Mobiilimaksusovelluksen suunnittelu lounasravintola BarLaurealle

Nico Hairetdin
Sallamaarit Jaako

2019 Laurea

Laurea-ammattikorkeakoulu

Mobiilimaksusovelluksen suunnittelu lounasravintola BarLaurealle

Nico Hairetdin
Sallamaarit Jaako
Tietojenkäsittelyn koulutusohjelma
Opinnäytetyö
Toukokuu, 2019

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

Nico Hairetdin, Sallamaarit Jaako

Mobiilimaksusovelluksen suunnittelu lounasravintola BarLaurealle

Vuosi 2019 Sivumäärä 49

Tämän opinnäytetyön tarkoituksena oli kehittää toteutustapa ravintolan lounaslippujen ostamisen ja käyttämisen mahdollistavalle mobiilisovellukselle. Sovelluksen käyttäjän tulee myös voida tarkastella ravintolan ruokalistaa. Työn toimeksiantaja on Laurea-ammattikorkeakoulun Leppävaaran toimipisteen lounasravintola BarLaurea. Opinnäytetyössä kuvataan ne keskeisimmät komponentit ja teknologiat, joiden avulla vaadittu toiminnallisuus voidaan saavuttaa. Työn tietoperusta pohjaa sähköisiin lähteisiin, eri teknologioiden ajantasaisiin teknisiin dokumentaatioihin sekä työn tekijöiden aiempaan osaamiseen.

Työssä hahmoteltiin mobiilisovelluksen toimintaa ja ulkonäköä rautalankamallien avulla ja kehitettiin suunnitelma järjestelmän eri osa-alueiden välisen tiedonsiirron toteuttamiselle Hypertext Transfer Protocol (HTTP) pyyntöjen avulla. Lisäksi työssä selvitettiin, kuinka React Native-sovelluskehyksen avulla kehitettyyn sovellukseen voidaan liittää maksamisen mahdollistava toiminnallisuus MobilePay-maksusovelluksen avulla ja kuinka MobilePayn tarjoaman rajapinnan kanssa kommunikoidaan Node.js-suoritusympäristössä. Mobiilisovelluksen tueksi suunniteltiin relaatiotietokanta ja Representational State Transfer (REST) rajapinnan reitit, joka mahdollistaa käyttäjien hallinnan ja lounaslippujen tallentamisen järjestelmään.

Asiasanat: Mobiilisovellus, React Native, MobilePay

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

Nico Hairetdin, Sallamaarit Jaako

Mobile payment application plan for BarLaurea

Year 2019 Pages 49

The aim of this bachelor's thesis was to develop a plan for a mobile application that provides a mechanism for users to purchase lunch tickets for the restaurant BarLaurea on Laurea Leppävaara campus. The users of the application must also be able to view the restaurant's weekly menu. This thesis described the most important components and technologies that are needed to achieve the abovementioned functionality. The theory is based on up-to-date online documentations of the chosen technologies and the authors' personal experience and knowledge.

A plan for the information exchange between the different components of the system using Hypertext Transfer Protocol (HTTP) requests was done and the functionality and the design of the mobile application was sketched by using wireframing. A mechanism for creating online payments in a React Native application by using MobilePay as a payment service provider was resolved. It was also found out how to communicate with the Application Programming Interface (API) of MobilePay on a server built with Node.js. A relational database and Representational State Transfer (REST) API routes were designed for saving and retrieving the tickets and user information.

Keywords: Mobile application, React Native, MobilePay

Sisällys

1 Johdanto

Nykypäivänä maailma on yhä voimakkaammin digitalisoitunut ja muuttunut enemmän mobilisoituvaan suuntaan. Muun muassa tästä syystä yritykset haluavat tarjota asiakkailleen helppoja ja toimivia mobiiliratkaisuja. Ratkaisujen odotetaan myös lisäävän yritysten liikevaihtoa.

Laurean Leppävaaran kampuksella toimivalla BarLaurea-lounasravintolalla on käytössään lounaslippujärjestelmä, joka toimii vanhanaikaisesti paperisilla lipuilla. Ravintola on jo pitkään halunnut sähköistää järjestelmän, muun muassa asiakaspalautteen takia, mutta sopivaa palveluntuottajaa ei ole löytynyt. Tästä syystä ravintola kääntyi meidän puoleemme ja kehitimme heille suunnitelman järjestelmän toteuttamiseksi.

Tämä opinnäytetyö esittelee perustellen yhdenlaisen toteutustavan kyseiselle palvelulle. Järjestelmän perimmäisenä tarkoituksena on tarjota ravintolan asiakkaille mahdollisuus ostaa lounaslippuja sekä tarkastella viikon ruokalistaa mobiilisovelluksen avulla. Lounaan lisäksi sovellus mahdollistaa lippujen myynnin myös muille tuotteille, kuten esimerkiksi kahville. Työ keskittyy järjestelmän eri osa-alueiden toteutukseen käytännön tasolla ja tarjoaa ratkaisun järjestelmän toimintalogiikan ja tiedonsiirron toteutukseen sen eri osa-alueiden välillä.

Järjestelmän laajuuden takia tämä opinnäytetyö ei sisällä sen täydellistä yksityiskohtaista toteutusta tai suunnitelmaa kaikilta osin. Sen sijaan työ keskittyy ongelmanratkaisuun yleisellä tasolla tarjoten käytännön esimerkkejä järjestelmän oleellisimpien ja välttämättömien komponenttien toteutuksesta. Työn pääpaino on kuitenkin mobiilisovelluksen ja sen taustakomponenttien toiminnassa.

Järjestelmäkokonaisuus rakentuu palvelimesta, tietokannasta, kassalla toimivasta QR-koodinlukijasta, puhelimella toimivasta mobiilisovelluksesta sekä maksupalveluntarjoajan tarjoamasta maksurajapinnasta. Työssä esitellään ratkaisuja näiden osioiden toteuttamiseksi sekä teoreettista taustaa ratkaisujen perusteiksi.

Teoreettisen taustan keräämiseksi käytimme erilaisten teknologioiden verkkodokumentaatioita. Lähestymistapana suunnitteluun käytimme NABC-mallia. Lisäarvoa saimme hiljaisesta tiedosta ja yhdessä oppimisesta.

2 Työn lähtökohdat ja toimeksiantaja

Projektin lähtökohtana oli löytää tapa toteuttaa mobiilimaksusovellus BarLaurea-lounasravintolalle. Opinnäytetyön toimeksiantaja on Laurea Leppävaaran kampuksella toimiva opetus- ja lounasravintola BarLaurea. Ravintolan asiakkaita ovat pääasiassa opiskelijat sekä talon henkilökunta. Ravintolassa syödään päivästä riippuen n. 200 – 600 ateriaa päivässä. Ravintolan asiakkailla on ennestään ollut mahdollisuus ostaa paperille painettuja lounaslippuja, joita vastaan lounaan voi lunastaa ravintolan kassalla. Lounaan voi maksaa myös esimerkiksi

pankkikortilla. Asiakaspalautteen takia ravintolalle on kuitenkin syntynyt tarve kehittää nykyaikaisempi tapa toteuttaa edellä mainittu lippujärjestelmä. Nykypäivänä luonteva tapa toteuttaa lippujärjestelmä on hyödyntää älypuhelinten mahdollisuuksia.

2.1 Projektin rajaukset ja tietoperusta

Projektin tietoperustana toimii aikaisempi JavaScript-ohjelmointikokemus ReactJS:n avulla, joka on syntaksiltaan saman kaltaista kuin React Native sekä SQL-tietokantaosaaminen. Käytimme myös hyödyksi aikaisemmin opittuja asioita mobiilikehittämisen kursseilta. Lisäksi käytimme React Nativen ja muiden kirjastojen verkkodokumentaatiota tietoperustana ja malliesimerkkeinä. Palvelimen puolella hyödynsimme myös aikaisempaa osaamistamme Node.js sekä Expressin parissa sekä käytimme verkkodokumentaatiota.

Järjestelmän toteutus ei onnistu yksin puhelinsovelluksella, vaan se vaatii toimiakseen myös palvelimen, tietokannan sekä kassalla toimivan lukulaitteen ohjelmistoineen lipun lukemista varten. Lisäksi BarLaurean henkilökunnan tulee pystyä muuttamaan tiettyjä erikseen määriteltyjä tietoja, kuten aterian hintaa. Tämä vaatii erillisen verkkoselaimella käytettävän hallintapaneelin toteutuksen.

2.2 Tutkimuskysymykset

Tutkimuskysymyksen tarkoitus on rajata tutkimusta. Tutkimuskysymys määrittelee mitä ja miten tutkitaan. Tutkimuskysymys myös antaa pohjaa näkökulmalle, mistä tutkimus tehdään. Tutkimuskysymystä voidaan selventää lisäkysymyksillä. (Näpärä 2017.)

BarLaurean toiveena oli mobiilimaksusovellus, mutta heillä ei ollut toteutustapaa sovellukselle. Tutkimuskysymyksinä on: kuinka kyseinen järjestelmä voitaisiin toteuttaa tehokkaasti mobiilisovelluksena? Mitä erilaisia teknologioita siihen tarvitaan? Kuinka maksaminen sovelluksen kautta voidaan toteuttaa?

2.3 Käsitteet

API: Application programming interface on ohjelmointirajapinta tiedonvälittämiseen järjestelmien välillä.

HTTP-protokolla: Asiakasohjelman ja palvelimen väliseen kommunikaatioon käytetty protokolla, joka perustuu pyyntöihin ja vastauksiin.

JSON: Avoimen standardin tiedostomuoto, lyhenne sanoista JavaScript Object Notation.

MerchantId: MobilePayn määrittelemä yksilöllinen tunniste kauppiaalle.

MobilePay API: Ohjelmointirajapinta MobilePay-maksujen hallinnointiin.

MobilePay App: Danske Bankin kehittämä mobiilimaksusovellus.

MobilePay SDK: Asiakassovelluksen ja MobilePay-sovelluksen välisen kommunikoinnin mahdollistava rajapinta.

NABC-malli: Kehitysmenetelmä, jota käytetään ideoiden esiin tuomisessa.

OrderId: Asiakasohjelman määrittämä maksukohtainen 4-50 merkkiä pitkä yksilöllinen tunniste MobilePay-maksulle.

QR-koodi: Quick Response Code, sisältää tietoa ohjelmallisesti luettavana kaksiulotteisena kuviokoodina.

ReactJS: Käyttöliittymien rakentamiseen suunniteltu JavaScript-kirjasto.

React Native: Sovelluskehys mobiilisovellusten rakentamiseen JavaScript-ohjelmointikielen avulla.

REST: Representational State Transfer on ohjelmointirajapinnan toteutukseen käytettävä malli.

Sekvenssikaavio: Komponenttien välistä vuorovaikutusta kuvaava kaavio. Kaavioissa aika kulkee ylhäältä alaspäin.

SQL: Relaatiotietokannan hallintaan käytettävä kyselykieli.

Token (JSON Web Token): Standardi, joka määrittelee turvallisen tavan siirtää tietoa JSON-muodossa digitaalisesti allekirjoitettuna.

TransactionId: MobilePayn määrittämä yksilöllinen tunniste maksun suoritukselle.

Väliohjelmisto (middleware): Ohjelmistokomponentti, jonka tarkoituksena on koneellisesti muokata ja käsitellä tietoa tiedonvälityksessä.

3 Mobiilisovellustyypit

Mobiilisovellustyyppejä on erilaisiin tarpeisiin ja niitä voidaan toteuttaa eri tavoilla. Sovelluksen suunnitteluvaiheessa mietitään mille alustalle sovellus halutaan toteuttaa ja kuinka paljon kehityskustannuksiin ollaan valmiita käyttämään rahaa. Mobiilisovellustyyppejä ovat Natiivisovellukset, Hybridi-Natiivi ja Hybridi-Web. (Hackernoon 2018.)

3.1 Natiivisovellukset

Natiivisovelluksella tarkoitetaan sovellusta, joka on tehty yksittäisen alustan käyttöön. Kaksi yleisintä alustaa ovat iOS ja Android. Android-sovelluksia kehitetään Android Studio -

ohjelmalla. Android sovellukset rakennetaan käyttäen Java ja Kotlin ohjelmointikieliä. iOS-sovelluksia kehitetään Xcodessa Objective C:llä tai Swiftillä. Jokaiselle alustalle on omat ohjeensa ja käytäntönsä, joita kehittäjien tulee noudattaa, sillä ne eroavat muun muassa ulkonäöltään ja toiminnoiltaan.

Hyötynä natiivisovelluksessa on sen hyvä suorituskyky ja nopeus, koska sovellus on optimoitu tietyn alustan käyttöön. Natiivisovelluksen käyttäjäkokemus voi olla parempi, kuin muiden sovellustyyppien, koska kaikki sovelluksen komponentit ja elementit luodaan käyttäjän alustaa varten. Natiivisovelluksen huonona puolena on hitaampi kehittäminen sekä korkeammat kehityskustannukset, sillä kehitystyöhön tarvitaan tietylle alustalle erikoistuneita ohjelmoijia. (Hackernoon 2018.)

3.2 Hybridi-Natiivi

Hybridi-Natiivi on sovellus, joka on kehitetty alkuperäisesti mobiilisovellukseksi, mutta toimii joka alustalla. Tarkoittaen sitä, että samalla ohjelmointikielellä saadaan aikaiseksi yksi tuote monen alustan käyttöön. Alustoja ovat muun muassa iOS, Android ja Windows. Ohjelmointikielinä käytetään HTML5, CSS ja JavaScriptiä.

Hybridi-Natiivin hyötynä on sen helppo ylläpitäminen, koska sovellusta pystytään helposti päivittämään uusille alustoille. Tämän vuoksi myös kehityskustannukset ovat pienemmät, sillä eri alustoille ei tarvitse erillisiä kehittäjiä. Huonona puolena Hybridi-Natiivissa on, että siinä saattaa esiintyä suorituskykyongelmia.

Sovelluksen testaaminen voi olla myös haastavampaa, koska kehittäjä testaa sovellusta sovelluskehyksessä eikä välttämättä huomioi käyttöjärjestelmän uusimpia vikoja. Käyttäjäkokemuksen huomioiminen voi olla hankalampaa, koska käyttäjät pitää huomioida monessa alustassa ja jokaisen alustan tyyli eroaa toisestaan. (Hackernoon 2018.)

