TULOSTAMINEN PILVIPALVELUSTA PAIKALLISELLE TULOSTIMELLE

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

Tietojenkäsittelyn koulutus
Kevät, 2023

Toni Nurmi

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tietojenkäsittelyn koulutus Tiivistelmä

Tekijä Toni Nurmi Vuosi 2023

Työn nimi Tulostaminen pilvipalvelusta paikalliselle tulostimelle

Ohjaaja Pentti Ojaniemi

TIIVISTELMÄ

Opinnäytetyön tarkoituksena oli luoda Android-sovelluksen, joka tulostaa pilvipalvelussa sijaitsevista tietokannoista terveydenhuoltoasemien potilaiden ajanvaraukset paikallisella kuittitulostimella. Sovellus tulee toimimaan itseilmoittautumislaitteena sairaaloissa ja muissa terveysasemissa.

Opinnäytetyö oli toiminnallinen ja sen tietopohja perustui paljolti Googlen sekä Microsoftin omiin kehittäjädokumentaatioihin. Koska Android-ohjelmoinnista ei ollut aiempaa kokemusta, työn teoriaosuudessa tutustuttiin myös hieman Androidin perusteisiin.

Vaikka työn pääasiallisena aiheena oli Android-sovellus, tätä projektia varten luotiin täysin uusi tietokanta pilveen, koska oikeisiin potilastietoihin ei ollut pääsyä projektia luodessa. Pilvipalvelun lisäksi luotiin myös käyttöliittymä, joka hakee tietokannasta ajanvaraukseen liittyvät tiedot ja lähettää ne Android-sovellukselle tulostettavaksi.

Android-sovelluksen kehittäminen osoittautui helpoksi ja aloittelijaystävälliseksi prosessiksi. Suurimmat haasteet tulivat Android-osuuden ulkopuolelta tietokannan ja käyttöliittymän kautta tietoturvallisista syistä, joihin työn alkaessa ei ollut juurikaan aikaisempaa kokemusta. Kaikissa ongelmissa kuitenkin auttoi paljon Microsoftin ja Googlen dokumentit sekä lukuisat muut internetistä löytyvät lähteet.

Avainsanat Android, Kotlin, React, pilvipalvelut, tulostaminen

Sivut 57 sivua ja liitteitä 3 sivua

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Degree Programme in Business Information Technology
Author Toni Nurmi
Subject Printing from cloud services to a local printer
Supervisor Pentti Ojaniemi

Abstract
Year 2023

ABSTRACT

The goal of the thesis was to create an Android application, that prints hospital's patients' appointments from a cloud service to a local receipt printer. The application will be a self-registration kiosk on hospitals and other healthcare institutions.

The thesis was practical, and its theory is based a lot on Google's and Microsoft's own developer documentations. The theoretical part of the thesis also included the basics of programming with Android, since there was no previous experience in creating an Android app.

Even though the Android application was the main subject of the thesis, due to patient records being unattainable during the creation of the project, there was a database created into cloud services for this project. On top of the database, there was also a user interface that was created to fetch the data from the cloud service and send it to the Android application for printing.

Programming an Android application turned out to be an easy and beginner friendly experience. The biggest challenges came from outside of the Android part, because of the database and user interface not being able to connect to unsecure connections, for which there was close to no previous experience. All the problems were sorted out due to Google's and Microsoft's documentations and the vast number of other sources on the internet.

Keywords Android, Kotlin, React, cloud services, printing
Pages 57 pages and appendices 3 pages

Sanasto

Android

Avoimen lähdekoodin mobiilikäyttöjärjestelmä.

Android SDK

(Software Development Kit, suom. ohjelmistokehityspaketti) Paketti, joka sisältää Android-sovelluskehitykseen tarkoitetun emulaattorin, vianetsintään tarkoitettuja työkaluja, kirjastoja, dokumentteja ja esimerkkikoodeja kehittäjiä varten.

C#

Ohjelmointikieli, joka on kehitetty .NET-alustalle.

Debuggeri

Tietokoneohjelma, joka auttaa löytämään ja korjaamaan ohjelmointivirheitä muista tietokoneohjelmista. Erittäin monessa ohjelmistoympäristössä on sisäänrakennettu debuggeri.

Emulaattori

Tietokonejärjestelmä, joka simuloi toisen järjestelmän toimintaa fyysiseltä tasolta lähtien käyttäytyäkseen mahdollisimman samalla tavalla mahdollistaen ohjelmien suorittamisen alustoilla, joille niitä ei ole alun perin tarkoitettu.

Entity Framework Core

Kevyt avoimen lähdekoodin järjestelmästä riippumaton teknologia, joka mahdollistaa tietoihin pääsyn. Mahdollistaa tietokantojen kanssa työskentelyn .NET olioilla.

