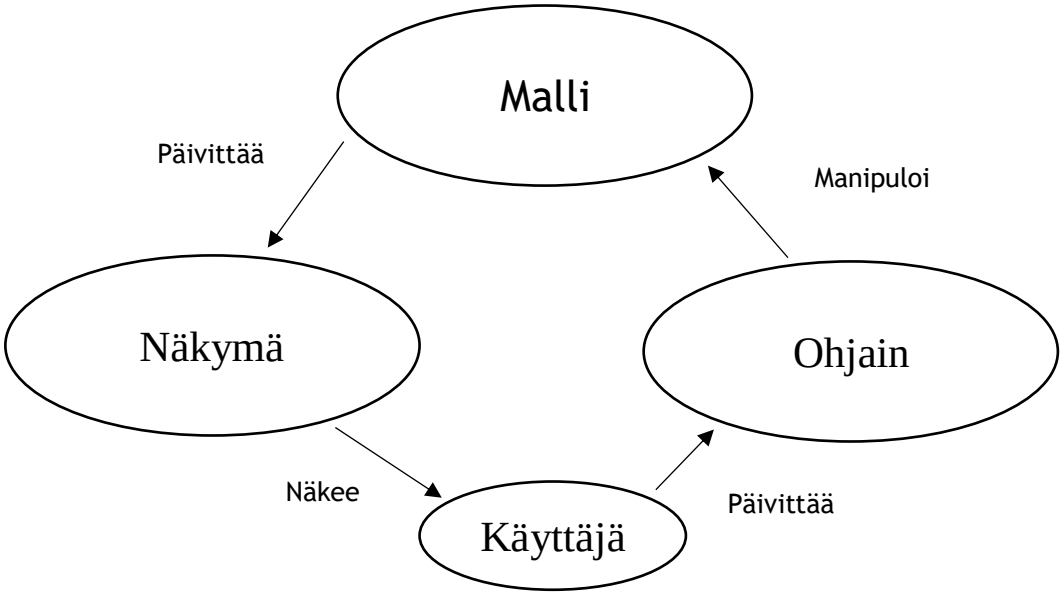
Material Design on Googlen kehittämä muotokieli. Material Design on kehitetty yhdistämään käyttökokemus eri laitteilla, käyttöjärjestelmillä sekä sovelluksien välillä. Material Design käyttää varjostusta, jonka avulla käyttöliittymäkomponentit saadaan näyttämään siltä, että ne sijaitsisivat eri tasoilla. Material Designissa on käyttöliittymäkomponenttien määrityksien ja sääntöjen lisäksi käyttökokemukseen liittyviä sääntöjä, kuinka esimerkiksi animaatiot eivät saa olla liian hitaita eikä nopeita. (Google 2019.)

Material Designista löytyy avoimen lähdekoodin sovelluskirjastoja Androidille, iOSille sekä web-alustalle. Sovelluskirjastojen avulla kehitystyötä nopeutetaan, kun samaa muotoilua voidaan käyttää eri kehitysalustoilla. Material Design on käytössä Googlen kehittämissä sovelluksissa, kuten YouTubessa, Play Kaupassa sekä Gmailissa. (Google 2019.)

5 MVC-arkkitehtuuri

Sovelluksen loppukäyttäjän käyttäessä sovellusta, käyttöliittymä manipuloi olioita koodissa, jolloin sovelluksen tila muuttuu ja oikein suunniteltuna se vähentää mahdollisia virhetiloja. Hyvä sovellus näyttää tilan muutokset reaaliaikaisesti käyttöliittymässä ja sovelluksen suunnitteluun tarvitaan arkkitehtuuri, jolla pyritään estämään virhetiloja. (Reenskaug & Coplien 2009.)

Model-View-Controller (MVC) arkkitehtuuri (Kuvio 1) määrittää oliot kolmeen eri rooliin malliin, näkymään ja ohjaimeen. Jokainen näistä kolmesta on eritelty toisistaan ja oliot kommunikoivat toisten tyyppisten olioiden kanssa. Malli-oliot määrittävät sovelluksen datamallit, joita sovellus prosessoi ja manipuloi. Datamalli voi esittää esimerkiksi sovelluksen käyttäjän tiedot. Datamalleilla voi olla yksi-yhteen suhteita tai yhdestä-moneen suhteita muiden datamallien kanssa. Näkymä-oliot ovat olioita, jotka ovat käyttäjän näkyvissä ja tätä kutsutaan käyttöliittymäksi. Pääasiassa näiden olioiden käyttötarkoitus on näyttää datamalleja ja mahdollistaa niiden muuttaminen loppukäyttäjän toimesta. Ohjain-oliot toimivat näkymien ja mallien välissä. Ohjain-oliot sisältävät logiikan, jonka avulla datamallia muutetaan ja sen jälkeen ne välittävät näkymään päivitetyn datamallin. (Apple Inc 2012.)



Kuvio 1: MVC-arkkitehtuuri

5.1 Hyödyt

Hyvin suunnitellun MVC-arkkitehtuurin yhtenä päämääränä on se, että sovelluksessa tulisi käyttää mahdollisimman montaa oliota, jotka ovat ainakin teoreettisesti uudelleen käytettävissä. Uudelleen käytettävissä olevat oliot vähentävät koodin määrää. Uudelleen käytettävällä koodilla sovellus on helpommin skaalautuva sekä ylläpidettävä. (Apple Inc 2012.)

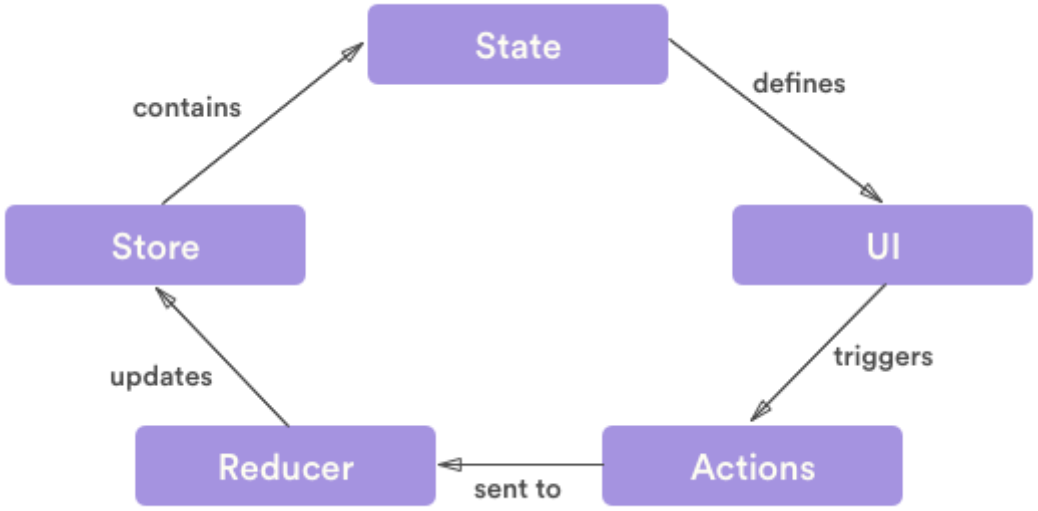
Yksi datamalli voidaan kytkeä moneen näkymään, millä saadaan kaikki näkymät tottelemaan yhtä datamallia ja sen muutokset aiheuttavat kaikkien näkymien muuttumiseen. Yksi näkymä voidaan koostaa useasta eri näkymästä, joten näkymät ovat uudelleen käytettävissä eikä kokonaista näkymää ole tarvetta tehdä, vaan voidaan tehdä osia, joita sitten yhdistellään ja käytetään sovelluksessa eri sivuilla. (Gamma & Helm & Johson & Vlissides 1994.)

5.2 React-Redux -sovelluksen arkkitehtuuri

React-sovelluksissa Redux on de facto tilanhallinnan kirjasto (Saring 2018). Reduxin avulla sovelluksella on yksi yhteinen olio, joka sisältää kaiken sovelluksen datan. Jokainen näkymä voi käyttää tätä yhteistä tilaa, jolloin esimerkiksi sovelluksen tila ei häviä sivua vaihtaessa. React-Redux -arkkitehtuuri (Kuvio 2) ei suoraan ole MVC-arkkitehtuuri, mutta niissä on samankaltaisuuksia, jolloin niitä voidaan verrata.