3.3 Hybridi-Web

Responsive Web App (RWA, suom. Responsiivinen verkkosovellus) oli varhaisia Hybridi-Web muotoja. RWA:lla tarkoitetaan sitä, että tiettyjä verkkoelementtejä muuttamalla saadaan toiminnallisuus erilaisille laitetyypeille, eri kokoisille ruuduille sekä eri alustoille. RWA tarvitsee toimiakseen internetyhteyden ja sivujen lataaminen huonolla yhteydellä voi olla hitaampaa. (Agicent 2018.)

Progressive Web App:n (PWA, suom. Progressiivinen verkkosovellus) suosio on kasvanut nopeasti viime vuosina, koska se on toiminnallisuudeltaan ja ulkonäöltään saman tapainen, kuin natiivisovellukset. Toiminnaltaan PWA on melkein samanlainen kuin RWA, mutta se toimii myös yhteydettömässä tai hitaan yhteyden tilassa ja mahdollistaa ilmoitusten käyttämisen samalla tavalla kuin natiivisovelluksissa. (Agicent 2018.)

Muita PWA:n tarjoamia ominaisuuksia ovat muun muassa nopeat latausajat sekä mahdollisuus lisätä sovellus kotinäytölle. Sana progressiivinen tarkoittaakin sitä, että ominaisuuksia voidaan ottaa käyttöön porrastetusti tarpeen mukaan, eikä kaikkea tarvitse ottaa käyttöön heti. PWA:n hyötynä onkin sen helppo käyttöönotto, koska sovellusta ei tarvitse ladata Google Play- tai Applen App Storesta. PWA sovelluksen lataaminen tapahtuu automaattisesti taustalla, jonka lisäksi sivulatauksella myös hoituu sovelluksen päivitykset (Ionic 2018). On kuitenkin pian mahdollista julkaista PWA sovelluksia Google Play Storessa käyttäen TWA (Trusted Web Activities) teknologiaa. (Web 2019.)

PWA:n huonona puolena on se, että se ei pysty käyttämään laitteen koko laskentatehoa, jonka johdosta prosessointiajat ovat pidempiä kuin natiivisovelluksessa. Tämä johtuu siitä, että PWA käyttää selainta alustanaan. Jotkut vanhemmat selaimet eivät myöskään tue PWA:ta (Medium 2018). Taulukossa 1 vertaillaan eri mobiilisovellustyyppien ominaisuuksia.

	Native	Hybird-Native	Hybrid-Web
Examples	iOS and Android SDKs	React Native, Xamarin, NativeScript, Flutter	Ionic
Languages	Obj-C, Swift, Java	JS + Custom UI Language / Interpreter	HTML + CSS + JS
Code Reuse	Totally Separate Code Bases per Platform	Shared Business Logic with Different UI Codebases	One codebase, UI codebase stays the same
Target Platforms	iOS & Android Native Mobile Apps	iOS & Android Native Mobile Apps	iOS, Android, Electron, Mobile and Desktop Browsers as a Progressive Web App, and anywhere else the web runs
Investment	Largest investment in staff and time	Medium investment in staff and time	Lowest investment in staff and time
UI Elements	Native UI independent to each platform	A selection of Native UI elements for iOS and Android UI elements are specific to the target platform and not shared Custom UI elements begin to require split UI code bases	Web UI elements that are shared across any platform, conforming to the native look & feel of wherever they are deployed Easily use completely custom UI elements
API Access / Native Features	Separate Native API & Codebases for each App	Abstracted Single-Codebase Native Access through Plugins (with ability to write custom Plugins)	Abstracted Single-Codebase Native Access through Plugins (with ability to write custom Plugins)
Offline Access	Works	Works	Works
Performance	"Native Performance" with well written code.	Indistinguishable difference on modern devices with well written code. See above.	Indistinguishable difference on modern devices with well written code. See above.

Taulukko 1: Mobiilisovellustyypit (Ionic 2018)

4 React Native

Facebook on kehittänyt React Native -sovelluskehyksen vuonna 2015. Sovellustyypiltään se on Hybridi-Natiivi, jolla voidaan kehittää sovelluksia Android- ja iOS-alustoille. React Native on ominaisuuksiltaan saman kaltainen kuin React, joka on tarkoitettu rakentamaan web-sovelluksia. Kummatkin tukevat JSX syntaksia, joka on JavaScriptin ja Extensible Markup Language:n (XML) sekoitusta. Sekä Reactissa että React Nativessa käytetään komponentteja, komponenttien elämänkaaria (eng. lifecycle), komponentti kohtaisia tiloja sekä attribuuttidatoja (eng. props). React Native käyttää sovelluksen piirtämiseen (eng. rendering) <View> komponenttia, joka on verrattavissa HTML-elementtiin . React Nativessa käytetään valmiita moduuleita, joita voidaan lisätä itse sovellukseen. (O'reilly 2018.)

Ajattelimme React Nativen olevan sopiva tämän kehitystyön toteuttamiselle, koska sen suorituskyky ja muistin käyttö ovat lähes samalla tasolla, kuin natiivisovelluksilla. (Medium 2017.)

4.1 React Native komponenttien käyttö

Sovelluksen rakentamisessa tarvitaan useita eri React Nativen tarjoamia komponentteja. Peruskomponentteja ovat View, ScrollView, Text ja TouchableOpacity. View-komponentin tarkoituksena on saada näkymä sovellukselle ja komponentti toimii samalla tavalla kuin HTML:n div-elementti. Lisäksi View-komponentti tukee Flexbox-asettelua, jolla pystytään tekemään asetteluja eri kuvaruutuihin. Esimerkiksi tässä sovelluksessa halutaan näkymä ruokalistalle, jossa on käytetty Flexbox-asettelua ruokalistan keskittämiseksi. ScrollView on komponentti, jolla voidaan tehdä listoja. Tällä komponentilla voidaan korvata View-komponentti. Esimerkiksi tässä sovelluksessa voidaan ruokalista näyttää listamuotoisesti, jos jokaisella ruokalistan elementillä on oma yksilöllinen avain. Tämä on hyödyllistä, jos näkymä on pitkä ja tarvitsee vierittämisen. (Facebook 2019.)

Text-komponentin tarkoituksena on näyttää tekstiä sovelluksessa ja sitä voidaan tyylitellä eri tarpeisiin. Text-komponentti tarvitsee View- tai ScrollView-komponentin vanhemmakseen. Tässä sovelluksessa Text-komponenttia käytetään muun muassa ruokalistan otsikointiin.

TouchableOpacity-komponenttia käytetään, kun halutaan koskettaa tiettyä näkymää tai nappia. Esimerkkinä tässä sovelluksessa halutaan koskettaa ”buy” nappia, joka vie uuteen näkymään. Nappia koskettaessa TouchableOpacity antaa palautetta kosketuksesta vähentämällä läpinäkyvyyttä. React-Native tarjoaa muitakin Touchable-komponentteja, jos ei halua perinteistä nappia sovellukseen vaan haluaa rakentaa itse oman tyylitellyn Touchable-komponentin. (Facebook 2019.)

Modal-komponentin avulla pystytään näyttämään sisältöä näkymän päällä niin, että vanhaa näkymää ei tarvitse sulkea. Esimerkiksi BarLaurean asiakassovelluksen Liput-näkymässä näytetään listaus ostetuista lipuista. Kun sovelluksen käyttäjä haluaa avata lipun käyttöä varten, on lipun sisältämä QR-koodi kätevä esittää juuri Modal-komponentissa. (Facebook 2019.)

React Nativessa voidaan tehdä myös omia komponentteja, mutta yleensä nopeampaa on käyttää valmiiksi rakennettuja ja tyyliteltyjä komponentteja. Omien komponenttien tekemisessä on se etu, että niistä saadaan juuri sellaisia kuin halutaan. Komponenteista voidaan tehdä uudelleenkäytettäviä. Kuviossa 1 nähdään esimerkki uudelleenkäytettävästä CardSection komponentista, jota voitaisiin hyödyntää esimerkiksi Liput-näkymässä jakamaan lounasliput omiin korttiosioihin.

CardSection.js

```
import React, { Component } from "react";
import { View } from "react-native";

const CardSection = props => {
  return (
    <View style={[styles.containerStyle, props.style]}>{props.children}</View>
  );
};

const styles = {
  containerStyle: {
    borderBottomWidth: 1,
    padding: 5,
    backgroundColor: "#fff",
    justifyContent: "flex-start",
    flexDirection: "row",
    borderColor: "#ddd",
    position: "relative"
  }
};
export { CardSection };
```

Kuvio 1: Uudelleenkäytettävä CardSection-komponentti

4.2 Työkaluja React Nativen kehittämiseen

React Native-projekti voidaan luoda esimerkiksi React Native CLI:n tai Expo CLI:n avulla. Molemmat näistä ovat Node.js ympäristössä ajettavia komentoriviohjelmia. Jos kehitystyötä tehdään React Native CLI:n avulla, tarvitaan Node.js:n lisäksi myös Python2:n ohjelmakoodin kääntämiseen sekä Android Studion ohjelman suorituksen näyttämiseen. (Raywenderlich 2018.)

Android Studio on Android-käyttöliittymälle kehitettävien sovellusten kehitysalusta. Android Studio tarjoaa myös sisäänrakennetun emulaattorin, jolla pystytään jäljittelemään fyysisen mobiililaitteen toimintaa. Tällä emuloidulla laitteella voidaan kehityksen alla olevaa sovellusta havainnollistaa ja testata. Android Studio vaatii toimiakseen Java SDK:n järjestelmään asennettuna. (Search Mobile Computing 2018.)

5 Sovelluskehitys

Järjestelmän suunnittelussa voidaan käyttää myös ketterästä ohjelmistokehityksestä tuttuja käyttäjätarinoita. Sovelluksen suunnitellun toimintalogiikan ja ulkonäön näkemiseksi voidaan luoda rautalankamalleja.

5.1 Ketterä kehitys

Ketterää kehitystä käytetään yleensä ohjelmistoprojekteissa, koska sen avulla voidaan parantaa tiimityöskentelyä ja minimoida turhaa työtä. Ketterässä kehityksessä seurataan tietynlaisia periaatteita ja määritelmiä. Yleisempiä käytäntöjä ketterässä kehityksessä ovat muun muassa päiväpalaverit, jatkuva integraatio, sprintit sekä käyttäjätarinat. (itewiki 2018.)

Käyttäjätarinat ovat ketterän kehityksen käytäntö, jossa tehdään sovelluksen vaatimusmäärittely etukäteen. Aloitusvaiheessa vaatimusmäärittelyn ei tarvitse olla tarkka vaan projektin edetessä sitä voidaan tarkentaa. Käyttäjätarinoiden tarkoituksena on hahmottaa sovelluksen toiminnallisuutta siten, että niillä kuvataan kuka, mitä ja miksi tehdään. (Ohjelmistotuotanto 2018.)

5.2 Rautalankamalli

Rautalankamallilla aloitetaan mobiilisovelluksen käyttöliittymän ja toimintalogiikan hahmottaminen suunnittelun alkuvaiheessa, joten se voi sisältää viivoja ja laatikoita, eikä välttämättä ole visuaalisesti kovin tarkka. Tarkoituksena on hahmottaa sovelluksen sisältöä, käytettyjä elementtejä ja toiminnallisuutta. (FixUi 2018.)

Rautalankamallista hyötyvät sekä kehittäjät että suunnittelijat. Rautalankamallin esittäminen mahdolliselle asiakkaalle antaa myös hänelle kuvan kehitystiimin ideoista ja siten asiakas voi esittää toiveitaan suunnitelmalle. Rautalankamallit ovat yksinkertaisia ymmärtää, mutta ne sisältävät tarpeeksi monipuolista tietoa auttaakseen kaikkia osapuolia. (InVision 2018.)

6 Käyttöliittymää tukevat osat

Mobiilisovelluksien käyttöliittymät vaativat useita taustapalveluita. Sovelluksissa yleisimpiä tukijärjestelmiä ovat HTTP-palvelin ja tietokanta. Näiden lisäksi vaaditaan tapa siirtää tietoa kaikkien osien kesken.

6.1 Palvelimen suunnittelu

Node.js on avoimen lähdekoodin Javascript-suoritusympäristö, joka mahdollistaa JavaScript-ohjelmien suorittamisen useilla eri alustoilla ilman verkkoselainta. Yksi tyypillinen käyttötarkoitus Node.js:lle on HTTP-palvelimen toteutus. (Node.js 2019.)

HTTP-palvelin on ohjelmisto, joka käsittelee asiakasohjelmistojen avulla lähetettyjä HTTP-kutsuja. HTTP-palvelimen tyypillisin tehtävä on vastaanottaa verkkoselaimista tehtyjä kutsuja ja vastata niihin esimerkiksi kuvatiedostolla tai HTML-koodia sisältävällä tiedostolla. Palvelinohjelmisto on usein myös yhteydessä tietokantaan, johon tallennetaan pysyvää tietoa kuten käyttäjätietoja.

HTTP-palvelimen toteutukseen Node.js:n avulla on kehitetty Express-niminen sovelluskehys. Sen tarkoituksena on helpottaa HTTP-palvelimen toteutusta yksinkertaistamalla Node.js:n omaa rajapintaa. Se tekee Node.js:n verkkotoiminnallisuudesta helpommin lähestyttävän muun muassa saapuvien HTTP-pyyntöjen käsittelyn osalta. (Hahn 2016, 20-25.)

Yleensä Node.js:n tukena käytetään paketinhallintajärjestelmää Node package manager (npm), joka mahdollistaa kirjastojen ja sovelluskehysten asentamisen Node.js-projektiin helposti. Express voidaan asentaa npm-paketinhallinnan avulla.

6.2 Tietokanta ja SQL

Tietokannan määritelmiä on monenlaisia, mutta yksinkertaisimmillaan tietokannaksi voi kutsua mitä tahansa tietoa sisältävää kokoelmaa. Kun tietokannoista puhutaan tietokoneiden yhteydessä, tarkoitetaan yleensä ohjelmistoa, joka täyttää tiettyjä vaatimuksia. Tiedon tallentamisen tietokantaan sekä sieltä tiedon hakemisen tulee olla nopeaa ja helppoa. Tämän lisäksi tietokannan tulee olla luotettava niin, että tiedon säilymiseen voi luottaa. Sen tulee myös mahdollistaa tapa erotella haettava tieto kaikesta muusta tietokannan sisältävästä tiedosta, jota ei juuri sillä hetkellä tarvita. (Allen 2006, 5-16.)