HTTP/HTTPS

Protokolla, jota www-palvelimet käyttävät tiedonsiirtoon. Termi ”HTTP” tulee sanoista ”Hypertext Transfer Protocol”, ”HTTPS”:ssä sanojen perään tulee ”Secure”-sana, joka käyttää TLS-salausta tiedon suojatun siirron varmistamiseksi.

HTTP-otsikko

Välitetään HTTP-pyynnön tai -vastauksen alussa. Sisältää tietoa pyynnön tai vastauksen kontekstista, kuten käytetty selain, pyynnön tyyppi (esim. POST tai GET) sekä muuta pyynnön ja vastauksen kannalta oleellista tietoa.

HTTP-metodi

HTTP-pyynnössä käytetty pyynnön tyypin ja tarkoituksen määrite, eli haetaanko, lähetetäänkö, päivitetäänkö vai poistetaanko resursseja.

JavaScript

Pääasiallisesti verkkoympäristöön soveltuva ohjelmointikieli.

JSON

(JavaScript Object Notation) Erittäin kevyt ja yksinkertainen tiedostomuoto tiedonvälitykseen ja sen tallennukseen.

Kotlin

Vuodesta 2019 lähtien Googlen suosittelema Android-sovelluskehitykseen tarkoitettu ohjelmointikieli.

Lambdailmaisu

(Lambda expression) Lyhyt koodilohko, joka ottaa vastaan vähintään yhden parametrin ja palauttaa arvon. Lambdailmaisut eivät tarvitse nimeä ja ne voidaan toteuttaa metodin keskellä.

Metodi

Koodilohko, joka suoritetaan vain, kun sitä kutsutaan muualla koodissa. Eroten lambdailmaisusta metodille annetaan aina oma nimi, jotta sitä voidaan kutsua erikseen.

.NET

Ohjelmistokomponenttikirjasto Microsoft Visual Studio-ympäristössä kehitettyjen ohjelmistojen käyttöön.

REST

(REpresentational State Transfer) HTTP tai HTTPS-protokollien avulla ohjelmointirajapintojen toteutettu arkkitehtuurityyli.

Web API

(Web Application Programming Interface) on ohjelmointirajapinta, jolla voi siirtää tietoja helposti sivustojen välillä.

XML

(Extensible Markup Language) Merkintäkieli, jota käytetään tiedon kuvaukseen tai formaattina tiedon tallentamiseen.

Sisällys

Kuvat, ohjelmakoodit ja taulukot

1 Johdanto

Opinnäytetyön aiheena on pilvipalvelusta kerättyjen tietojen tulostaminen paikallisella tulostimella. Työn käytännönosan tuloksena syntyy ohjelma, jonka tarkoitus on toimia itseilmoittautumisautomaattina, esimerkiksi sairaalassa, jossa asiakas voi skannata Android-laitteeseen kytketyllä viivakoodinlukijalla henkilökorttinsa viivakoodin. Skannauksen jälkeen automaatti rekisteröi asiakkaan ilmoittautuneeksi sairaalaan vastaanottoa varten. Tämän jälkeen asiakkaalle tulostuu kuitti tulostimesta, joka on myös yhdistetty Android-laitteeseen. Kuitissa lukee käynnin tarkoitus, huone, johon asiakkaan aika on varattu sekä kellonaika, jolloin asiakkaan vastaanotto alkaa. Tarvittaessa kuittiin voidaan myös tulostaa kartta, joka näyttää reitin suorinta tietä automaatilta vastaantottohuoneeseen.

Projektia varten tarvitaan tulostin, joka henkilön tunnistautuessa tulostaa kuitin tarvittavilla tiedoilla. Vaihtoehtoina tähän oli monia tulostinvalmistajia, mutta parhaimmaksi todettiin nykyisten kuittitulostinmarkkinoiden huippu; Star Micronics ja tulostimeksi valittiin heidän TSP650-sarjan tulostin. Jatkokehityksessä ohjelman koodia muokataan toimimaan useammilla tulostimilla asiakkaan tarpeiden mukaan, mutta tässä opinnäytetyössä käytetään vain Star Micronicsin omia Android-kirjastoja hyödyksi.

Tätä opinnäytetyötä tehtäessä käytössä oli vain Android-puhelin, johon sai kiinni vain tulostimen, joten tässä projektissa viivakoodinlukijan sijasta sovelluksessa on vain nappi, joka simuloi mitä tapahtuisi, jos viivakoodinlukijalla lukisi henkilökortin viivakoodin.