Redux actioneita voi verrata ohjaimiin ja niitä käytetään aina silloin, kun halutaan, että sovelluksessa tapahtuu jotakin. Esimerkiksi kun käyttäjä kirjautuu sisään sovellukseen, tehdään

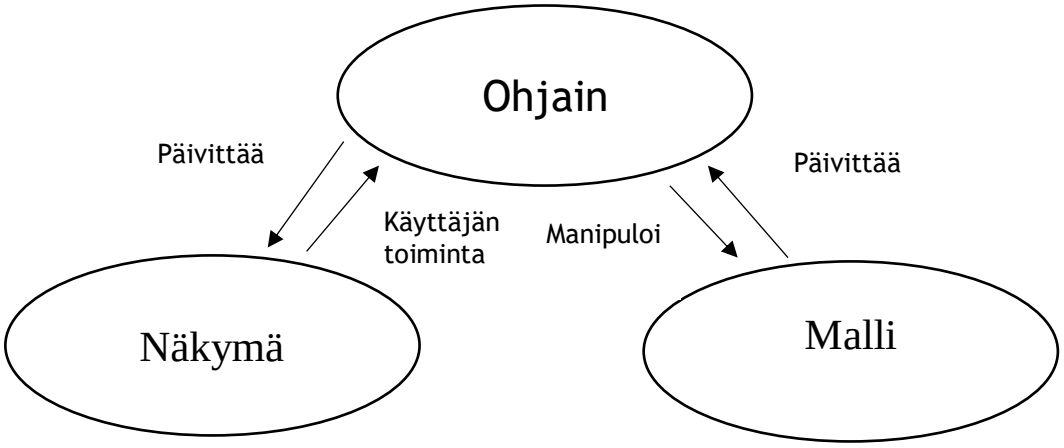
siitä Redux action. Reduxin reduceria voidaan verrata MVC-malliin, koska se hallitsee, kuinka sovelluksen tilaa muutetaan riippuen sille tulevasta actionista. React komponentteja voidaan verrata näkymiin ja niitä voi olla kahdenlaisia; komponentti, joka esittää vain datamallin tai toinen komponentti, joka hallitsee kaikki actionit ja tilanmuutokset sovelluksessa. (Levkovsky 2017.)



Kuvio 2: React-Redux -sovelluksen arkkitehtuuri (Levkovsky 2017)

5.3 Applen MVC-arkkitehtuuri

MVC-arkkitehtuuri on keskeinen osa hyvää suunnittelua iOS-sovelluksessa. Monet oliot ovat uudelleenkäytettäviä ja sovellus on helposti laajennettavissa. Monet Applen rajapinnoista ovat MVC-arkkitehtuuripohjaisia. Applen MVC-arkkitehtuuri (Kuvio 3) eroaa normaalista MVC-arkkitehtuurista siten, että näkymä ja malli eivät kommunikoi toistensa kanssa vaan kaikki tieto menee ohjaimen läpi. Arkkitehtuurin kannalta on parempi pitää malli ja näkymä erillään toisistaan, mikä parantaa mallien uudelleenkäyttöä, koska niitä ei ole sidottu yhteen näkymään, vaan samaa mallia voidaan käyttää useassa näkymässä. (Apple Inc 2012.)



Kuvio 3: Applen MVC-arkkitehtuuri

Applen standardiarkkitehtuuri on MVC (Apple Inc 2012) ja sovellusta tehdessä lähtökohtaisesti jokaiselle näkymälle luodaan oma ohjain, jolla voidaan esimerkiksi hakea dataa, kun näkymä on latautunut. Näkymät iOS-sovelluksissa ovat nimeltään Storyboard, jonne voidaan lisätä useita yhden ruudun näkymiä, jolle voidaan lisätä esimerkiksi tekstikenttiä ja nappeja. Storyboardin avulla voidaan määrittää eri näkymien suhteita toisiinsa. Applen UIKit-komponenttikirjasto tarjoaa monia valmiiksi tehtyjä näkymiä, kuten esimerkiksi tekstikenttiä, nappeja ja kuvanäkymiä. (Apple Inc 2019.)

6 Tutkimusmenetelmät

Prosessin tutkimiseen käytettiin tutkimusmenetelmiä, joiden avulla kartoitettiin projektimalliin liittyvät ongelmat. Tutkimustulokset ovat apuna kehitystyön jälkeen tapahtuneessa projektimallin arvioinnissa ja sen parantamisessa. Kehitysprojektissa käytettiin tutkimusmenetelminä tapaustutkimusta ja toimintatutkimusta. Tapaustutkimuksella sekä toimintatutkimuksella tutkittiin projektimallin laatua.

Tutkimusten tarkoituksena on löytää asioita, joita tutkija ei tiedä ja lisätä tietoa asioista muille. Kaikilla tutkimuksilla on kolme tarkoitusta, jotka ovat uuden tiedon luominen, tietoväitteiden testaaminen sekä uuden teorian luominen. Tutkimuksen tärkein päämäärä on tuottaa alkuperäistä tietoa, yleensä tuomalla kriittisiä näkökulmia olemassa oleviin tietoperustoihin. (McNiff & Lomax & Whitehead 2010.)

6.1 Tapaustutkimus

Tapaustutkimukset ovat yleensä käytössä psykologiassa, sosiologiassa tai poliittisessa tieteessä. Näillä aihealueilla tapaustutkimuksen päämääränä on lisätä tietoa yksilöistä, ryhmistä tai organisaatioista sekä sosiaalisista, poliittisista ja niihin liittyvistä ilmiöistä. Sovelluskehityksessä tapaustutkimusta käytetään, kun tutkitaan kuinka kehitys, toimenpiteet ja ylläpito tapahtuu sovelluskehittäjän näkökulmasta eri olosuhteissa. (Runeson & Höst 2008.)

Tapaustutkimuksen loppuun viemisessä on viisi päävaihetta, jotka ovat tapaustutkimuksen suunnittelu, tiedon keräämisen menetelmien määrittely, tiedon keruu, analyysi kerätystä tiedosta sekä raportointi. Tiedon keruu ja analysointi voidaan hoitaa asteittain. Joustavuuteen on myös rajana se, että tapaustutkimuksella tulisi olla tietyt päämäärät alusta asti. Tapaustutkimus on joustava, mutta se ei tarkoita sitä, että tutkimuksen suunnittelu olisi turhaa, vaan päävastoin hyvä suunnitelma tapaustutkimukselle on välttämätön sen onnistumiseen. (Runeson & Höst 2008.)

6.2 Toimintatutkimus

Toimintatutkimus on keinojen löytämistä toimintatapojen parantamiseksi, joten se on tiedon lisäämistä. Toimintatutkimuksen päämääränä on kehittää osaamista toimintatapojen parantamiseksi, mikä eroaa perinteisten tutkimusten päämääristä. Toimintatutkimuksessa keskitytään oppimisen parantamiseen eikä käyttäytymisen parantamista. (McNiff & Lomax & Whitehead 2010.)

Toimintatutkimuksessa on kaksi vaihetta projektin suunnitteluun. Ensimmäinen osio on suunnitelman luominen projektista ja siitä miksi se toteutetaan juuri tällä tavalla, mikä antaa perustelut omalle tekemiselle. Toinen vaihe on mukana olevien logististen ja käytännöllisten näkökulmien pohdiskelut, joihin lukeutuvat projektin ajankohta ja projektin sidosryhmät. Syyt ja keinot ovat toisiinsa liittyviä, koska ne perustuvat arvoihin. (McNiff & Lomax & Whitehead 2010.)

Toimintatutkimus sopii hyvin ketterillä menetelmillä toteutettuun projektiin, koska sovellus toteutetaan osissa, jolloin jokaisen osan jälkeen voidaan kerätä taustateorioita ja analysoida niitä. Analysoinnin jälkeen voidaan ottaa osan aikana opitut asiat huomioon seuraavassa toteutuksessa ja jatkuvasti kehitetään omaa toimintaa.

6.3 Reliabiliteetti ja validiteetti

Tutkimuksen luotettavuus riippuu tutkimuksen reliabiliteetista ja validiteetista. Reliabiliteetilla tarkoitetaan tutkimustiedon laatua. Laadullisessa tutkimuksessa reliabiliteettia ei kuitenkaan yleensä ottaen tarkoiteta laatua, vaan reliabiliteetin tarkoitus on lähempänä tutkimuksen käyttövarmuutta. Validiteetti ilmaisee kuinka hyvin tutkimus mittaa tutkittavan ilmiön

ominaisuutta. Validiteettiin vaikuttaa kuinka tutkimusongelmaa halutaan tutkia. (Golafshani 2003.)