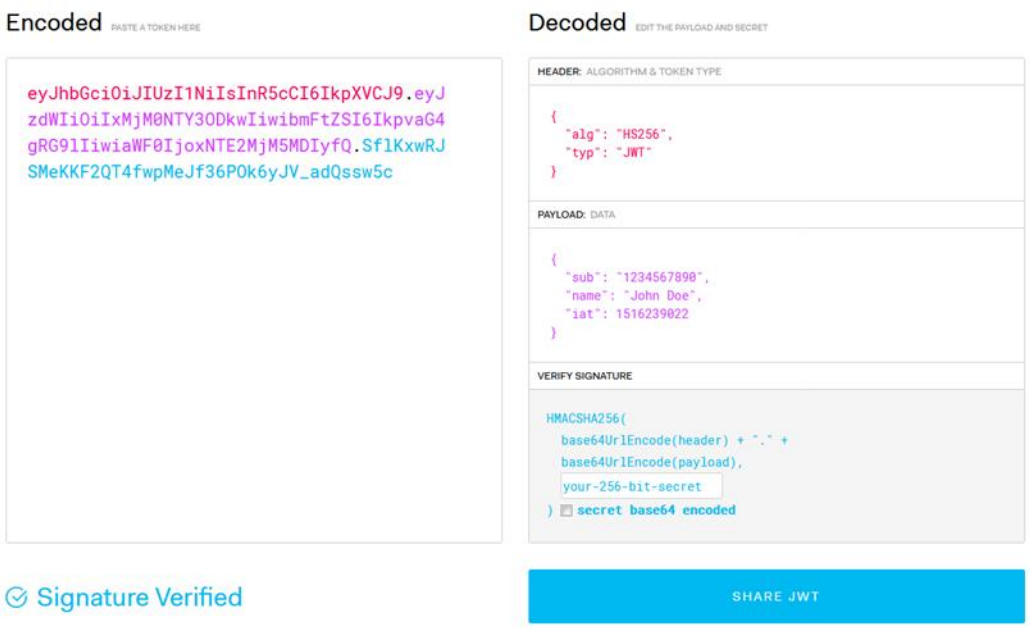
Yksi tapa toteuttaa edellä mainitut vaatimukset täyttävä tietokanta on relaatiotietokanta, jota hallitaan structured query language-kielellä (SQL). SQL on standardi, jota useat sen eri toteutukset pyrkivät noudattamaan. Verkkosovelluksien toteutuksissa suosittu MySQL on yksi SQL-standardia osittain noudattava avoimen lähdekoodin tietokantaohjelmisto joka toimii lähes kaikilla käyttöjärjestelmillä. (TechTarget 2018.)

6.3 Tiedonsiirto järjestelmän eri osien välillä

Ohjelmointirajapinta (eng. Application programming interface, API) mahdollistaa tiedon välittämisen järjestelmistä toisiin. Jotkin ohjelmointirajapinnat on tehty Representational state transfer (REST) ohjeistusta käyttäen. Tämä tarkoittaa sitä, että käytetään HTTP:ssä määritettyjä metodeja erilaisiin käyttötarkoituksiin. Esimerkiksi tiedon lähettämiseen käytetään HTTP pyyntöä POST ja tiedon poistamiseen HTTP DELETE. Tiedon hakeminen tapahtuu HTTP GET pyynnöllä palvelimelta ja tätä varten tarvitaan ohjelmointirajapinnan Uniform Resource identifier (URI) osoite. (REST API Tutorial 2019.)

JSON Web Token (JWT) standardin avulla pystytään todentamaan HTTP-pyynnön lähettäjä turvallisesti JSON-objektina. Kuviossa 2 nähdään esimerkki salatusta tokenista sekä sitä vastaavasta selkokielisestä sisällöstä. Käytämme tätä menetelmää käyttäjän oikeellisuuden tarkistamiseen. Kirjautumisen yhteydessä palvelin generoi ja lähettää käyttäjälle tokenin. Tämän jälkeen token lisätään käyttäjän jokaiseen HTTP-pyyntöön joko otsakkeisiin (eng. headers) tai runkoon (eng. body). Tokenin avulla palvelin tietää kuka pyynnön on tehnyt, jolloin oikean tiedon hakeminen tietokannasta onnistuu helposti ja luotettavasti. Esimerkiksi

käyttäjän tehdessä ostoksen, palvelin tietää kenelle ostos lisätään. Passport on Node.js:n väliohjelmisto (eng. middleware), jonka tarkoituksena on helposti todentaa ja purkaa token, jotta siitä saadaan tiedot käyttöön. Esimerkiksi jos palvelimelta halutaan nykyisen kirjautuneen käyttäjän tiedot, niin pyyntö lähetetään palvelimelle tokenin kera. Palvelin vastaanottaa pyynnön, purkaa sen mukana tulevan tokenin ja hakee tietokannasta tokenin omistajan tiedot ja palauttaa ne. (Medium 2017.)



Kuvio 2: Esimerkki JWT:stä vasemmalla salattuna ja oikealla purettuna. (JWT 2019)

6.4 Mobiilimaksaminen

Mobiilimaksaminen on laaja käsite, sillä se voi tarkoittaa mitä tahansa puhelimella tai muulla älylaitteella tapahtuvaa maksua, eikä sille ole olemassa mitään varsinaista virallista määritelmää. Esimerkiksi tavallinen verkkopankkimaksu puhelimella tehtynä voidaan käsittää mobiilimaksamiseksi. Teknisesti ottaen tällöin kyseessä on kuitenkin tavallinen, niin sanottu verkkoostos. Muita mobiilimaksamisen muotoja ovat laskujen maksaminen puhelimella, ostoksen maksaminen matkapuhelinliittymän saldolla tai rahansiirto mobiililompakoiden välillä, sekä maksaminen kassalla erillisen maksusovelluksen avulla. (Laitinen 2019.)

NFC-sirujen yleistyminen puhelimissa on mahdollistanut niin sanotun lähimaksamisen puhelimen avulla. Tässä maksutavassa puhelin viedään kassalla olevan lukulaitteen lähelle, jolloin maksutapahtuma perustuu kassajärjestelmän sekä puhelimessa olevan maksusovelluksen väliseen kommunikointiin NFC-tekniikan avulla. Maksu- ja lompakkosovellukset kuten MobilePay, Pivo, Aktia Wallet, Nordea Wallet, S-Pankin S-Mobiili, Google Pay ja Apple Pay tarjoavat maksamisen lisäksi myös muita ominaisuuksia, kuten mahdollisuuden tilisiirtoon sekä

maksutapahtumien ja tilitietojen tarkastelun. Edellä mainituista vaihtoehdoista ainakin MobilePay mahdollistaa kassalla maksamisen myös QR-koodin avulla tai vaihtoehtoisesti Bluetooth tekniikkaa hyödyntäen. (Danske Bank 2019.)

7 Tutkimus- ja kehittämismenetelmät

Tutkimusmenetelmien tarkoituksena on auttaa meitä rakentamaan laadullinen kehitystyö ravintola BarLaurean kanssa. Tätä kehitystyötä suunnitellessamme kävimme läpi erilaisia käytännön ongelmia sekä päätöksien tekoa esimerkiksi siitä mitä teknologiaa meidän tulisi käyttää projektissamme. Projektin alussa olemme kartoittaneet mahdolliset projektin rajaukset ja ottaneet huomioon asiakkaan toiveet. Tarkoituksenamme oli käyttää hyödyksi jo olemassa olevaa hiljaista tietoa sekä kerätä uutta tietoa kehitystyötämme varten.

7.1 Hiljaisen tiedon tutkimus ja tiedonkeräämismenetelmä

Aikaisemmilta projekteilta meille on kehittynyt niin sanottua hiljaista tietoa. Tämä tarkoittaa sitä, että meille on syntynyt aikaisempaa kokemuksellista tietoa, jota voidaan hyödyntää tässäkin projektissa. Kuitenkin tätä tietoa voi olla hankalaa määritellä, koska se voi olla henkilölle tiedostamatonta. (Pohjalainen 2012, 1-4.)

Hiljaista tietoa voidaan välittää ja omaksua myös jäljittelyn ja oppimisen kautta. Jäljittelyssä voimme katsoa mallia esimerkiksi jonkin toisen samankaltaisen ohjelmiston lähdekoodista. Opittavaa asiaa toistetaan, kunnes se alkaa tuntumaan rutiinilta. Tällöin tiedostetusta tiedosta tulee hiljaista tietoa. (Pohjalainen 2012, 1-4.)

Tiedonkeräämismenetelmä tarjoaa oivallisen avun ongelmanratkaisuun ja päätöksentekoon. On olemassa muutamia erilaisia tiedonkeräysmenetelmiä, kuten toissijainen tiedonkeruu, tapaustutkimukset, kokeilu, havainto, haastattelut ja tutkimukset sekä toimintatutkimus (Lancaster 2005, 65-77). Hyvä tiedonlähde on esimerkiksi internet ja teknologioiden dokumentaatiot.

7.2 Työskentely yhdessä ja muilta oppiminen

Yhdessä oppiminen tapahtuu siten, että pienryhmä ihmisiä tapaa ajoittain jutellakseen projektinsa ongelmista toisilleen. Ryhmää kutsutaan tavallisesti nimellä ”set” ja ryhmän mukana on yleensä valmentaja, joka vastaa oppimisen edistämisestä. Ryhmässä vuorotellen kuvataan oman projektin ongelma ja saadaan muilta ryhmäläisiltä ideoita ongelmanratkaisuun. Toiminnanoppimisen avulla pyritään löytämään ratkaisuja ongelmiin muiden avulla. (Lancaster 2005, 14.)

On tärkeää tavata ajoittain pienenä ryhmänä ongelmien ratkaisemiseksi. Nämä tapaamiset auttavat osallistujia selkeyttämään yhteistä kuvaa halutusta lopputuloksesta. Tapaamiset ovat

yleensä suurin oppimistilanne osallistujille. Tapaamisissa voi olla mukana myös mahdollinen asiakas, jolloin tehdystä työstä saa palautetta säännöllisin väliajoin.

7.3 NABC-malli kehittämismenetelmänä

NABC-malli on alun perin lähtöisin Stanfordin yliopistosta. Sitä käytetään apukeinona kehitystyössä, arvioinnissa ja ideoiden esiin tuomisessa. Mallin luojat halusivat esittää selkeämmän lähestymistavan innovaation kehittäjille, jotka halusivat tuoda ideansa esille arvioiden samalla sen toimivuutta (Niels Christ 2012). NABC-mallia apuna käyttäen pystytään kirkastamaan ideoita arvokkaammiksi ja ainutlaatuisemmiksi. Tämä tarkoittaa sitä, että jos meillä on ideana tehdä mobiilimaksusovellus, kuinka voisimme tehdä sovelluksestamme ainutlaatuisen verrattuna muihin olemassa oleviin vastaaviin sovelluksiin. NABC-mallin tarkoituksena on parantaa idean arvoa mallin neljällä vaiheella, jotka ovat:

- N – Needs - Tarkastellaan käyttäjien tarpeita kehitysidean perustana. Kuka olisi valmis käyttämään tuotetta/palvelua? Mitä käyttäjä tarvitsee?

- A – Approach - Lähestytään ja käsitellään käyttäjän tarpeita ajatuksien ja ideoiden kautta. Mikä olisi erikoinen lähestymistapa?

- B – Benefits – Minkälaisia etuja käyttäjä saa tästä kehitysideasta suhteessa muihin samankaltaisiin kehitysideoihin.

- C – Competition – Kilpailijat ja ongelmatilanteet kehitysidean rakentamisessa. Pyritään kehittämään kilpailijoiden ratkaisua parempi tuote/palvelu tutustumalla kilpailijoiden tuotteisiin ja palveluihin. (Innovation English 2018.)

7.4 Validiteetti ja reliabiliteetti

Tutkimusmenetelmää voidaan tarkastella validiteetilla sekä reliabiliteetilla. Validiteetilla pystytään tutkimusmenetelmän näkökulmasta ilmaisemaan, kuinka hyvin tutkimuksessa käytetyt menetelmät sopivat tutkimukseen ja kuinka hyvin se mittaa tutkittavaa tietoa. (Hiltunen 2009.)

Validiteetilla pyritään kertomaan kuinka paljon tutkimuksen tulokset vastaavat todellisuutta (Hiltunen 2009). Tässä työssä tutkimusmenetelmän validiteettia on vaikeaa määrittää, sillä tutkimuksen tulos (eli valitut mobiilisovelluksen kehitysstrategiat ja teknologiat) määräytyy lopulta sovelluksen kehittäjän mielipiteistä ja mieltymyksistä. Voimme kuitenkin sanoa, että olemme päätyneet hyvään ratkaisuun, jota tukevat useat lähteet ja käytänteet sekä asiakkaan tarpeet ja mielipiteet.

Validiteettia on mahdollista tarkastella monesta eri näkökannasta. Yksi näkökulma on loogi-nen validiteetti, jolla voidaan tarkastella tutkimuksen kokonaisuutta kriittisesti tutkijan

omasta syntyneestä käsityksestä (Hiltunen 2009). Tämän tutkimuksen tulos on validi nimenomaan loogisesti.

Reliabiliteetilla pystytään osoittamaan, että tutkimuksen tulos ei ole vain sattumanvaraista ja mittaustulokset olisivat myös toistettavissa. Reliabiliteettia voidaan tarkastella kahdella osatekijällä stabiliteetilla tai konsistenssilla. Stabiliteetilla arvioidaan ilmiön ajallinen pysyvyys, jolloin voidaan päätellä, ettei se ole sattumanvaraista. Konsistenssilla pystytään mittaamaan tuloksien yhtenäisyys eli, että tutkimuksen tulokset ovat johdonmukaisia. (Hiltunen 2009.)