Tässä opinnäytetyössä reliabiliteettia voidaan arvostella tutkimalla, onko projektimalli kuinka käyttövarma ja voidaanko sitä käyttää oikeassa projektissa. Validiteettia voidaan arvostella työssä tutkimuksen laadun, perusteellisuuden ja luotettavuuden mukaan.

7 Projektimallin suunnitelma

TypeScript kääntäjä kääntää koodin JavaScriptille. On olemassa erilaisia kääntäjiä, joilla voidaan kääntää ohjelmointikieltä toiseksi, mutta ei ole olemassa kääntäjää, jolla pystyisi kääntämään TypeScriptistä koodia Swiftiksi. Tällaista kääntäjää ei ole järkevää lähteä rakentamaan itse, koska sen tekemiseen ja testaamiseen menisi liian kauan. Projektimalleja ei löytynyt, joissa olisi juuri React-sovelluksen pohjalta tehty natiivisovellus hyödyntäen mahdollisimman paljon alkuperäistä React-sovelluksen arkkitehtuuria. Projektimallin suunnitelma tehtiin pääasiassa tutkimalla ja vertailemalla arkkitehtuuria, sekä käymällä läpi suunnitelma kokeneen iOS-kehittäjän kanssa.

7.1 Datamallit

JavaScriptissä datamalleja ei ole tyypitetty, joten JavaScript-koodista on mahdotonta lukea ja kirjoittaa datamallit uudestaan Swiftillä. Onneksi tässä tapauksessa oli käytetty TypeScriptiä, jossa kaikki datamallit tyypitetään, joten datamallien uudelleen kirjoittaminen Swiftille pitäisi onnistua helposti.

Esimerkkinä (Kuvio 4) käytettiin henkilöä, jolta löytyy merkkijonot etu- ja sukunimi, syntymäaikaa, joka on päivämäärä sekä ikä, joka on kokonaisluku. Vasemmalla puolella oleva esimerkki on TypeScript-rajapintaluokka, jota voidaan käyttää luomalla uusi muuttuja ja määrittelemällä sen tyypiksi henkilö, jolloin sillä tulisi olla kaikki nämä kentät. TypeScriptissä on käytössä myös luokat, mutta tässä sovelluksessa on käytetty pelkästään rajapintaluokkia. Käyttäjäliittymät eroavat luokista siten, että ne ovat virtuaalisia rakenteita, jotka ovat olemassa pelkästään TypeScriptin kontekstissa, jolloin TypeScript kääntäjä käyttää niitä pelkästään tyyppien tarkistamiseen. Swiftissä käytetään luokkaa, jolle voidaan määritellä joko var tai let -muuttujia, joista var-muuttujaa voidaan muuttaa luonnin jälkeen, mutta let-muuttujaa ei voida muuttaa luokan luonnin jälkeen.

```
interface Person {
    firstname: string;
    lastname: string;
    birthday: Date;
    age: number;
}
```

```
class Person {
    var firstname: String
    var lastname: String
    var birthday: Date
    var age: Int
}
```

Kuvio 4: TypeScript ja Swift datamallit

7.2 Rajapintakutsut

Tässä sovelluksessa rajapintana on REST, joka on muodostunut standardiksi sovelluksissa, jossa palvelin- ja käyttösovellus ovat erillään toisistaan. Rajapintaa voidaan kutsua monesta eri sovelluksesta, esimerkiksi tässä tapauksessa web- sekä mobiilisovelluksesta. Mobiilisovelluksissa REST-rajapinnat ovat käytetyimpiä (Tea 2017).

Hyvään sovellusarkkitehtuuriin kuuluu, että rajapintakutsut tehdään niin, että niitä voidaan käyttää monessa paikkaa ilman, että joudutaan koko kutsua kirjoittamaan joka kerta uudelleen. Redux-arkkitehtuurissa rajapintakutsuja käytetään pääsääntöisesti actioneissa, jolloin ne ovat uudelleenkäytettäviä useissa eri paikoissa. Applen MVC-arkkitehtuurissa rajapintakutsuja tehdään ohjaimissa, joten rajapintakutsut joudutaan tekemään erillisiin tiedostoihin funktioiksi, jotta niitä voidaan kutsua useassa eri ohjaimessa. Tämä vähentää ylimääräisen koodin määrää ja jatkossa on helpompi ylläpitää koodia, kun muutokset voidaan tehdä yhteen paikkaan monen paikan sijasta.

Web-sovelluksissa rajapintakutsuihin käytetään moderneista selaimista löytyvää Fetch-rajapintaa. iOS-sovelluksissa rajapintakutsuihin yksi käytetyimmistä kirjastoista on nimeltään Alamofire (Kuprenko 2018). REST-rajapinta palauttaa datan sovellukseen JSON-muodossa (JavaScript Object Notation) ja web-sovelluksissa dataa voidaan käyttää suoraan JSON-muodossa, mutta iOS-sovelluksessa joudutaan käyttää kirjastoa, jotta JSON-data saadaan muutettua Swift luokaksi. Käytetyin kirjasto tähän on nimeltään SwiftyJSON. Tämä datan muuttaminen Swift luokiksi on melko työlästä, jos vastauksena saadaan monimutkaisia JSON-datamalleja. Verrattuna React-sovellukseen tämä tuottaa lisää työtä, koska data pitää siirtää Swift malliksi ja sen lisäksi pitää testata, että se on oikein siirretty.

Rajapintakutsujen syntaksi eroaa sen verran, että ne joudutaan kirjoittamaan kokonaan uudelleen. Web-sovelluksen rajapintakutsuja voidaan käyttää siten hyödyksi, että katsotaan mihin tehdään kutsu ja mikä datamalli kutsulle annettaan parametriksi, sekä mitä kutsu palauttaa onnistuessaan ja mikä sen datatyyppi on.

7.3 Ohjaimet

Applen ohjaimissa logiikkana on se, että näkymän latautuessa tehdään rajapintaan kutsu, minkä jälkeen saatu data näytetään näkymässä. React-sovelluksessa samaa käytäntöä käytetään, näkymän latautuessa tehdään rajapintaan kutsu, jonka jälkeen näkymä päivittyy, kun tila on päivittynyt saadulla datalla. Ohjaimissa toteutetaan myös käyttäjän toiminnasta aiheutuvat rajapintakutsut, kuten datamallin muokkaaminen tai lisääminen. Applen arkkitehtuurissa nämä ovat ohjaimissa olevissa funktioissa, kun taas React-arkkitehtuurissa näitä funktioita voidaan käyttää komponenteissa eli pienimmissä näkymissä, joka saattaa vaikeuttaa arkkitehtuurin siirtämistä.

Redux-arkkitehtuurissa tilan hallintaan liittyvät funktiot kirjoitetaan actioneihin, jolloin niitä voidaan käyttää uudelleen ja samalla selkeytetään ohjaimien toimintaa, koska vähennetään logiikkaa ohjaimissa. Tämä on asia, joka toteutetaan myös iOS-sovelluksen arkkitehtuuriin, jotta ohjaimet pysyvät selkeämpinä ja funktiot ovat uudelleen käytettävissä.

MVC-arkkitehtuurissa pidetään tärkeänä, että ohjaimissa ei säilytetä sovelluksen tilaa, eli malleja. Redux-arkkitehtuuri ratkaisee tämän ongelman siten, että sovelluksessa on yleinen tila, jota voidaan käyttää jokaisessa ohjaimessa. iOS-sovelluksessa tämä onnistuu luomalla singletonin, eli staattisen muuttujan datamalliin, jota pystytään käyttämään eri ohjaimissa.

7.4 Näkymät

React-sovelluksen käyttöliittymässä on yleensä käytetty useita komponentteja tai näkymiä. iOS-sovelluksessa voidaan luoda näkymiä esimerkeiksi napeille, joille voidaan antaa tyylejä valmiiksi, jotta tyylejä ei joudu esimerkiksi joka kerta muuttamaan jokaiselle näkymälle erikseen ja muutokset on helpompi tehdä. React-sovelluksen yleiset näkymät on hyvä siirtää eteenpäin iOS-sovellukseen, mutta käyttöliittymänäkymien siirtäminen on hankalaa. React-näkymä eroaa niin paljon iOS-näkymästä, joten voi olla helpompaa katsoa käyttöliittymää visuaalisesti ja sen perusteella tehdä iOS-näkymä.