8 Järjestelmän suunnittelu

Järjestelmän suunnittelussa hyödynsimme NABC-mallia ja pyrimme työskentelemään asiakaslähtöisesti. Suurimmassa osassa suunnittelua keskityimme löytämään helpon ja nopean maksutavan sovellukseen, koska vaatimuksena oli löytää yksinkertaisempi ratkaisu nykyiseen lippujärjestelmään. Lisäksi tässä luvussa pohditaan mahdollisia järjestelmän kriittisiä kohtia sekä virhetilanteiden käsittelyä. Kappaleessa käytetään teknologisiin käytännön esimerkkeihin valitsemaamme React Native-sovelluskehystä.

8.1 Käyttäjätarinat

Aloitimme sovelluksen suunnittelun luomalla käyttäjätarinoita (eng. User stories). Tarkoituksena oli työskennellä asiakaslähtöisesti sekä antaa asiakkaalle selkeä kuva sovelluksen toiminnallisuudesta. Halutessaan asiakas pystyi osallistumaan antamalla palautetta käyttäjätarinoista ja kertomaan toiveistaan.

Taulukossa 2 esittelemme käyttäjätarinoita, joita suunnittelimme BarLaurea sovellusta varten. Taulukossa on tärkeysjärjestyksessä lueteltuna käyttäjätarinoita, jotka kuvaavat sovelluksen ominaisuuksia. Kun käyttäjätarina otetaan työn alle, se jaotellaan edelleen pienemmiksi tehtäviksi. Näiden tehtävien kokonaisuus muodostaa valmiiksi kehitetyn käyttäjätarinan.

<	User story	Status	In Sprint	
			Started	Finished
1	As a user, I can see the menu	Not started		
2	As a user, I can register	Not started		
3	As a user, I can login	Not started		
4	As a user, I can identify as a student	Not started		
5	As an admin, I can create menu	Not started		
6	As an admin, I can remove menu	Not started		
7	As an admin, I can update menu	Not started		
8	As a logged user, I can buy BarLaurea food	Not started		

Taulukko 2: Käyttäjätarinat

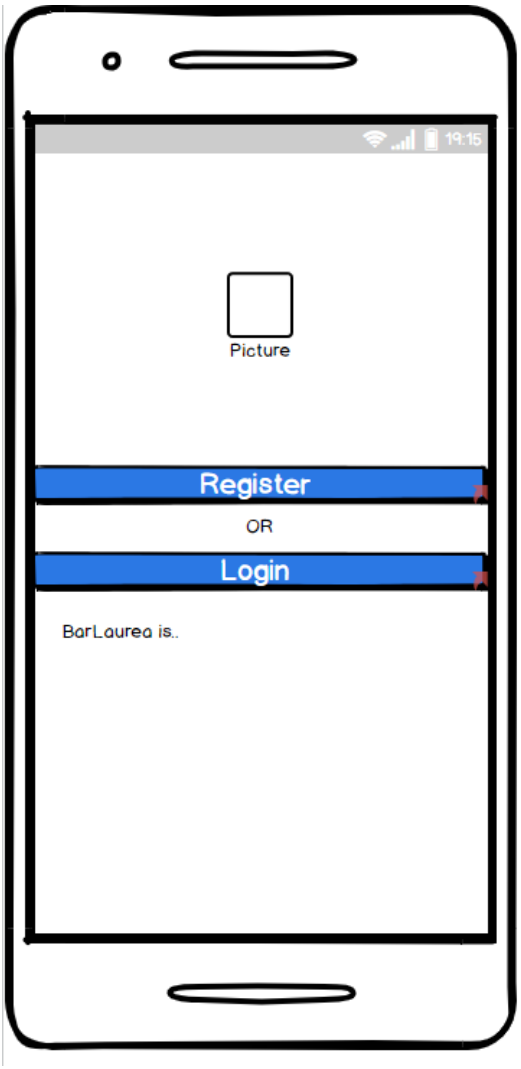
8.2 Rautalankamallin suunnittelu

Aloitimme luomalla oman rautalankamallimme ensin paperille, mutta tämän jälkeen siirryimme käyttämään Balsamiq-suunnittelutyökalua. Suunnittelutyökalu tarjoaa oivalliset mahdollisuudet rautalankamallin luomiseen.

Käynnistyskuvan tarkoituksena on näyttää käyttäjälle yrityksen kuva tai logo, sillä aikaa kun sovellus latautuu (kuvio 3). Sovelluksen ladattua käynnistyskuvalta siirrytään yleensä toiminnallisempaan näkymään. Jotta käyttäjä ei ärsyyntyisi odotusajasta, niin on suositeltavaa, ettei käynnistyskuva näkyisi liian pitkään (UX Planet 2017).

Kuvio 3: Käynnistyskuva

Käynnistyskuvan jälkeen käyttäjä tuodaan ”Tervetuloa”-näkymään, jossa käyttäjä voi rekisteröityä tai kirjautua sisään. Tässä näkymässä on yrityksen kuva ja tiedot, jotka näkyvät kuviossa 4. Sovelluksen ulkoasu tulee pohjautumaan enemmän Laurean visuaaliseen ilmeeseen, kun se toteutetaan. ”Picturen” tilalle tulisi BarLaurean logo ja taustalle heidän ravintolastaan kuva, joka on näkyvissä myös heidän kotisivuillaan. Rautalankamallin tekemisessä emme huomioineet visuaalista ilmettä sen tarkemmin, koska tämä on vasta raakaversio sovelluksesta.

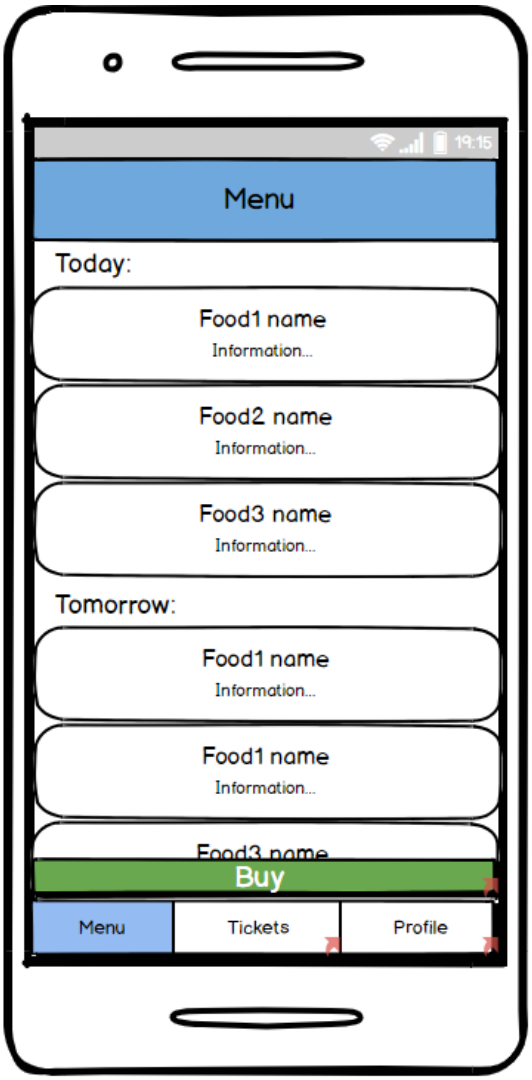


Kuvio 4: Tervetuloanäkymä

Käyttäjä pääsee näkemään ruokalistan sekä ostamaan lounaslippuja, kun hän on rekisteröitynyt ja kirjautunut sovellukseen. Rekisteröinti ja -kirjautumisnäkymät ovat ulkoasultaan melkein samanlaiset, mutta rekisteröintinäkymässä on salasanan varmennus täytettävänä tietona. (kuvio 5). Pyrimme pitämään molemmat näkymät selkeinä sisältäen vain tarvittavat täytettävät tiedot ja napit. Ulkoasun tulisi vaikuttaa yhtenäiseltä eri näkymien kesken ja olla mielekäs käyttää.

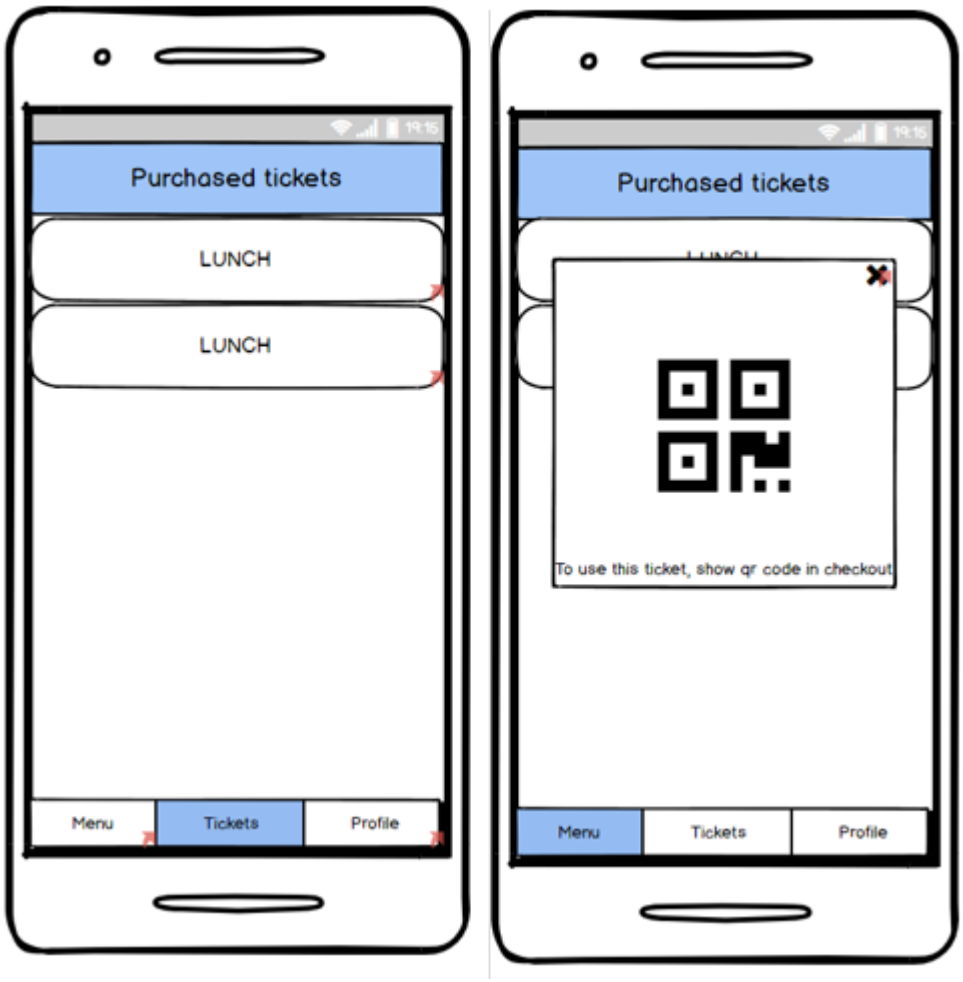
Kuvio 5: Rekisteröinti- ja kirjautumisnäkymät

Kirjautumisen jälkeen käyttäjä siirretään pääikkunaan missä näkyy ruokalista ja esitetään mahdollisuus ostaa lounaslippu painamalla ostonappia (kuvio 6). Sovelluksessa alhaalla sijaitsee navigaatiopalkki, josta pääsee liput- ja profiilinäkymiin. Siirtyminen eri näkymiin sovelluksessa tulee olla selkeää, jotta käyttäjä löytää helposti haluamansa. On tärkeää huomioida oikeanlainen nimeäminen navigaatiota tehdessä niin, että se on informatiivista käyttäjälle. Suositeltavaa on myös käyttää kuvakkeita, jotka kuvastavat navigoitavaa kohdetta. (Orbit Media Studios 2016.)



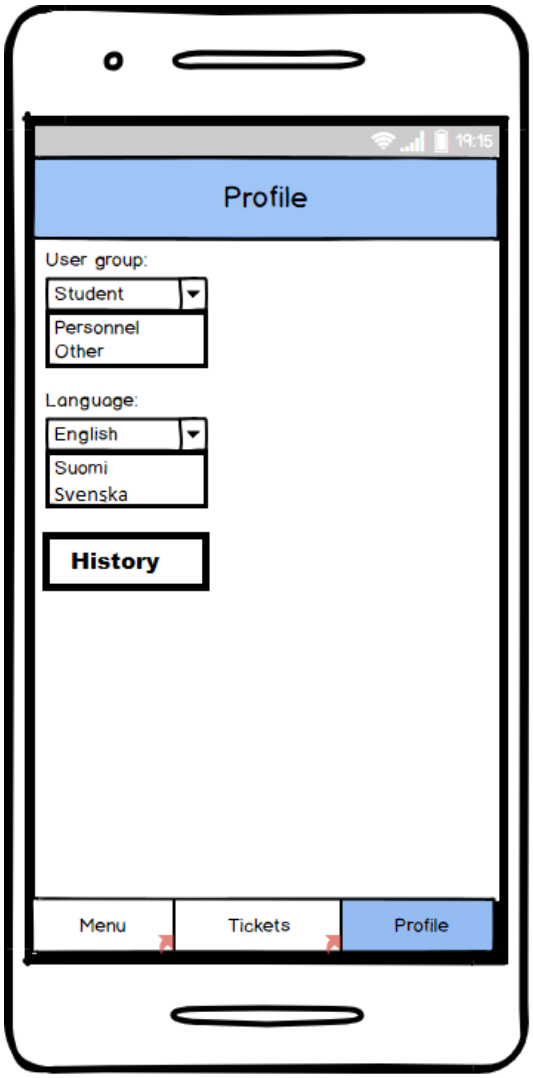
Kuvio 6: Ruokalistanäkymä

Liput-näkymässä käyttäjä näkee ostamansa liput koko ruudun levyisenä laatikkona ja lippua painamalla käyttäjälle avautuu QR-koodi, jolla voidaan lunastaa lounas BarLaurea-ravintolassa (kuvio 7). QR-koodia näytetään ostotilanteessa kassalla QR-koodinlukijalle. Näkymässä näkyvät vain käyttämättömät liput ja käytetyt liput näkyvät käyttäjäsivulla.



Kuvio 7: Liput-näkymä ja avautunut QR-koodi

Profiilissa käyttäjä voi valita mihin käyttäjäryhmään kuuluu ja sitä mukaan saada ravintolasta alennuksia (kuvio 8). Vaihtoehtoina ovat opiskelija, henkilökunta tai muu käyttäjä. Käyttäjäryhmän alapuolella on valintaikkuna kielen valitsemiseen ja nappi historianäkymään siirtymiseksi. Historianäkymä on identtinen lippunäkymän kanssa, mutta näyttää käytetyt liput.



Kuvio 8: Käyttäjän profiilinäkymä

8.3 Järjestelmän toimintalogiikka

Suunnittelemamme järjestelmä voidaan jakaa neljään eri sovellukseen ja rajapintaan. React-Nativen avulla toteutettava puhelinsovellus on varsinainen käyttöliittymä, jonka käyttäjä näkee. Se tarvitsee kuitenkin avukseen palvelimen, jonka avulla muun muassa käyttäjän tunnistaminen tapahtuu. Palvelimen lisäksi järjestelmä tarvitsee kassalla toimivan QR-koodin lukijalaitteen. Järjestelmän neljäs osa on MobilePay API (ohjelmointirajapinta), jonka kanssa järjestelmän muut sovellukset keskustelevat.

Lounaslipun ostaminen ja lounaslipun käyttäminen, eli lounaan lunastaminen kassalla ovat kaksi erillistä toimintoa. Tiedonsiirto järjestelmän eri komponenttien välillä on suunniteltu tehtäväksi HTTP-pyyntöinä ja vastauksina lukuunottamatta asiakassovelluksen ja MobilePay-sovelluksen välistä liikennettä, joka tapahtuu MobilePayn julkaiseman rajapinnan kautta puhelimessa.

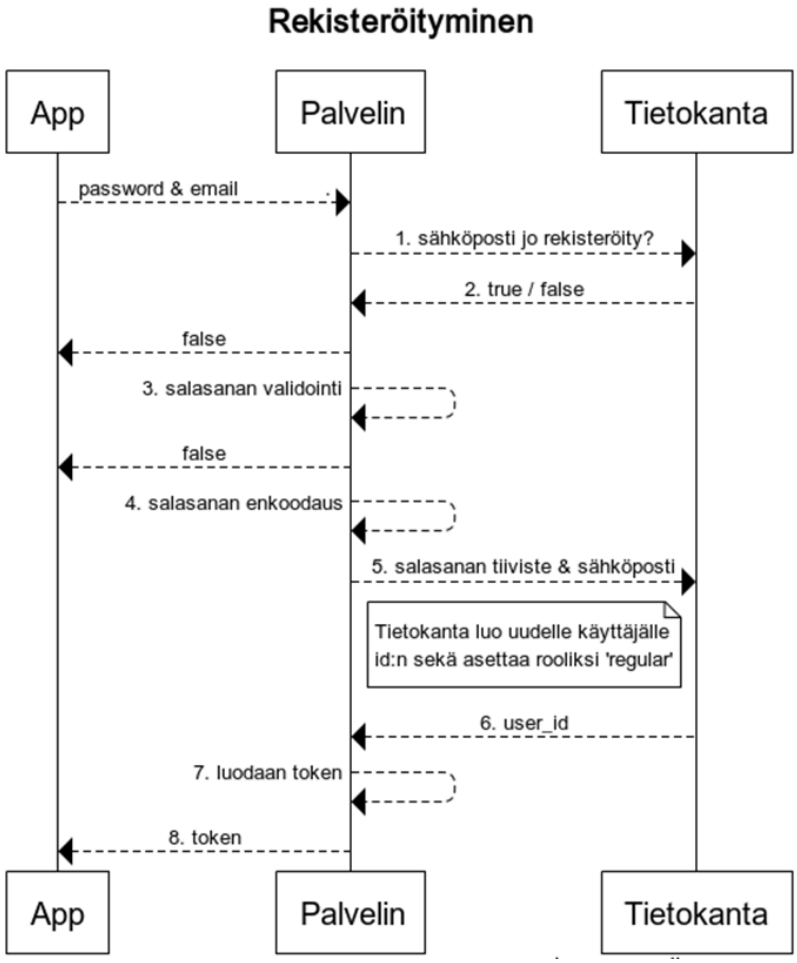
8.3.1 Rekisteröityminen ja kirjautuminen

Käyttäjän identiteetin varmentaminen toteutetaan token-perusteisena. Token-perusteisessa tunnistamisessa käyttäjää varten luodaan yksilöllinen token, jota vastaan käyttäjälle annetaan pääsy tiettyyn resurssiin palvelimella (The World Wide Web Consortium 2001). Käytännössä tämä tarkoittaa sitä, että käyttäjä on kirjautuneena silloin, kun hänellä on hallussaan kyseinen token. Token voi olla ikuisesti voimassa tai sille voi asettaa määrätyn voimassaoloajan.

Käyttäjä rekisteröityy sovellukseen antamalla sähköpostiosoitteen sekä salasanan asiakassovelluksessa (kaaviossa ”App”). Sähköpostiosoite ja salasana välitetään palvelimelle POST-tyyppisellä HTTP-pyynnöllä. Palvelin tarkastaa, onko sähköpostiosoite jo rekisteröity sekä validoi salasanan sen pituuden osalta. Salasanan on oltava vähintään 8 merkkiä pitkä merkkijono. Jos edellä mainitut tarkistukset epäonnistuvat, palvelin vastaa asianmukaisella virheilmoituksella.

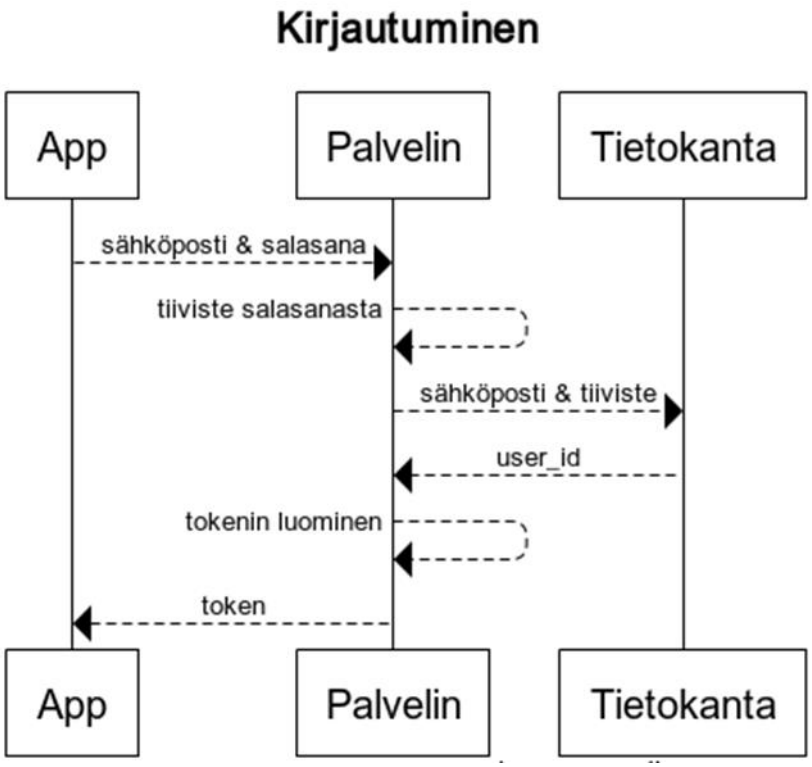
Kaaviossa on yksinkertaistamisen vuoksi käytetty ”false” ilmaisemaan virhetilannetta. Jos sähköpostia ei ole rekisteröity ja salasana on hyväksyttävä, palvelin luo salasanasta tiivisteen käyttäen bcrypt-kirjaston JavaScript-toteutusta. Salasanan tiiviste sekä sähköposti voidaan nyt tallentaa tietokannan ”user”-tauluun.

Tietokanta tallentaa uudelle käyttäjälle yksilöllisen id:n (user_id) sekä rooliksi oletusarvon ”regular”, jonka jälkeen se palauttaa juuri luodun käyttäjän id:n. Seuraavaksi palvelin luo tokenin käyttäen jsonwebtoken-kirjaston JavaScript-toteutusta ja palauttaa sen asiakassovellukselle. Token sisältää uuden käyttäjän sähköpostiosoitteen sekä user_id:n JSON-muodossa. Rekisteröityminen on kuvattu kuviossa 9.



Kuvio 9: Sekvenssikaavio rekisteröitymisen toiminnallisuudesta

Käyttäjä kirjautuu järjestelmään antamalla sähköpostiosoitteen sekä salasanan, jotka välitetään palvelimelle POST-tyyppisenä HTTP-pyyntönä (kuvio 10). Palvelin generoi salasanasta tiivisteen käyttäen bcrypt-funktiota, jonka jälkeen suorittaa kyselyn tietokantaan tiivisteen sekä sähköpostin perusteella. Jos tietokannasta löytyy vastaava sähköposti-tiiviste-yhdistelmä, tarkoittaa se sitä, että käyttäjä on tunnistettu ja käyttäjälle voidaan luoda token, joka luovutetaan käyttäjälle alkuperäisen HTTP-pyynnön vastauksessa.

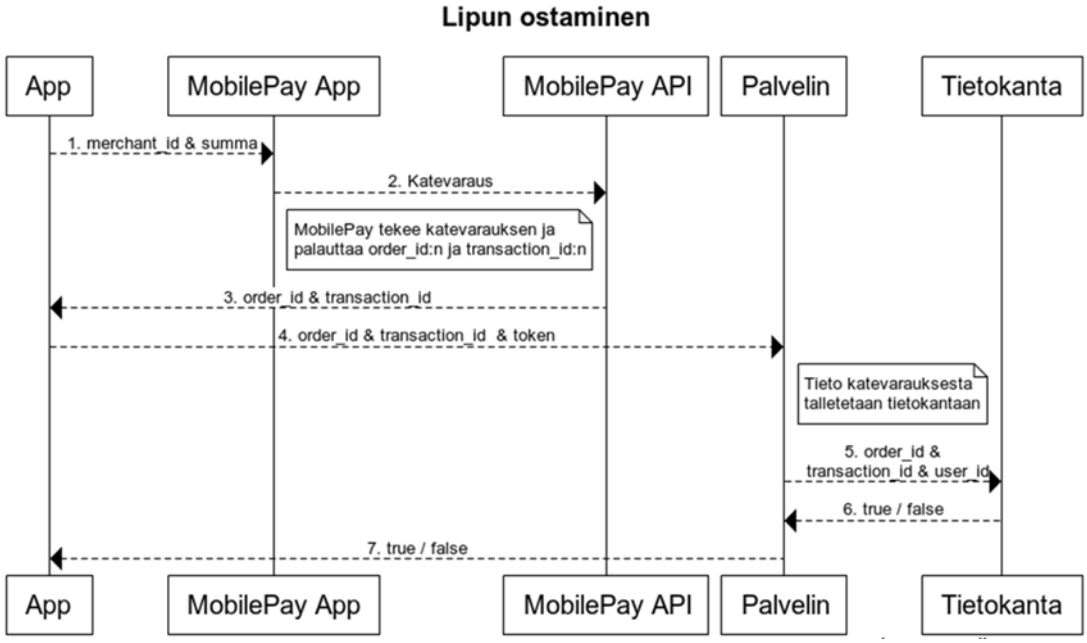


Kuvio 10: Sekvenssikaavio kirjautumisen toiminnallisuudesta

8.3.2 Lipun ostaminen

Kuviossa 11 nähtävä lipun ostaminen on tapahtuma, joka käynnistyy käyttäjän painaessa maksunappia asiakassovelluksessa, jolloin sovellus kutsuu MobilePay-sovellusta ja antaa kutsun parametreiksi merchantId:n sekä maksettavan summan sentteinä. MobilePay-sovellus tekee katevarauksen MobilePay API:n avulla, jonka jälkeen se palauttaa takaisin order_id:n sekä transaction_id:n, jotka ohjataan takaisin asiakassovellukseen.

Vastauksen saavuttua asiakassovellukseen, on aika tallentaa tieto ostoksesta tietokantaan. Tämä tehdään POST-tyyppisellä HTTP-pyynnöllä palvelimelle. Pyyntöön liitetään mukaan transaction_id, order_id sekä käyttäjän yksilöllinen token. Palvelin purkaa tokenin ja tallentaa sen sisältämän user_id:n sekä pyynnön mukana olevan transaction_id:n sekä order_id:n tietokannan 'ticket'-tauluun. Tämän lisäksi rivin statustietueen arvoksi asetetaan 'reserved'. Tämä tarkoittaa sitä, että katevaraus on tehty, mutta lippu on käyttämätön. Tieto onnistuneesta ostoksesta palautetaan takaisin asiakassovellukseen.