Näkymät tehdään Xcodessa Storyboardeille, joita voidaan muokata Xcoden avulla, eikä kehittäjän tarvitse kirjoittaa koodia itse ollenkaan saadakseen käyttöliittymän toteutettua. Yleensä tämä taas aiheuttaa sen, että oppimiskäyrä on jyrkempi, koska vaihtoehtoja kaikille ominaisuuksille on niin paljon ja niitä joutuu etsimään ja tutkimaan. Näkymät on mahdollista myös luoda koodin avulla ohjaimissa samalla tavalla kuin React-sovelluksessa, mutta tässä tilanteessa opinnäytetyön tekijä piti Storyboardin käyttämistä parempana ratkaisuna oppimisen kannalta.

7.5 Suunnitelman yhteenveto

Suunnitelmassa lähdettiin liikkeelle mallien luomisella, jotka ovat helpoimmin siirrettävissä. Rajapintakutsut vaativat myös näitä malleja, jonka takia mallit on tehtävänä ensimmäisenä.

Mallien jälkeen kehitetään rajapintakutsut, jotka ovat myös siirrettävissä suoraan koodia katselemalla, mutta suuremman työn takana. Yleensä rajapintakutsuihin ei tehdä testejä, vaan tarkastellaan sitä mitä kutsusta saadaan vastaukseksi, mutta tässä tapauksessa on helpointa tehdä testit ja olettaa, että kutsut onnistuvat ja vastauksena saadaan malleja käyttävät vastaukset kutsusta. Tämä helpottaa jäljellä olevaa kehitystä siten, että kutsuja voidaan suoraan käyttää ohjaimissa, koska tiedetään, että kutsut toimivat. Rajapintakutsujen testauksen jälkeen luodaan näkymät React-sovelluksen käyttöliittymän mukaan ja tehdään käyttöliittymänäkymät valmiiksi, jonka jälkeen ohjaimiin voidaan lisätä käyttölogiikkaa näkymiin, kuten nappeihin.

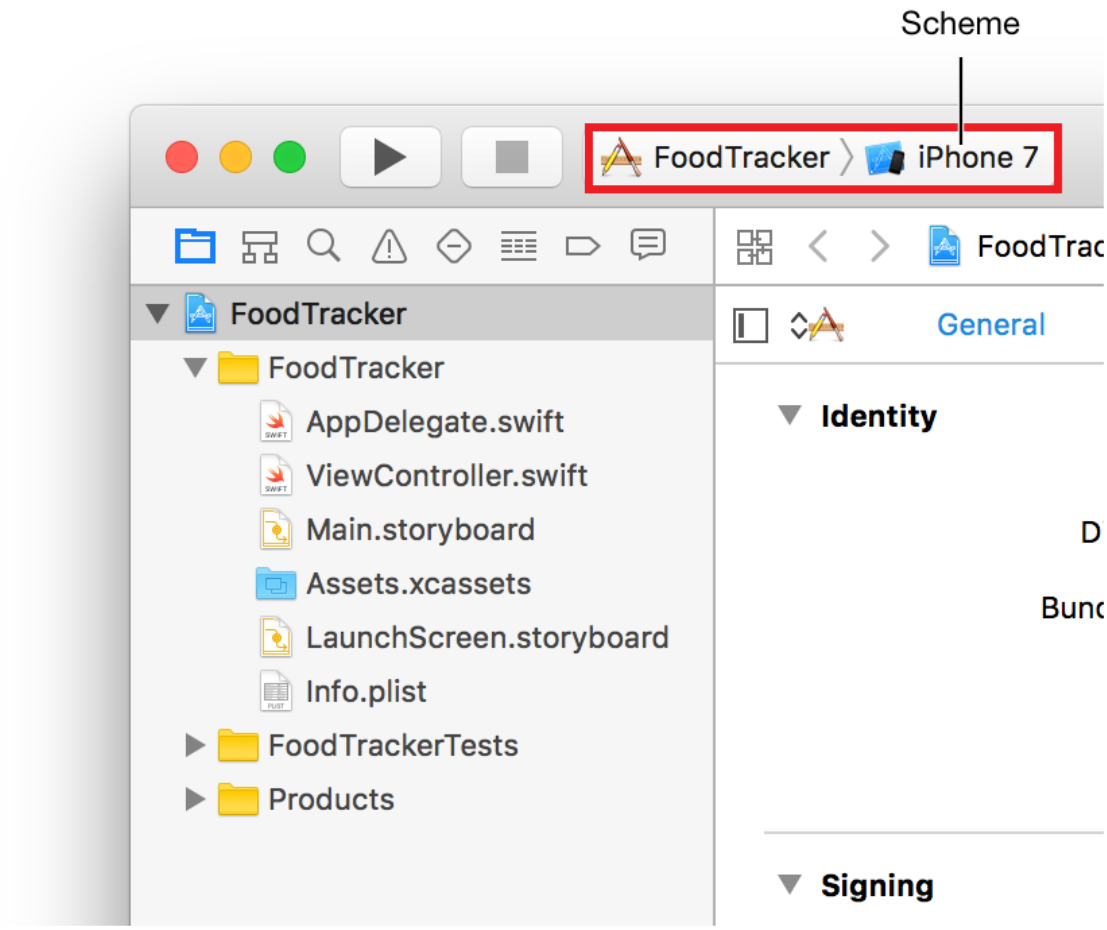
Suunnitelman aikatauluksi oli alustavasti otettu neljä viikon kestävää sprinttiä, joiden aikana voitaisiin kokeilla projektimallia käytännössä. Ensimmäiseen sprinttiin suunniteltiin tehtäväksi projektin rakenne, tarvittavat kirjastojen asennukset sekä mallit. Toisessa sprintissä toteutettaisiin rajapintakutsut ja niiden testaaminen. Kolmannen sprintin aikana tavoitteena olisi saada kirjautumissivu valmiiksi sekä navigaatio sisäänkirjautuneena käyttäjänä sekä yhden käyttäjäroolin alustavat näkymät ja toiminnallisuus. Viimeisessä sprintissä toisen käyttäjäroolin alustavat näkymät ja toiminnallisuus.

8 Toteutus

Toteutus tehtiin yhden kehittäjän tiimissä, jossa Scrum-mallia ei voida toteuttaa oikeaoppisesti, joten sitä sovellettiin mukautettuna versiona. Sovelluksen määritykset oli tehty alkuperäisessä projektissa, jossa web-sovellus toteutettiin, joten tässä tapauksessa ei tarvittu sprinteille määrittelyä tuotteen omistajalta. Toteutuksessa huomioidaan miten web-sovelluksen arkkitehtuuria tulisi muuttaa, jotta sovellus olisi mahdollisimman helposti muutettavissa.

8.1 iOS-projektin alustaminen ja mallit

Ensimmäinen sprintti alkoi luomalla iOS-sovellusprojekti ja sen alustamisella. Xcode-ohjelmassa projektin luominen on tehty helpoksi ja projektia luodessa voidaan päättää, luodaanko muun muassa yhden näkymän sovellus tai monen näkymän sovellus. Oletuksena projekti luodaan yhden näkymän sovelluksena sekä projektiin ei ole lisätty yksikkötestejä. Projektiin voidaan lisätä näkymiä sekä yksikkötestejä luomisen jälkeen, joten luomisen ohella ei ole tarvetta lisätä näitä. Projektin luomisen jälkeen voidaan sovellusta ajaa Xcoden avulla simulaattorissa, jossa voidaan valita millä laitteella halutaan sovellusta ajaa (Kuvio 5). Simulaattorilla voidaan simuloida sovellusta jokaisella iPhonella sekä iPadilla, joille sovellusta kehitetään.



Kuvio 5: Xcode simulaattorin laitteen valitseminen (Apple Inc 2018)

Projektiin luotiin kansiot MVC-arkkitehtuurin mukaisesti, jotta tiedostojen käsittely tulevaisuudessa on helpompaa, kun tiedetään mistä tietyntyyppiset tiedostot löytyvät ja joillakin tiedostoilla voi olla samankaltaisia nimiä. Muut iOS-projektin oletuksena luodut tiedostot lisättiin tukikansioon, jotta ne eivät olisi tiellä.