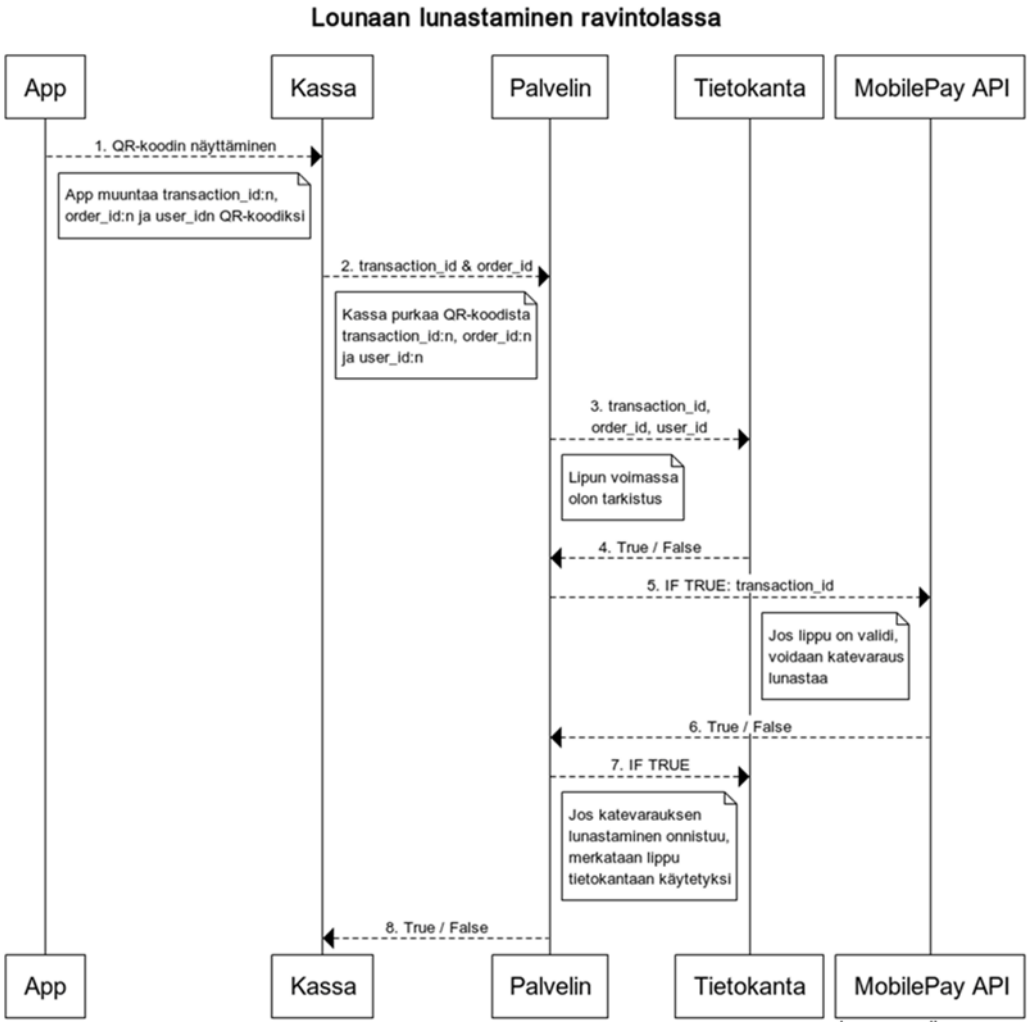


Kuvio 11: Sekvenssikaavio lipun ostamisesta

8.3.3 Lounaan lunastaminen

Kuviossa 12 on kuvattu lounaan lunastaminen ravintolassa. Asiakassovelluksen muistissa lippu on JSON-tyyppinen tietorakenne, jonka sisältönä on transaction_id, order_id, sekä käyttäjän yksilöllinen user_id.

Käyttäjä avaa lipun sovelluksessa Liput-välilehdellä sijaitsevasta listasta, jossa näkyvät käyttäjän käyttämättömät liput. Sovellus luo QR-koodin kyseisestä JSON-objektista. Lounaan lunastaminen tapahtuu näyttämällä lippua kassalla olevalle QR-koodilukijalle. Kassalla oleva lukija purkaa QR-koodin ja tekee POST-tyyppisen HTTP-pyynnön palvelimelle, jonka sisällöksi tulee QR-koodin sisältämä JSON-objekti.



Kuvio 12: Sekvenssikaavio lounaan lunastamisesta ravintolassa

Toinen tärkeä komponentti maksusovelluksessa on QR-koodi, joka luodaan merkkijonosta. Merkkijonon sisällön muodostavat maksukohtainen order_id ja transaction_id sekä käyttäjäkohtainen user_id yhdistettynä yhdeksi merkkijonoksi. Merkkijonon eri osat voidaan erotella kaksoispisteellä (kuvio 13).

```
const qrcode = `${order_id}:${transaction_id}:${user_id}`
```

Kuvio 13: QR-koodin merkkijono

Palvelin tarkastaa pyynnön lähettäjän identiteetin purkamalla pyynnön mukaan sisällytetyn tokenin. Jos token puretaan onnistuneesti, voidaan lipun voimassaolo tarkistaa hakemalla 'ticket'-taulusta kyseistä lippua vastaava rivi. Lipun statuksen ollessa 'reserved', on lippu käyttämätön, joten katevaraus voidaan lunastaa MobilePay API:n avulla.

Katevarauksen lunastaminen tehdään palvelimen suorittamana PUT-tyyppisenä HTTP-pyyntönä MobilePay API:n endpointtiin /api/v1/reservations/merchants/{merchantId}/orders/{orderId}. Pyynnön sisällöksi liitetään lunastettava summa sentteinä. MobilePay API vastaa onnistuneeseen pyyntöön transaction_id:llä ja epäonnistuneeseen pyyntöön virheilmoituksella (GitHub 2017.). Jos varauksen lunastaminen onnistuu, asetetaan kyseisen rivin statukseksi 'captured', jonka jälkeen palvelin vastaa koodinlukijalle onnistuneesta ostoksesta. Jos vastauksena on virheilmoitus, edelleenohjataan kyseinen virheilmoitus takaisin koodinlukijalle.

8.3.4 Ruokalistojen hakeminen

Tällä hetkellä BarLaurea käyttää Jamix-tuotannonohjausjärjestelmää, johon päivitetään ruokalistoja sekä annoksien ravintoarvotietoja. Jamix tarjoaa myös palvelun, mistä ruokalistoja ja muita tietoja voidaan välittää toisiin sivustoihin. Tarkoituksena on hakea Jamixista tarvittavat ruokalistatiedot sovellukseen käyttämällä HTTP-pyyntöä GET. Tämän jälkeen ruokalistanäkymä voidaan päivittää uusilla tiedoilla.

Tulevaisuudessa on mahdollisuus helposti vaihtaa ruokalistoja tarjoavaa palvelua tai kehittää oma toteutus. Tämän mahdollistaa API-tyylinen ruokalistojen hakeminen, sillä mahdollisen uuden järjestelmän käyttöönottaminen ei vaadi muutoksia mobiilisovellukseen.

Axios on JavaScriptin promise-objekteihin perustuva kirjasto, joka helpottaa HTTP-pyyntöjen kanssa työskentelyä. Kuviossa 14 nähdään esimerkki GET-tyyppisestä HTTP-pyynnöstä käyttäen axios-kirjastoa.

```
axios.get('/user?ID=12345')
  .then(function (response) {
    console.log(response);
  })
  .catch(function (error) {
    console.log(error);
  });
```

Kuvio 14: Esimerkki GET pyynnöstä käyttäen axiosia.

8.4 Mobiilimaksamisen toteuttamisen suunnittelu

Mobiilimaksamisen ensimmäinen edellytys oli helppous. Lisäksi maksun käsittelyn tuli olla turvallista ja mahdollista peruuttaa. Maksutavan sekä maksupalveluntarjoajan valinta tapahtui vertailemalla alan eri toimijoiden palveluita sekä sovelluksia. Opinnäytetyön toimeksiantajan toiveena on sovellus, jonka avulla asiakkaat voivat ostaa lounaslippuja etukäteen jo ennen

kassalle saapumista. Lähimaksaminen ei siis tule tässä tapauksessa kyseeseen, kun otetaan huomioon edellä mainittu vaatimus.

Yksi mahdollinen toteutustapa maksamiselle on perinteinen maksu verkkopankin kautta. Tätä vaihtoehtoa puoltaa se, että pääosin nuoresta asiakaskunnasta lähes kaikilla on pääsy verkkopankkiin, sekä se, että sitä voidaan pitää varsin turvallisena maksutapana (Yrittäjät 2017). Tämä on kuitenkin asiakkaan näkökulmasta suhteellisen hankalakäyttöinen maksutapa, kun ottaa huomioon, että maksettava summa on pieni (2,60€) ja maksu tapahtuu usein, mahdollisesti päivittäin.

8.4.1 Maksutavan valinta

Otettuamme huomioon toteutettavuuden helppouden, maksamisen kustannukset ravintolalle, sekä turvallisuuteen liittyvät seikat, suunnittelimme maksamisen toteutettavaksi MobilePayn AppSwitch-tekniikalla. MobilePayn AppSwitch-tekniikka mahdollistaa helpon ja turvallisen tavan toteuttaa maksaminen sovelluksessamme. Koska varsinainen maksuliikenne on MobilePayn vastuulla, tarkoittaa tämä muun muassa sitä, että asiakassovelluksen ei tarvitse tallentaa esimerkiksi asiakkaiden luottokorttitietoja itse. Toisaalta asiakassovelluksen ei myöskään tarvitse huolehtia maksamiseen liittyvistä teknisistä yksityiskohdista, koska kommunikaatio sovelluksen ja MobilePay-sovelluksen välillä tapahtuu MobilePayn kehittämän liitännäisen avulla. (MobilePay 2019.)

Sovelluksemme käyttäjälle tämä tarkoittaa sitä, että hänellä on oltava MobilePay asennettuna ja käyttöön otettuna omassa puhelimessaan. Maksaminen tässä tapauksessa toimii niin, että käyttäjä painaa sovelluksestamme maksunappia, jolloin käyttäjä siirtyy automaattisesti MobilePay-sovellukseen, jossa varsinainen maksaminen tapahtuu. Maksamisen jälkeen käyttäjä siirtyy automaattisesti takaisin omaan sovellukseemme. (MobilePay 2019.)

8.4.2 Katevarauksen tekeminen MobilePay SDK:n avulla

BarLaurean maksusovelluksen perustoiminnallisuus kuten napit, näkymät, tekstit ja tekstikentät voidaan rakentaa React Nativesta löytyvien peruskomponenttien avulla, mutta tarvitsemme myös toiminnallisuutta, jota varten täytyy asentaa moduuleita React Nativen ulkopuolelta. Toinen näistä on react-native-mobilepay, joka mahdollistaa MobilePay SDK:n kanssa kommunikoinnin React Native sovelluksessa. MobilePay SDK puolestaan mahdollistaa maksujen tai katevarausten tekemisen MobilePay-sovelluksen kautta niin, että käyttäjä siirtyy maksun ajaksi MobilePay sovellukseen (GitHub 2018). MobilePay SDK:ta voidaan käyttää React Nativessa seuraavasti: asennetaan ensin react-native-mobilepay komennolla npm install react-native-mobilepay –save. Tämä lataa tarvittavat riippuvuudet sekä ottaa sen käyttöön Node.js-projektissa lisäämällä tiedon konfiguraatiotiedostoon 'package.json'. Asentamisen lisäksi se täytyy vielä linkittää React Native-sovellukseen komennolla react-native link react-native-

mobilepay (GitHub 2018). Nyt asennettu moduuli voidaan ottaa koodissa käyttöön sisällyttämällä se import-avainsanan avulla (kuvio 15).

```
import MobilePay from 'react-native-mobilepay';
```

Kuvio 15: MobilePay import

Ennen maksun tai katevarauksen suoritusta täytyy MobilePaylle antaa tiettyjä asetustietoja. merchantId on MobilePayn kauppiaalle myöntämä kauppiaskohtainen tunnus, sen voi tarkistaa kirjautumalla MobilePay developer portaaliin. Se tarvitaan, jotta maksu menee oikealle yritykselle. Tunnuksen lisäksi MobilePaylle kerrotaan maa, jossa operoidaan sekä maksun tyyppi. Maksun tyyppi on tässä tapauksessa katevaraus. merchantUrlScheme voidaan Androidilla kehitettäessä jättää tyhjäksi merkkijonoksi. Esimerkkiasetusten määrittely ja käyttöönotto nähdään kuviossa 16.

```
const merchantId = 'APPFI0000000000'
const country = MobilePay.COUNTRY_FINLAND
const merchantUrlScheme = ''

MobilePay.setup(merchantId, country, merchantUrlScheme)
MobilePay.setCaptureType(MobilePay.CAPTURE_TYPE_RESERVE)
```

Kuvio 16: MobilePay asetukset

Nyt katevaraus voidaan suorittaa kutsumalla MobilePay-moduulin funktiota 'createPayment' ja antamalle sille parametreiksi yksilöllisen orderId:n sekä summan sentteinä. Esimerkki katevarauksen suorittavasta funktiosta kuviossa 17.

```
createPayment = () => {
  let orderId = 'order-1234'
  let amount = 260

  MobilePay.createPayment(orderId, amount).then(result => {
    if (!result.isCancelled) {
      console.log(`Success. ${result.orderId}${result.transactionId}`)
    } else {
      console.log("Cancelled.")
    }
  }).catch(error => {
    console.log("Failed")
  })
}
```

Kuvio 17: Maksun tekeminen

8.5 Järjestelmän kriittiset kohdat ja virhetilanteiden käsittely

Järjestelmä rakentuu neljän erillisen ohjelmiston ja rajapinnan välisen kommunikaation varaan käyttäen langatonta tiedonsiirtoa, joten on syytä varautua mahdollisiin virhetilanteisiin lippua ostettaessa sekä lippua käytettäessä. Ilmeisimpiä virhetilanteita ovat yhteyden katkeaminen väärällä hetkellä, esimerkiksi lippua ostettaessa. Virhetilanteiden oikeanlainen käsittely on välttämätöntä järjestelmän toiminnan kannalta.

Asiakas on maksanut tuotteesta, joten on varmistuttava siitä, että ei pääse syntymään tilannetta, jossa asiakas ei saa vastinetta rahalleen. On siis voitava olla varma, että riippumatta virheen syystä, asiakas saa aina joko rahansa takaisin tai vaihtoehtoisesti lunastettua aterian. Käytännössä tämä ratkaistaan niin, että katevaraus lunastetaan vasta siinä vaiheessa, kun asiakas näyttää lippua koodinlukijalle ravintolan kassalla. Tämä on se tilanne, jossa lippu muuttuu tietokannassa 'reserved' tilasta 'captured' tilaan katevarauksen lunastuksen ollessa onnistunut. Jos katevarausta ei lunasteta 14 vuorokauden sisällä varaushetkestä, varaus puretaan automaattisesti MobilePayn toimesta ja käytetty raha on jälleen asiakkaan uudelleenkäytettävissä. Näin ollen asiakas saa aina rahansa takaisin, jos lipun käyttäminen ei onnistu. (MobilePay 2019.)

Yksi mahdollinen virhetilanne on, jos asiakkaan yhteys katkeaa lipun ostamisen jälkeen ennen kuin lippua on ehditty tallentaa palvelimella sijaitsevaan tietokantaan. Tilanne on ongelmallinen, koska katevaraus on tehty, mutta palvelin ei ole tietoinen kyseisen lipun olemassaolosta. Tämän vuoksi asiakassovelluksen on saatava palvelimelta varmistus onnistuneesta talletuksesta ennen kuin asiakkaalle näytetään hänen ostamansa lippu. Käytännössä tämä voidaan ratkaista siten, että asiakassovelluksen muistissa lipun tietoihin lisätään boolean-tyyppinen tietue (true/false). Asiakassovellus näyttää ainoastaan liput, joiden talletus tietokantaan on varmistettu (true). Tämän takia varmistamattomat liput pystytään nyt tunnistamaan ja niiden tallentamista tietokantaan voidaan yrittää myöhemmin uudestaan. Tämä voidaan tehdä joko ajastuksella tai niin, että talletusta yritetään aina sovelluksen uudelleen käynnistyessä.

Virhetilanne ei aina ole tekninen tai ohjelmistosta johtuva. Virhetilanne voi syntyä myös käyttäjän toiminnan seurauksena. Esimerkiksi kun käyttäjä maksaa ostoksen MobilePay-sovelluksessa, mutta poistuu kotinäkymään ennen MobilePayn vastausta, tapahtuu virhetilanne. Edellä kuvattu toiminta aiheuttaa tilanteen, jossa palvelin ei saa koskaan tietoa ostoksen suorituksesta, eikä lippu ole näin ollen asiakkaan käytettävissä. Tässä tapauksessa asiakas saa rahansa takaisin 14 vuorokauden kuluttua MobilePayn automaattisen palautuskäytännön mukaisesti, eikä lippua koskaan käytetä. (MobilePay 2019.)

9 Käytettävät teknologiat ja työkalut

Mobiilisovelluksen kehittämiseksi käytetyn React Nativen lisäksi tarvitsimme muitakin teknologioita järjestelmän kokonaisuuden toteuttamiseksi. Palvelimen puolella päädyimme

Express/Node.js teknologiaan ja JSON Web Token Passport -todentamiseen. Lisäksi tässä luvussa kerromme myös tietokannan suunnittelusta käyttäen SQL-tietokantaa.

9.1 Palvelimen rakentaminen

Sovelluksen ja tietokannan välille tarvitaan palvelin, joka hoitaa tietokantakyselyiden luomisen sovelluksen HTTP-pyyntöjen perusteella. Palvelin luodaan toteuttamaan REST-rajapinta, käyttäen Express-kehystä Node.js ympäristössä. Selkeyden vuoksi palvelin toteutetaan kahdessa eri osassa, reititys ja tietokantayhteys. Sovellus on yhteydessä palvelimen reitittimeen, joka välittää tarvittavat tiedot tietokantayhteyttä ylläpitävään osaan tietokantakyselyiden suorittamiseksi.

Express on suosittu sovelluskehys Node.js:lle, jonka avulla voidaan helposti luoda rajapinnan toteuttava palvelin. Expressin avulla luodaan reititin, joka kuuntelee määritetyn portin eri reittejä. Asiakassovellus tekee HTTP-pyyntöjä näihin reitteihin, ja reitin sekä pyynnön tyypin mukaan pystytään määrittämään mitä palvelimen tulee suorittaa. Liitteessä 1 esimerkkejä reiteistä ja niiden toiminnoista.

9.1.1 Palvelinohjelmiston asennus

Node.js on Javascript suoritusympäristö, joka mahdollistaa JavaScript-ohjelmien suorittamisen useilla eri alustoilla. Yleensä Noden tukena käytetään npm-paketinhallintajärjestelmää (node package manager), joka mahdollistaa kirjastojen yhdistämisen sovellukseen helposti.

Node.js suoritusympäristössä toimivalle sovellukselle voidaan asentaa ylimääräisiä moduuleja paketinhallinnan avulla. Nämä moduulit eivät ole Node.js:n sisäänrakennettuja ominaisuuksia. Riippumatta siitä, ovatko moduulit Node.js:n sisäänrakennettuja ominaisuuksia vai paketinhallinnan kautta asennettuja, ne voidaan ottaa käyttöön saman 'require' avainsanan avulla. Käytämme npm-paketinhallintaa. Saavuttaaksemme halutun toiminnan palvelimen osalta mahdollisimman helposti ja luotettavasti, käytämme muutamia ylimääräisiä moduuleja, joiden tehtävänä on muun muassa rakentaa tietokantayhteys sekä suoraviivaistaa HTTP-pyyntöjen käsittelyä.

- **bcryptjs** tarjoaa salasanan tiivistealgoritmin, jota käytetään salasanan suojaamiseen. Emme halua tallentaa käyttäjän salasanaa selkokielisenä tietokantaan.

- **express** on HTTP-palvelimen toteuttava kirjasto, jonka avulla on helppo luoda esimerkiksi REST-rajapinnan mukainen toiminnallisuus.

- **jsonwebtoken** on JSON-pohjainen menetelmä yksityisellä avaimella suojatun käyttöoikeustokenin hallinnoimiseen.

- **mysql2** MySQL-protokollan toteuttava asiakasohjelma. Sen avulla voidaan luoda yhteys tietokantaan ja suorittaa SQL-kyselyitä.

Node.js projekti voidaan aloittaa luomalla aloituskonfiguraatio komennolla npm init. Tämän jälkeen tarvitsemamme riippuvuudet voidaan asentaa npm-paketinhallintatyökalua käyttäen komentoa npm install --save bcryptjs express jsonwebtoken mysql2.

9.1.2 MobilePay AppSwitch

MobilePay tarjoaa yrityksille erilaisia ratkaisuja maksujen vastaanottamista varten. Yksi näistä on AppSwitch, joka on suunniteltu maksujen tekemiseen kolmannen osapuolen sovelluksen (asiakassovellus, kauppiassovellus) kautta. AppSwitch mahdollistaa menetelmän, jossa asiakas valitsee sovelluksestamme haluamansa tuotteen, mutta siirtyy maksun suoritusta varten automaattisesti MobilePay sovellukseen. Vastaavasti maksun suoritettuaan asiakas siirretään automaattisesti takaisin kauppiassovellukseen. Kauppias voi tarkastella ja hallinnoida asiakkaiden maksusuorituksia ohjelmallisesti MobilePay AppSwitch API:n kautta. Tarvitsemme tätä ominaisuutta katevarausten lunastamiseksi, kun asiakas näyttää QR-koodia ravintolan kassalla, koska asiakkaan ostovaiheessa tekemä maksu toteutetaan tapauksessamme ainoastaan katevarauksena.

AppSwitch API on REST-toiminnallisuuden toteuttava rajapinta, jota tyypillisesti käytetään palvelinohjelmiston kautta. AppSwitch API:n käyttöönotto vaatii kuitenkin muutamia aloitustoimenpiteitä. Kauppiaan on ensin valittava AppSwitch käyttöön MobilePayn developer portaalissa. Jotta rajapintaan voidaan tehdä onnistuneesti HTTP-pyyntöjä, on luotava varmenne (eng. certificate) jonka avulla palvelimelta suoritettavat HTTP-pyynnöt allekirjoitetaan. Varmenne voidaan joko hankkia varmenteita myöntävältä yritykseltä, tai luoda itse esimerkiksi OpenSSL:ää käyttäen.

OpenSSL-varmenteen toiminta perustuu avainpariin, joista toinen on julkinen ja toinen yksityinen. Julkinen avain toimitetaan MobilePaylle sähköpostilla ja yksityistä avainta säilytetään palvelimella (GitHub 2019). Kaikki AppSwitch API:iin tehtävät pyynnöt tulee allekirjoittaa yksityisen avaimen avulla. MobilePay vaatii, että allekirjoitus on JSON Web Signature-tyyppinen. Node.js ympäristössä JWS-spesifikaation mukainen allekirjoitus voidaan luoda npm-paketinhallinnan avulla asennettavaa jws-kirjastoa hyödyntäen. JWS asennetaan Node.js-projektiin komennolla npm install jws. Nyt allekirjoitus voidaan luoda kuvion 18 esimerkin mukaisesti.

```
const signature = jws.sign({
  header: { alg: 'HS256' },
  payload: 'https://api.mobeco.dk/appswitch/api/v1/merchants/merchant_id/orders/order_id',
  secret: 'yksityinenavain',
})
```

Kuvio 18: JWT-allekirjoituksen luominen

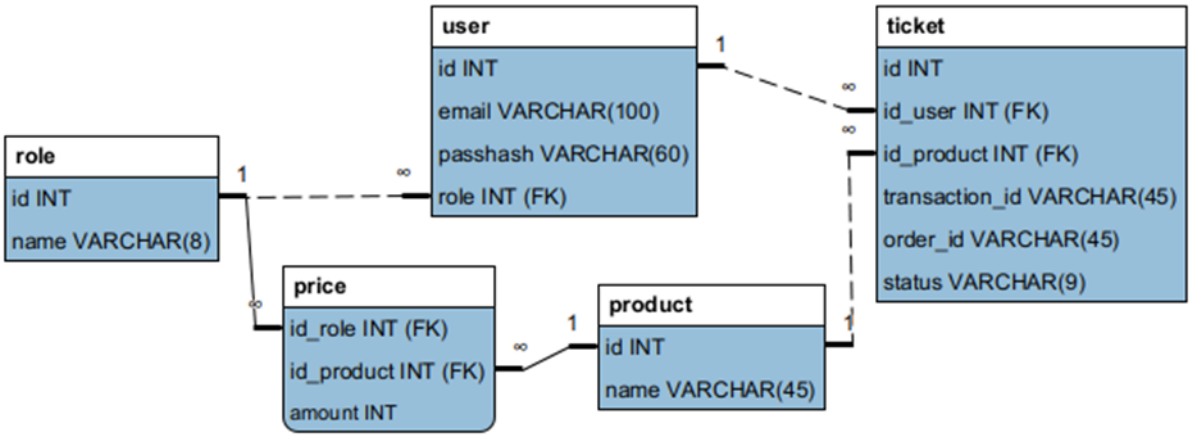
Kaikkiin AppSwitch API:iin tehtäviin pyyntöihin on asetettava kaksi otsaketta (eng. header). Mobile-Pay developer portaalin myöntämä kauppiaskohtainen 'subscription key' tulee otsakkeeksi avaimella 'Ocp-Apim-Subscription-Key'. Toinen vaadittava otsake on 'AuthenticationSignature' jonka sisällöksi tulee aiemmin luotu jws-tyyppinen allekirjoitus (GitHub 2018). Olettaen, että MobilePay on käsitellyt sekä hyväksynyt AppSwitchin käyttöönoton sekä sähköpostilla lähetetyn julkisen avaimen, voidaan rajapintaan tehdä HTTP-pyyntöjä käyttäen edellä mainittuja otsakkeita.

9.2 Tietokannan suunnittelu

Palvelimella sijaitsevan tietokannan tarkoituksena on pitää kirjaa rekisteröityneistä käyttäjistä, olemassa olevista tuotteista, niiden hinnoista sekä käyttäjien ostamista lipuista. Lippujärjestelmän vaatima toiminnallisuus voidaan saavuttaa tietokannalla, joka täyttää seuraavat vähimmäisvaatimukset. Tietokannan vähimmäisvaatimukset:

- Uuden käyttäjän lisääminen
- Käyttäjän roolin muuttaminen
- Kirjautuminen palvelimelle
- Uuden lipun tallentaminen
- Lippujen personointi
- Lipun tilan muuttaminen
- Uusien tuotteiden lisääminen
- Hintakategoriat (regular, personnel, student)
- Uusien kategorioiden lisäys
- Jokaisella tuotteella oltava hinta jokaiseen kategoriaan

Kyseinen tietokanta voitaisiin toteuttaa myös dokumenttipohjaisena, niin sanottuna NoSQL-tietokantana, mutta käytämme esimerkkinä SQL-tietokantaa. Nämä vähimmäisvaatimukset täyttävä tietokanta voidaan toteuttaa SQL-tietokannassa viiden taulun avulla. Käyttämämme tietokantamalli mahdollistaa myös uusien hintakategorioiden luomisen (kuvio 19).



Kuvio 19: Tietokantakaavio asiakassovelluksesta

10 Yhteenveto ja johtopäätökset

Työn tarkoituksena oli suunnitella opetus- ja lounasravintola BarLaurealle mobiilimaksusovellus heidän toiveidensa mukaan sekä sitä tukeva taustajärjestelmä. Projekti oli melkoisen laaja, jonka vuoksi huomioitavaa ja teknologiaa oli lopulta enemmän kuin aluksi kuvittelimme.

Kävimme projektin alkuvaiheessa läpi eri sovellustyyppivaihtoehtoja mobiilisovelluksen osalta ja päädyimme jo varhaisessa vaiheessa valitsemaan React Nativen sovelluskehykseksi, pääasiassa kehitystyön sujuvuuden takia. Johtopäätöksiin päädyttiin käyttämällä hyödyksi teknologioista aikaisemmin opittua hiljaista tietoa sekä verkkodokumentaatioista kerättyä tietoa. Järjestelmän toimivuuden kannalta kaikkein haasteellisimmaksi yksityiskohdaksi osoittautui sopivan maksupalveluntarjoajan löytäminen ja lopulta maksun suorittaminen valitun MobilePayn avulla.

MobilePayn toiminnan ja maksusuorituksen toteutustavan selvittäminen vei meiltä huomattavan suuren osan työajasta. Onnistuimme kuitenkin selvittämään, miten katevaraus tehdään React Native-sovelluksessa sekä kuinka se lopulta lunastetaan MobilePayn kehittämän rajapinnan kautta palvelimen puolella.

Suunnittelimme myös SQL-tietokannan, jonka avulla käyttäjistä ja heidän ostamistaan lipuista sekä lippujen tilasta pidetään kirjaa. Tietokanta voi olla myös dynaamisia malleja tukeva NoSQL-tietokanta.

Tutkimuksessa kerätty tieto täyttää sille validiteettiset ja reliabiliteettiset vaatimukset. Validiteetti määriteltiin löyhäksi, koska viime kädessä tieto perustuu tutkijoiden mielipiteisiin. Reliabiliteetissa työn konsistenssi on mielestämme hyvä, mutta stabiliteettia ei voi määrittää ohjelmointiteknologioiden nopean kehittymisen ja vaihtuvuuden takia.

Tämän työn lähtökohtana oli asiakkaan keräämä tieto heidän tarpeistaan mobiilimaksusovellukselle. Kyseinen tieto on validia, koska se tukee heidän taloudellista kasvustrategiaansa ja digitalisoitumista. Koemme reliabiliteetin olevan riittävä, sillä molemmat tutkimukseen osallistujat päätyivät tutkimuksen avulla samaan lopputulokseen.