Alamofire ja SwiftyJSON kirjastot eivät ole käytettävissä iOS-sovelluksissa oletuksena, joten ne joudutaan asentamaan itse. Näiden asentamiseen on olemassa useita tapoja asentaa. Näiden asentamiseen käytettiin CocoaPodsia, joka on muodostunut standardiksi kirjastojen asentamiseen iOS-sovelluksiin. CocoaPods joudutaan asentamaan komentoriviltä tietokoneelle, jotta sitä voidaan käyttää. Asennuksen jälkeen projektiin alustetaan CocoaPodsin avulla Podfile (Kuvio 6), jolle määritetään käytettävät kirjastot. Kirjastojen määrittelyjen jälkeen voidaan kirjastot asentaa projektiin CocoaPodsin komentorivikomennolla, mikä lataa projektiin kirjastot ja asentaa ne käytettäväksi projektiin. Kirjastojen asennuksen yhteydessä CocoaPods luo projektiin Xcode workspace -tiedoston, jonka kautta projekti joudutaan avaamaan, koska Xcode ei tunnista muuten kirjastoja.

```
# Uncomment the next line to define a global platform for your project
# platform :ios, '9.0'

target 'App' do
  # Comment the next line if you're not using Swift and don't want to use
dynamic frameworks
  use_frameworks!

  # Pods for App
    pod 'SwiftyJSON', '~> 4.3.0'
    pod 'Alamofire', '~> 4.4'

end
```

Kuvio 6: Podfile

Malleja lähdettiin kirjoittamaan katsomalla web-sovelluksen lähdekoodia ja muuttamalla ne Swift luokiksi. TypeScriptissä voidaan määrittää, että muuttujan arvo on valinnainen, eli se voi olla myös tyhjä. Web-sovelluksessa oli käytetty valinnaisia arvoja kentillä, jotka eivät olleet rajapinnan vastauksissa valinnaisia arvoja, mikä sekoittaa sovelluksen osissa käytettävää logiikkaa. Tähän voinee vaikuttaa se, että niitä on käytetty rajapintaan lähetettävinä malleina, jolloin rajapinta luo nämä kentät. Mallien sijainti web-sovelluksessa oli sekainen eikä ollut varmuutta, että mitä malleja rajapintakutsuissa on käytössä, jolloin rajapintakutsuja jouduttiin tutkimaan tarkemmin.

8.2 Rajapintakutsut

Rajapintakutsuja (Kuvio 7) lähdettiin tekemään katsomalla web-sovelluksen lähdekoodista kutsuja, sekä siitä, mitä kutsusta tulee vastaukseksi. Rajapintakutsut luotiin projektiin rajapinta-kansioon ja omiin tiedostoihin, jotta ne olisivat uudelleen käytettäviä ohjaimissa ilman, että koodia tarvitsee kopioida useisiin paikkoihin. Tiedostoihin tehtiin funktiot, joiden avulla JSON-data saataisiin muutettua Swift luokiksi. Rajapintakutsu funktioille annettiin parametreinä päätösfunktio, joka palauttaa valinnaisen objektin tai taulukon objekteja ja virheen, joka on totuusarvomuuttuja, jonka avulla voidaan päätellä, että tapahtuiko rajapintakutsussa virhe vai ei. Jos virhettä ei ole, niin silloin kutsu oli onnistunut ja data voidaan päivittää ohjaimessa näkymään. Virhetilanteessa näytetään käyttäjälle virhe, jolloin käyttäjän ei tarvitse olettaa, että data olisi hävinnyt.

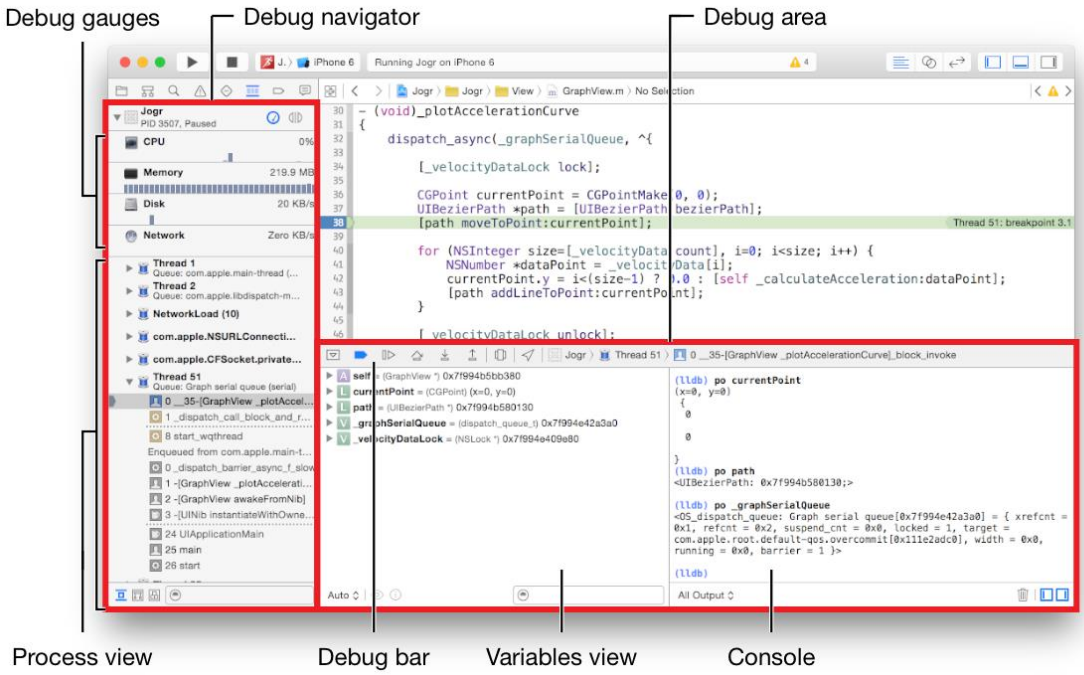
```
Alamofire.request(url, method: .get).validate().responseJSON { response in
    switch response.result {
    case .success(let value):
        let json = JSON(value)
        print("JSON: \(json)")
    case .failure(let error):
        print(error)
    }
}
```

Kuvio 7: Esimerkki Alamofire kutsusta SwiftyJSONin avulla

Sisäänkirjautumis-kutsussa rajapinnassa oli käytössä sisällöntyyppinä ”multipart/form-data” tyyppinen sisältö. Nykyaikaisista selaimista löytyvä fetch-rajapinnan avulla tämä voidaan tehdä normaalina post-pyyntönä, mutta Alamofire-kirjastoa käyttäessä joudutaan rajapinnasta käyttämään upload-pyyntöä, joka on pääasiallisesti tarkoitettu tiedostojen lataamiselle. Tämä hidasti kehitystä, koska dokumentointia jouduttiin käymään läpi oikean esimerkin löytämiseksi.

Alkuperäinen suunnitelma oli, että rajapintakutsut ja datan siirtäminen JSON-datasta Swift luokaksi olisi testattu yksikkötesteillä ennen käyttöönottoa ohjaimissa. Tällöin käytössä olisi ollut testilähtöinen kehitys. Luokkiin olisi pitänyt periä Swiftin XTestCase-luokka, jotta luokat olisivat olleet käytettävissä yksikkötesteissä. Se ei työnä olisi ollut suuri, mutta yksikkötestaamiseen ei tuntunut löytyvän ajallisesti tarpeeksi tehokasta tapaa.

Rajapintakutsut päädyttiin ottaa käyttöön ilman yksikkötestien tekemistä. Rajapintakutsut olivat käytössä jo web-sovelluksessa, joten niiden oletettiin olevan kunnossa. Yksikkötestien tarkoituksena ei olisi ollut testata, että kutsut palauttavat oikeaa dataa, vaan se, että datat muutettaisiin oikein luokiksi. Lisäämällä koodiin tulosteita, jolloin Xcoden konsoliin tulostuu esimerkiksi muuttujia, voidaan nähdä ovatko muuttujan arvot oikeita. Xcodessa (Kuvio 8) voidaan lisätä pysäytyspisteitä, jolloin nähdään muuttujien arvot pysäytyspisteessä. Näillä menetelmillä rajapintakutsut päädyttiin testaamaan ja ne osoittautuivat nopeammaksi ratkaisuksi kuin yksikkötestien tekeminen.



Kuvio 8: Xcode-sovellus (Apple Inc 2018)

Rajapintakutsut ovat suojattuja ja niihin tarvitaan käyttöoikeustietue, joka validoidaan rajapinnassa ja sen jälkeen tarkastetaan käyttäjän oikeus rajapintaan. Tämä käyttöoikeustietue saadaan rajapinnasta tekemällä sisäänkirjautumiskutsu käyttäjätunnuksella ja salasanalla. Käyttöoikeustietueen tallentamiseen on käytännössä kaksi hyvää tapaa iOS-kehityksessä, jotka ovat staattinen muuttuja ja KeychainAccess-rajapinta. Staattiseen muuttujaan päädyttiin, koska se oli tässä vaiheessa kehittämistä helpompi ja nopeampi tapa lisätä. NSUserDefaults-rajapintaa ei voi käyttää, koska sitä ei katsota tietoturvalliseksi minkään arkaluontoisen tiedon säilyttämiseen.

Sovelluksen käynnistyttyä uudelleen joudutaan sisään kirjautumaan uudelleen, koska staattinen muuttuja nollaantuu sovelluksen käynnistyessä uudelleen. Rajapinnoissa on yleensä käytössä päivittävä käyttöoikeustietue, jonka avulla voidaan tehdä kutsu saadakseen käyttöoikeustietue, mutta tässä rajapinnassa ei ollut toteutettuna vielä tällaista ominaisuutta. Tällaisessa tapauksessa jouduttaisiin käyttämään KeychainAccess-rajapintaa, jotta sovelluksen uudelleen käynnistämisestä huolimatta päivittävä käyttöoikeustietue olisi tallessa.

8.3 Ohjaimet

Ohjaimia alettiin tekemään katsomalla web-sovelluksen lähdekoodista, että mitä sovelluksen näkymä tekee, kun se on latautunut. Reactissa luokkakomponenteissa on käytössä lifecycle metodi, joka on nimeltään ComponentDidMount (Kuvio 9). Sen avulla voidaan näkymän latautumisen jälkeen tehdä rajapintakutsuja, jotta sivulla voidaan näyttää dataa. iOS-sovelluksessa

on samankaltainen metodi nimeltään viewDidLoad (Kuvio 9), jonka avulla voidaan toteuttaa sama logiikka kuin web-sovelluksessa.

```
override func viewDidLoad() {
    super.viewDidLoad()
}
```

```
componentDidMount() {

}
```

Kuvio 9: iOS-sovelluksen ja React-sovelluksen metodit

Ohjaimiin voidaan lisätä funktioita, jotka ovat käytössä ohjaimessa. Tyypillisesti ohjaimiin lisätään funktioita, joilla tehdään käyttäjän toiminnasta, kuten napin painalluksesta, jotakin. Ohjaimiin lisättiin funktiot rajapintakutsujen käyttöä varten. Rajapintakutsuja varten joudutaan luomaan ohjaimen sisällä muuttuja, jossa tarkoituksena on säilyttää rajapintakutsuista saatua dataa. Rajapintakutsujen funktioille joudutaan antamaan lopetusfunktio, joka hoitaa logiikan saadun datan ja mahdollisen virheen sattuessa. Kutsun onnistuessa vaihdetaan datan hallintaan luodun muuttujan arvo kutsusta saadulla arvolla. Mahdollisessa virhetilanteessa käyttäjälle näytetään virheilmoitus.

Xcode tarjoaa vaihtoehdon listaohjaimen luomiseen, jonka avulla saadaan ohjaimelle valmiina olevat funktiot riviosioiden lisäämiksesi listalle. Luoduissa funktioissa voidaan määritellä esimerkiksi sektioiden määrä, riviosioiden määrä ja riviosion näkymä. Tämä vähentää listaohjaimiin kirjoitettavan koodin määrää.

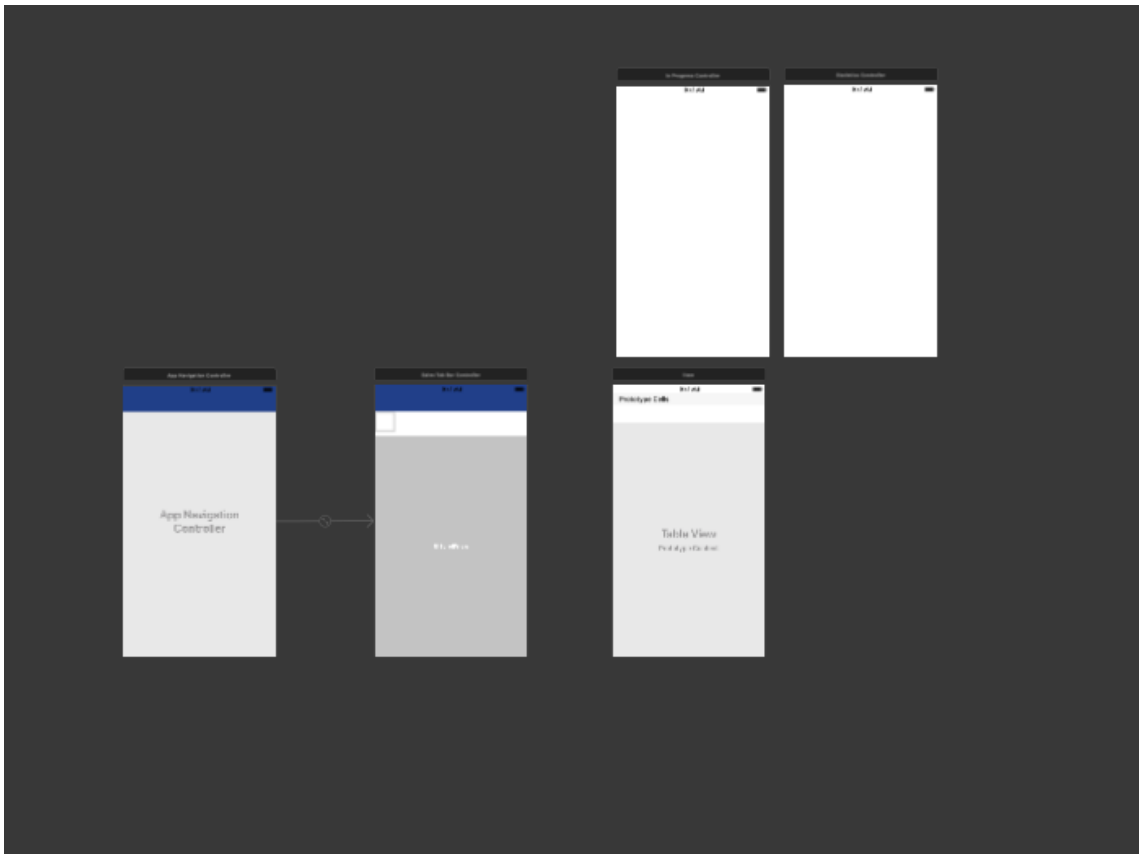
8.4 Näkymät

Näkymien tekemiseen Xcodessa on näkymärakentaja, jonka avulla näkymälle voidaan lisätä erilaisia komponentteja. Komponenttien lisääminen onnistuu etsimällä komponentti listasta ja raahaamalla ja pudottamalla se halutun näkymän päälle. Näkymärakentajan avulla onnistuu myös uusien näkymien lisääminen Storyboardille ja näkymien välisten yhteyksien määrittäminen.

Näkymärakentajalla valitaan laite, jonka näyttökokoa käytetään rakentamisessa. Samalla laitteella näkymä näyttää samalta kuin näkymärakentajassa, mutta oletuksena näkymä näyttää erilaisilta laitteilla, jotka käyttävät eri näyttökokoa. Responsiiviset näkymät saadaan aikaan käyttämällä Xcoden tarjoamaa Auto Layoutia, joka laskee automaattisesti näkymän ja komponenttien mitat näyttökoon perusteella.

Sisäänkirjautumissivu ja käyttäjäroolit lisättiin jokainen omalle Storyboardille (Kuvio 10), jolloin on helpompi hahmottaa sovelluksen näkymät. Sisäänkirjautumissivun Storyboardille

lisättiin oma App Navigation näkymä, koska sovellukseen joutuu kirjautumaan sisälle päästäkseen navigoimaan käyttäjäroolien välillä.