Asiakkaan kuvauksen mukainen järjestelmäkokonaisuus on tehokasta ja taloudellista toteuttaa työssä kuvatulla tavalla. React Native on mobiilisovelluksen tekemiseen suorituskykyinen ja monipuolinen sovelluskehys. Palvelimen kehityksessä Express/Node.js on yleinen ja helppo käyttöinen kokonaisuus. JWT Passport mahdollistaa helpon ja turvallisen käyttäjähallinnan tietokannasta riippumatta.

11 Jatkokehitys

Jatkokehityksenä sovellukseen voisi lisätä automaattisen opiskelijatunnistuksen esimerkiksi Shibboleth-tunnistautumisjärjestelmän kautta tai vaihtoehtoisesti Frankin sovelluksen avulla. Frank on sähköisen opiskelijakortin tarjoava järjestö. Se tunnistaisi opiskelijat laajemmin myös muista kouluista, koska tällä hetkellä sovelluksen tietokanta on suunniteltu tunnistamaan vain Laurean opiskelijat. Lisäksi sovelluksessa voisi myös näkyä Leppävaaran koulun kahvilan hinnasto ja saada yhtenäisiä tarjouksia ravintola BarLaurean kanssa.

Teknologian jatkokehityksenä React Native antaa helpon mahdollisuuden päivittää Android sekä iOS alustalle yhtä aikaa. React Native ja React ovat syntaksiltaan hyvin samankaltaisia. On siis mahdollista hyödyntää jo olemassa olevaa koodia myöhemmin, jos halutaan rakentaa PWA tukemaan myös selainkäyttäjiä.

Tämän hetkisessä suunnitelmassa maksutavaksi on valittu MobilePay, mutta MobilePayn rinnalle pystyttäisiin lisäämään muitakin maksutapoja kuten PayPal. Lisäksi tuleva open banking (PSD2-maksupalveludirektiivi) mahdollistaisi suorat veloitukset asiakkailta (Nordea 2017).

Järjestelmän toimintaa eri osa-alueiden välisen tiedonsiirron osalta kuvaavassa osiossa on otettu huomioon myös QR-koodilukijan rooli. Emme kuitenkaan saaneet käyttöömme kyseistä lukijalaitetta, joten emme päässeet kokeilemaan ja tutkimaan sen toimintaa.

Toinen projektista pois rajattu komponentti on henkilökunnan käyttöön tarkoitettu hallintapaneeli, jonka avulla olisi mahdollista lisätä sovellukseen tuotteita sekä muuttaa niiden tietoja ja hintoja. Tietokanta kuitenkin suunniteltiin niin, että kyseinen toiminnallisuus on mahdollista ottaa myöhemmin käyttöön.

Työssä esitelty järjestelmä on suunniteltu helposti laajennettavaksi. Tämän mahdollistaa eri osien erottaminen toisistaan, jolloin yhden osan muuttaminen tai uuden osan lisääminen ei vaikuta muiden osien toimintaan suuresti.

12 Oman oppimisen arviointi

Opimme projektissa erityisesti suunnitelmallisuuden, dokumentoinnin ja kommunikoinnin tärkeyttä. Kun tekee töitä muiden kanssa, on tärkeää tietää mitä he tekevät. On myös pystyttävä kertomaan muille omat ajatukset ja suunnitelmat omiin työn alla oleviin asioihin liittyen. Teknisessä mielessä tämä oli laaja kokonaisuus. Kumpikaan ei ole aiemmin tehnyt projektia, jossa tarvitaan näin monen eri osa-alueen osaamista ja hallintaa. Ymmärryksemme kasvoi huomattavasti siitä, että miksi projekteilla on niin monta ammattilaista eri rooleissa, jotka osaavat huomioida paremmin oman osaamisalueensa. Saatoimme välillä ajatella liian kehittäjämäisesti, jonka takia meidän piti harjoitella ajattelemaan enemmän käyttäjäystävällisesti. Olemme tottuneet suunnittelemaan pieniä sovelluksia omaan käyttöömme, mutta tässä meidän piti huomioida muiden käyttäjien kannalta kokonaisuutta.

Opintojen osalta hyödylliseksi on osoittautunut erityisesti JavaScript ja siihen liittyvien kirjastojen käyttäminen sekä SQL-tietämys. Opiskelimme projektin aikana myös molemmille aiemmin tuntematonta React Native-sovelluskehystä. On hauska huomata, että uuden teknologian opettelu ei ole niin työlästä, jos se on sukua jollekin aiemmin tutulle teknologialle. Molemmilla on jonkin verran kokemusta esimerkiksi Angularista ja mobiilisovellusten kehittämisestä sen avulla, sekä toisaalta web-sovellusten kehittämisestä Reactin avulla, joten React Nativea ei tarvinnut opetella täysin tyhjältä pohjalta. Meille myös selkeytyi kuinka käyttää yhdessä erilaisia teknologioita ja jo aikaisempien opittujen asioiden ymmärtäminen syventyi. Lisäksi pääsimme opettelemaan parempaa asiakasläheisempää työskentelyä ja kommunikaatiota.

Projektia tehdessämme saimme hyvät valmiudet työelämään, koska tiimitaitomme kehittyivät ja ymmärrämme nyt paremmin mitä kaikkea tällaiseen projektiin kuuluu. Toimeksiantajamme oli kiitollinen osallisuudestamme ja iloinen siitä, että vihdoin projektia saadaan eteenpäin. Kokemuksena tämä oli opettavainen ja oli mahtavaa, että saimme toimia tiiminä. Olemme myös kiitollisia siitä, että saimme mahdollisuuden suunnitella sekä osittain toteuttaa Bar-Laurealle tämän tyyppisen projektin, jolle he olivat jo pitkään etsineet tekijöitä.

Lähteet

Painetut

Allen G, 2006. SQL For Dummies 6th Edition. Hoboken, New Jersey

Hahn, E. 2016. Express In Action. Manning Publications.

Lancaster, G. 2005. Research methods in Management: a concise introduction to research in management and business consultancy. Oxford: Elsevier Butterworth-Heinemann.

Pohjalainen, M. 2012. Hiljaisen tiedon käsite ja hiljaisen tiedon tutkimus. Tampere: Informaatiotutkimuksen yhdistys.

Sähköiset

Agicent 2018. Progressive web apps vs responsive web apps vs native apps. Viitattu 13.3.2019. https://www.agicent.com/blog/progressive-web-apps-vs-responsive-web-apps-vs-native-apps/

Danske Bank 2019. MobilePay. Viitattu 15.3.2019. https://danskebank.fi/en-fi/MobilePay-content/Pages/P2PHelp.aspx

Facebook 2019. Components and APIs. Viitattu 28.3.2019. https://facebook.github.io/react-native/docs/components-and-apis.html

FixUi 2018. Mikä ihmeen rautalankamalli? Viitattu 15.3.2019. https://fixui.fi/kaytettavyys/mika-ihmeen-rautalankamalli/

GitHub 2017. Capture Amount Interface Description. Viitattu 17.3.2019. https://github.com/MobilePayDev/MobilePay-AppSwitch-API/blob/master/REST%20APIs/v1/capture%20amount%20interface%20description.md#resources

GitHub 2018. Merchant requests to the MobilePay AppSwitch or Connect APIs. Viitattu 29.3.2019. https://github.com/MobilePayDev/MobilePay-Merchant-API-Security/blob/master/Merchant-request.md

GitHub 2018. MobilePay AppSwitch SDK. Viitattu 28.3.2019. https://github.com/MobilePayDev/MobilePay-AppSwitch-SDK

GitHub 2018. React-Native Bindings for Danske Bank Mobilepay AppSwitch iOS + Android. Viitattu 28.3.2019. https://github.com/madsleejensen/react-native-mobilepay

GitHub 2019. Technical documentation for onboarding merchants. Viitattu 29.3.2019. https://github.com/MobilePayDev/MobilePay-Merchant-API-Security/blob/master/Merchant-onboarding.md

Hackernoon 2018. Native App Development vs. Hybrid App Development 12.3.2019. https://hackernoon.com/native-app-development-vs-hybrid-app-development-dd83122a738c

Hiltunen, L. 2019. Validiteetti ja reliabiliteetti. Jyväskylän Yliopisto. Viitattu 4.5.2019. http://www.mit.jyu.fi/OPE/kurssit/Graduryhma/PDFt/validius_ja_reliabiliteetti.pdf

Innovation English 2018. Nabc. Viitattu 14.3.2019. https://innovationenglish.sites.ku.dk/model/nabc/

InVision 2018. How to make your first wireframe. Viitattu 25.5.2019. https://www.invisionapp.com/inside-design/how-to-wireframe/

Ionic 2018. Ionic vs. React Native: A Comparison Guide. Viitattu 14.3.2019. https://ionicframework.com/resources/articles/ionic-vs-react-native-a-comparison-guide

Ionic. What is a Progressive Web App and Why You Need One. Viitattu 14.3.2019. https://ionicframework.com/resources/articles/what-is-a-progressive-web-app-and-why-you-need-one

itewiki 2018. Ketterät menetelmät, agile, LEAN ja scrum. Viitattu 24.5.2019. https://www.itewiki.fi/opas/ketterat-menetelmat-agile-lean-ja-scrum/

JWT 2019. Debugger. Viitattu 14.3.2019. https://jwt.io/

Laitinen, O. 2019. Mobiilimaksaminen vuonna 2019 – Näin se toimii. Viitattu 15.3.2019. https://www.salkunrakentaja.fi/2019/01/mobiilimaksaminen-2019/

Medium 2017. Comparing the Performance between Native iOS (Swift) and React-Native. Viitattu 15.3.2019. https://medium.com/the-react-native-log/comparing-the-performance-between-native-ios-swift-and-react-native-7b5490d363e2

Medium 2017. Learn using JWT with Passport authentication. Viitattu 21.3.2018. https://medium.com/front-end-weekly/learn-using-jwt-with-passport-authentication-9761539c4314

Medium 2019. 3 Limitations of Progressive Web Apps. Viitattu 14.3.2019. https://medium.com/@strv/3-limitations-of-progressive-web-apps-98b5afca9338

MobilePay 2019. Is it possible to make a reservation payment? Viitattu 25.3.2019. https://developer.mobilepay.dk/node/164

MobilePay 2019. MobilePay AppSwitch. Viitattu 16.3.2019. https://www.mobilepay.fi/yrityksille/sovellukset%20ja%20verkkokaupat/mobilepay%20appswitch

Niels Christ 2012. The NABC Method from Stanford Research Institute, SRI. Viitattu 14.3.2019.

https://nielschrist.wordpress.com/2012/07/13/the-nabc-method-standford-research-institute-sri/

Node.js 2019. Introduction to Node.js. Viitattu 23.5.2019. https://nodejs.dev/

Nordea 2017. Open Banking tulee - mitä se tarkoittaa? Viitattu 3.4.2019. https://www.nordea.com/fi/media/uutiset-ja-lehdistotiedotteet/News-fi/2017/2017-03-09-open-banking-tulee.html

npm 2018. axios. Viitattu 19.3.2019. https://www.npmjs.com/package/axios

Näpärä, L. 2017. Tutkimuskysymyksen muodostaminen. Viitattu 28.5.2019. https://www.spoken.fi/tutkimuskysymyksen-muodostaminen/

Ohjelmistotuotanto 2018. Ketterät menetelmät: Scrum. Viitattu 19.3.2019. http://ohjelmistotuotanto.panuleppaniemi.com/agile-scrum/

Orbit Media Studios 2016. Website Navigation: 7 Best Practices, Design Tips and Warnings. Viitattu 7.5.2019. https://www.orbitmedia.com/blog/website-navigation/

O'reilly 2018. Chapter 1. What Is React Native? Viitattu 14.3.2019. https://www.oreilly.com/library/view/learning-react-native/9781491929049/ch01.html

Raywenderlich 2018. React Native Tutorial: Building Android Apps with JavaScript. Viitattu 28.3.2019. https://www.raywenderlich.com/247-react-native-tutorial-building-android-apps-with-javascript

REST API Tutorial 2019. HTTP Methods. Viitattu 19.3.2019. https://restfulapi.net/http-methods/

Search Mobile Computing 2018. Android Studio. Viitattu 9.5.2019. https://searchmobilecomputing.techtarget.com/definition/Android-Studio

TechTarget 2018. MySQL. Viitattu 23.5.2019. https://searchoracle.techtarget.com/definition/MySQL

The World Wide Web Consortium 2001. Token Based Authentication -- Implementation Demonstration. Viitattu 16.3.2019. https://www.w3.org/2001/sw/Europe/events/foaf-galway/papers/fp/token_based_authentication/

UX Planet 2017. Mobile UI Design: Basic Types of Screens. Viitattu 15.3.2019. https://uxplanet.org/mobile-ui-design-basic-types-of-screens-aa1857e31339

Web 2019. Using Trusted Web Activities. Viitattu 14.3.2019. https://developers.google.com/web/updates/2019/02/using-twa

Yrittäjät 2017. Verkkopankkia käyttää jo mummokin: Tutkimustulokset perustelevat konttoreista luopumista. Viitattu 16.3.2019. https://www.yrittajat.fi/uutiset/559823-verkkopankkia-kayttaa-jo-mummokin-tutkimustulokset-perustelevat-konttoreista

Kuviot

Taulukot

Liitteet

Liite 1: Suunnitellut reitit (eng. endpoints)

Reitti	Tyyppi	Kuvaus
/api/user	GET	Haetaan tokenin osoittaman käyttäjän tiedot tietokannasta.
/api/user/:id	DELETE	Käyttäjä voidaan poistaa id:n perusteella.
/api/user/tickets	GET	Haetaan tokenin osoittaman käyttäjän liput.
/api/user/tickets	POST	Lisätään tokenin osoittamalle käyttäjälle uusi lippu.
/api/user/tickets/:id	PUT	Varmistetaan id:n osoittama lippu tokenin haltijan ostokseksi ja muokataan lippua. Esimerkiksi lippu merkitään käytetyksi.
/api/prices	GET	Haetaan ajan tasalla olevat hinnat.
/api/prices	PUT	Muokataan hintoja
/api/login	POST	Tarkistetaan pyynnön mukana tulleet käyttäjätiedot ja generoidaan token, joka lähetetään takaisin vastauksena.
/api/register	POST	Tarkistetaan pyynnön mukana tulleet käyttäjätiedot ja luodaan uusi käyttä-jä tietokantaan. Generoidaan token, joka lähetetään takaisin vastauksena.