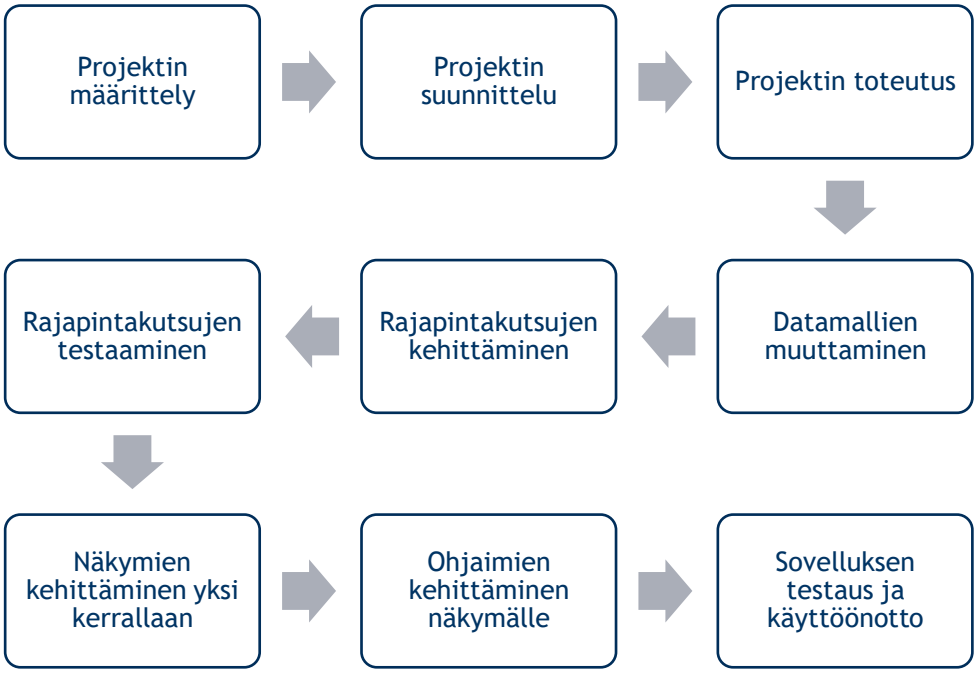
Kuvio 10: Yhden käyttäjäroolin näkymät Storyboardilla

Näkymille perittiin App Navigation ohjain, jonka avulla näkymiin saadaan navigointipalkki, johon voidaan lisätä esimerkiksi sovelluksen nimi ja nappi vetolaatikkomenun aukaisemiseksi. Sovelluksessa kahdella käyttäjäroolilla oli vetolaatikkomenun lisäksi työkalupalkkimenu, jonka avulla helpotetaan käyttäjäroolin näkymien navigointia. Sovelluksen työpalkkimenu oli suunniteltu olevan ylhäällä Material Designin mukaan, mutta iOS-sovelluksissa standardina työkalupalkkimenu on alhaalla. Tämän takia jouduttiin ottamaan XLPagerTabStrip kirjasto, jolla saataisiin työpalkkimenu ylhäälle. XLPagerTabStripille lisättiin näkymä, jonka sisälle lisättiin työkalupalkkimenun lisäksi vieritettävä näkymä, jolla näytetään työkalupalkkimenun valinnan perusteella oikea näkymä. Näkymälle lisättiin ohjain, jossa määritettiin kaikki työkalupalkkimenussa näytettävien näkymien ohjaimet. Näytettävissä näkymissä määritettiin millä nimellä näkymät näkyvät työkalupalkkimenussa.

Käyttäjäroolien Storyboardille lisättiin näkymä vetolaatikkomenua varten. Tähän jouduttiin asentamaan SideMenu-kirjasto, jotta saatiin ponnahdusikkuna aukeamaan vetolaatikkomenua painaessa. Näkymälle lisättiin linkkejä eri käyttäjärooleihin, jotta navigaatio niiden välillä olisi mahdollista.

9 Kehitetty projektimalli

Kehitetyssä projektimallissa (Kuvio 11) projekti aloitetaan määrittelyllä. Projektin määrittelyssä selvitetään projektin tekijät, tavoitteet sekä aikataulu. Projektin suunnittelussa arvioidaan, onko alkuperäinen web-sovellus muunnettavissa. Projektimallin yhtenä vaatimuksena on TypeScriptin käyttö web-sovelluksessa. Suunnitteluvaiheessa määritetään sovellusarkkitehtuuri ja kyseisessä projektimallissa on käytetty MVC-arkkitehtuuria. Suunnitteluvaiheeseen kuuluu myös sprinttien suunnitteleminen. Sprinttien määriin sekä sisältöihin vaikuttavat sovelluksen koko. Pienelle sovellukselle voidaan suunnitella neljä viikon kestävää sprinttiä, joista ensimmäisessä alustetaan sovellus ja muutetaan datamallit. Toisessa sprintissä rajapintakutsut kehitetään ja testataan. Kolmannessa sprintissä kehitetään puolet näkymistä ja niiden ohjaimet. Viimeisessä sprintissä kehitetään loput näkymistä ja niiden ohjaimista. Toteutuksen jälkeen sovellusta testataan ja otetaan käyttöön.



Kuvio 11: Kehitetty projektimalli

Jokainen projekti on erilainen, joten sprinttien sisällöt saattavat muuttua projektin edetessä. Projektille on tarvetta varata riittävästi aikaa, jotta projektissa voidaan ottaa huomioon projektikohtaiset ongelmat. Sovelluksen testauksen laajuus on projektikohtaista, mikä voi vaikuttaa sprinttien sisältöihin. Myös käyttöönotto riippuu projektista ja sen aikana voi tulla ilmi ongelmia, jotka voivat viivästyttää projektia. Projektimallia voidaan soveltaa käyttämään toista sovellusarkkitehtuuria tai kehitysalustaa.

10 Yhteenveto ja johtopäätös

Kehitystyön tuloksena syntyi projektimalli, jonka avulla sovellus voidaan siirtää natiivisovellukseksi mahdollisimman ketterästi. Kehitystyön aikana web-sovelluksen arkkitehtuurissa huomattiin puitteita, jotka ovat hidastivat työtä. Arkkitehtuurisiin huomioihin otettiin kantaa, jotta niitä voitaisiin parantaa tai uuden sovelluksen kehittäessä tehdä paremmin. Projektimallin avulla nopeutetaan natiivisovelluksen kehitystyötä sen sijaan, että se rakennettaisi kokonaan alusta käyttämättä valmista arkkitehtuuria.

Kehitystyön aikana toteutettu sovellus jäi keskeneräiseksi ja siitä jäi toteuttamatta osa toiminnoista käyttöliittymälle, johon vaikutti sovelluksen koko. Kehitystyö ei edennyt niin pitkälle, että käyttöliittymä olisi saatu muutettua aivan samanlaiseksi kuin web-sovelluksessa, vaikka molemmissa versioissa oli käytössä sama käyttöliittymäkirjasto. Tämä johtuu siitä, että web-sovelluksen komponentit olivat kustomoituja eikä täysin oletuksena olevia. Kaikki mallit sekä rajapintakutsut saatiin siirrettyä, joten sovelluksesta jäi pelkästään käyttöliittymien luominen ja viimeistely.

Ensivaikutelmat projektimallista olivat sellaisia, että projektissa ei olisi paljon yhtenäistä logiikkaa ja suurinta osaa koodista ei saisi käytettyä hyödyksi. Ensivaikutelmat tällä kertaa vaihtuivat projektin aikana ymmärryksen ja osaamisen lisääntyessä. Projektimallissa saatiin tehokkaasti siirrettyä logiikkaa täysin toiseen arkkitehtuuriin.

Projektimallin suunnitteluun tietoa hankittiin keräämällä sähköisiä aineistoja arkkitehtuureista sekä teknologioiden ominaisuuksista. Sähköisiä aineistoja löytyi paljon, mutta tarkkaa tietoa tutkimusongelmaan ei löytynyt. Projektimallin suunnitelmasta kerättiin tietoa yrityksen sisäisesti keskustelemalla kokeneempien kehittäjien kanssa parhaista ratkaisuista.

Tutkimuksen tulokset olivat odotettuja, sillä kehitystyö oli ajallisesti lyhyt sekä siinä oli yksi kehittäjä. Projektimallin luomiseen olisi tarvittu enemmän tutkimustietoja, mikä ei ollut tässä tapauksessa mahdollista, että siitä olisi saatu poistettua mahdolliset ongelmat kehitysvaiheessa. Nykyistä projektimallia voidaan kehittää eteenpäin tulevissa projekteissa. Kehitystyön perusteella voidaan arvioida projektimallilla toteutettujen projektien aikavaatimuksia, jotka kasvoivat hiukan suunnitteluvaiheessa arvioiduista. Reliabiliteetiltaan tutkimus oli kelvollinen, koska tutkimuksen tuloksena oli käyttövalmis projektimalli. Projektimallissa on käyttövarmuuteen parannettavaa, koska useasti projektien määrä projektimallilla vaikuttavat sen laatuun. Validiteettiin vaikuttivat tässä työssä vähäinen ajankäyttö sekä projektin toteuttaminen mukautetulla versiolla Scrum-mallista, mutta silti validiteetista pätevä. Työn toimeksiantajalta saatu palaute oli hyvää.

11 Oman osaamisen arvioiminen

Työn aihe valikoitui työnantajan tarpeista sekä omasta mielenkiinnosta iOS-sovelluskehitykseen. Android-sovelluskehitys oli itselle tuttua, joten oman osaamisen lisääminen iOS-alustalle oli tärkeätä. Projektin kehitystyön toteutusta ennen harjoittelin omatoimisesti iOS-kehitystä tekemällä pieniä sovelluksia sekä lukemalla useita artikkeleita.

Projekti oli haasteellinen, koska se oli melko suuri toteutettavaksi sekä tehtävään oli rajoitettu aika, jonka myötä asiat jouduttiin tekemään mahdollisimman tehokkaasti. Tästä johtuen rajapintakutsujen testaaminen jouduttiin jättämään väliin, mikä olisi ollut tärkeä asia oppimisen kannalta. Aikatauluun vaikutti myös se, että ennen projektisuunnitelman luontia jouduttiin opettelemaan iOS-kehitystä käyttämällä hyvin runsaasti aikaa.

Mallien luominen on normaalia kaikessa oliopohjaisessa ohjelmoinnissa, joten niiden luominen ei ollut haastavaa. Rajapintakutsuissa iOS-kehityksessä jouduttiin käyttämään kirjastoa, jonka käyttäminen alussa oli pitkälti opettelemista. JSON-datan muuttamiseksi Swift luokiksi päädyttiin käyttämään suosituinta kirjastoa, mutta kirjaston käyttäminen tuntui olevan melko hidasta. Toteutuksen jälkeen tähän käyttötarkoitukseen löytyi toinen kirjasto, joka käyttö saattaisi olla nopeampaa. Navigoinnin luominen osallistui melko haastavaksi, koska alkuperäisin ohjaimia ei voinut käyttää, jonka vuoksi jouduttiin ottamaan käyttöön kaksi kirjastoa. Näkymien luomisten vaikeus oli tiedossa jo ennen projektin alkua. Niiden vaikeus osoittautui todella aikaa vieväksi tämän hetkisellä osaamisella.

Työn aikana lisäsin tieto tasoani arkkitehtuureista. Ennen kehitystyötä minulla ei ollut tietoa, että miten React-Redux -arkkitehtuuria voitaisiin verrata perinteiseen MVC-arkkitehtuurin. Arkkitehtuurien tutkiminen toi ilmi, että tiettyä arkkitehtuuria käyttäessä tulisi asioita myös ajatella toisen arkkitehtuurin näkökulmasta. iOS-kehityksessä on olemassa muitakin käytettyjä arkkitehtuureja, joiden kokeileminen olisi tärkeä oman kehityksen näkökulmasta.

Lähteet

Sähköiset

Apple Inc. 2012. Model-View-Controller. Viitattu 24.3.2019. https://developer.apple.com/library/archive/documentation/General/Conceptual/CocoaEncyclopedia/Model-View-Controller/Model-View-Controller.html

Apple Inc. 2018. Start Developing iOS Apps (Swift). Viitattu: 15.5.2019. https://developer.apple.com/library/archive/referencelibrary/GettingStarted/DevelopiOSAppsSwift/BuildABasicUI.html#//apple_ref/doc/uid/TP40015214-CH5-SW1

Apple Inc. 2019. About Swift. Viitattu 7.4.2019. https://swift.org/about/#swiftorg-and-open-source

Apple Inc. 2019. UIKit. Viitattu 24.3.2019. https://developer.apple.com/documentation/uikit

Apple Inc. 2019. Xcode 10. Viitattu 7.4.2019. https://developer.apple.com/xcode/

Brooks, F. 1974. The Mythical Man-Month. Viitattu: 22.4.2019. http://ce.sharif.edu/courses/93-94/1/ce474-1/resources/root/theMythicalManMonth.pdf

Dawson, C. 2009. Projects in Computing and Information Systems. Viitattu: 22.4.2019. http://dinus.ac.id/repository/docs/ajar/Projects-in-Computing-and-Information-Systems-A-Student%E2%80%99s-Guide-2nd-Edition-April-2009.pdf

Facebook Inc. 2019. Tutorial: Intro to React. Viitattu 6.4.2019. https://reactjs.org/tutorial/tutorial.html#what-is-react

Gamma, E & Helm, R & Johnson, R & Vlissides, J. 1994. Design Patterns Elements of Reusable Object-Oriented Software. Viitattu 24.3.2019. https://sophia.javeriana.edu.co/~cbustaca/docencia/DSBP-2018-01/recursos/Erich%20Gamma,%20Richard%20Helm,%20Ralph%20Johnson,%20John%20M.%20Vlissides-Design%20Patterns_%20Elements%20of%20Reusable%20Object-Oriented%20Software%20%20-Addison-Wesley%20Professional%20%281994%29.pdf

Golafshani, N. 2003. Understanding Reliability and Validity in Qualitative Research. Viitattu 19.5.2019. https://nsuworks.nova.edu/cgi/viewcontent.cgi?referer=https://scholar.google.fi/&httpsredir=1&article=1870&context=tqr/

Google. 2019. Design. Viitattu: 22.5.2019. https://material.io/design/

Kerzner, H. 2017. Project management: a systems approach to planning, scheduling, and controlling. Viitattu: 22.4.2019. http://202.166.170.213:8080/xmlui/bitstream/handle/123456789/3849/Project%20management%20%20a%20systems%20approach%20to%20planning%2C%20scheduling%2C%20and%20controlling.pdf?sequence=1&isAllowed=y

Kolehmainen, A. 2019. Google kehottaa käyttämään kotlinia – Android-java sai naulan arkkuunsa. Viitattu 13.5.2019. https://www.tivi.fi/uutiset/google-kehottaa-kayttamaan-kotlinia-android-java-sai-naulan-arkkuunsa/3818e838-3eae-4ca6-a879-e17cae6d89c4

Kuprenko, V. 2018. Top 10 Swift Libraries from GitHub That Are Worth Using. Viitattu: 26.5.2019. https://www.codementor.io/kuprenkoauthor/top-10-swift-libraries-from-github-that-are-worth-using-k8g8gifph

Levkovsky, M. 2017. Thinking in Redux (when all you've known is MVC). Viitattu 24.3.2019. https://hackernoon.com/thinking-in-redux-when-all-youve-known-is-mvc-c78a74d35133

McNiff, J & Lomax, P &Whitehead, J. You and Your Action Research Project. Viitattu: 14.5.2019. http://213.55.83.214:8181/Education/27500.pdf

Microsoft, 2019. TypeScript. Viitattu 6.4.2019. https://www.typescriptlang.org/

Potter, J. 2019. Npm trends. Viitattu 6.4.2019. https://www.npmtrends.com/angular-vs-react-vs-vue

Reenskaug, T & Coplien, J. 2009. Guidelines for conducting and reporting case study research in software engineering. Viitattu 7.4.2019. https://www.artima.com/articles/dci_vision.html

Rising, L & Janoff, N. 2019. The Scrum Software Development Process for Small Teams. Viitattu 22.4.2019. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.544.6092&rep=rep1&type=pdf

Runeson, P & Höst, M. 2008. Viitattu: 14.5.2019. https://link.springer.com/article/10.1007/s10664-008-9102-8

Saring, J. 2018. State of React State Management for 2019. Viitattu 24.3.2019. https://blog.bitsrc.io/state-of-react-state-management-in-2019-779647206bbc

Tea, M. 2017. A Massive Guide to Building a RESTful API for Your Mobile App. Viitattu: 26.5.2019. https://savvyapps.com/blog/how-to-build-restful-api-mobile-app

Verheyen, G. 2014. Scrum is not an acronym. Viitattu 22.4.2019. https://gunthervergheyen.com/2014/01/09/scrum-is-not-an-acronym/

Vincit, 2019. Suomen suosituimmat mobiililaitteet Hesburgerin datan perusteella vuonna 2018. Viitattu 6.4.2019. https://www.vincit.fi/blog/suomen-suosituimmat-mobiililaitteet-hesburgerin-datan-perusteella-vuonna-2018/

Kuviot