Opinnäytetyön keskeisinä aiheina ovat React-ohjelmointikielellä luodulla käyttöliittymällä REST-palvelusta saatujen tietojen lukeminen ja niiden lähettäminen Android-sovellukselle, joka ohjaa paikallista tulostinta tulostamaan tarvittavat tiedot kuitille.

Opinnäytetyössä pyritään vastaamaan seuraaviin kysymyksiin:

- Miten asiakkaalle tulostetaan kuitti, joka sisältää pilveen tallennetut tiedot?
- Miten React-koodilla saadaan ohjattua Android-käyttöjärjestelmään liitettyä tulostinta?
- Miten asiakkaat tulevat hyötymään tästä työstä?

2 Opinnäytetyössä hyödynnettävät laitteet ja teknologiat

Opinnäytetyössä tullaan hyödyntämään monia eri ohjelmointiympäristöjä (Integrated Development Environment, lyh. IDE), kuten Microsoftin Visual Studio, Visual Studio Code sekä Googlen Android-sovellusten luomiseen kehittämä Android Studio. Ohjelmointikielinä tulee toimimaan C#; jolla luodaan sovellukselle yhteys tietokantaan, jota käytetään pilvessä PostgreSQL tietokantahallintajärjestelmässä, JavaScript; joka toimii tämän opinnäytetyön käyttöliittymän pohjana sekä Kotlin; jolla tämä kaikki saadaan toimimaan Android-laitteella.

Fyysisinä laitteina toimii testausvaiheessa OnePlus-mobiililaite ja lopullisessa versiossa toimeksiantajan valitsema Android-tabletti, joka tukee Android 11 tai uudempaa käyttöjärjestelmää. Android-laitteeseen kiinnitetään USB:lla kuittitulostin, johon oli vaihtoehtoina muutamia valmistajia kuten Toshiba ja Zebra, mutta Android ohjelmistokehityspakettien saatavuuden takia lopulta päädyttiin Star Micronicsin valmistamaan TSP654II-kuittitulostimeen.

Android-laitteeseen luotava ja asennettava sovellus tulee toimimaan itseilmoittautumisautomaattina sairaaloissa ja siinä kiinni oleva kuittitulostin tulostaa potilaalle lapun, joka tulee sisältämään tämän vuoronumeron, vastaanottoajan, vastaanottohuoneen sekä ohjeet, miten kyseiseen huoneeseen kuljetaan automaatilta.

Opinnäytetyön valmistuessa toimeksiantajan käyttöönottamassa versiossa Android-laitteeseen tulee kiinni myös viivakoodinlukija, jonka avulla sairaalan asiakkaat tunnistautuvat ja ilmoittautuvat paikalle skannaamalla henkilökorttinsa lukijalla. Koska tätä projektia tehdessä käytössä ei ole kuin Android-laite, jossa on yksi USB C-liitäntä, tätä viivakoodinlukijaa ja sen toimintoa tullaan simuloimaan JavaScriptissä luodulla napilla, jota Android-laitteella painetaan, kun halutaan tulostaa asiakkaalle kuitti.

2.1 Itseilmoittautumislaite

Itseilmoittautumisautomaatit eivät ole uusia keksintöjä. Ensimmäinen itsepalvelukioski kehitettiin jo vuonna 1977 Amerikassa, jonka jälkeen teknologian kehittyessä automaattien suosio räjähti

valtavasti 2000-luvun alussa. Suomessa näitä laitteita on nähty 2010-luvun vaihteesta asti ja niiden määrä on noussut huomattavasti viimeisen muutaman vuoden aikana. (Kioskindustry, 2023; Yle, 2013)

Automaattien tarkoituksena on nopeuttaa ilmoittautumistoimenpidettä huomattavasti ja vähentää sairaalan henkilökunnan työpanosta ilmoittautumisen osalta. Myös inhimilliset kirjaamisvirheet poistuvat parhaimmassa tapauksessa kokonaan, kun henkilötietoja ei lue ihminen, vaan kone viivakoodin avulla. (Finn-ID, n.d.)

Nykyään tämän henkisiä itsepalveluautomaatteja löytyy monenlaisista eri palveluista, kuten ostoskeskuksissa karttoina, pikaruokaravintoloista ruoan tilaamiseen, lentokentiltä ja rautatieasemilta matkalippujen tilaamiseen sekä tietenkin opinnäytetyön pääkohteista, eli terveydenhuoltoasemilta vastaanotolle ilmoittautumiseen. (Redyref, 2021) Kuitenkin varmaan tunnetuin ja pisimpään yleisessä käytössä ollut itsepalveluautomaatti on pankkiautomaatti, joita löytyy kadunvarsilta ympäri maailmaa, jossa asiakkaat voivat pankkiautomaatti-, pankki-, luotto- tai prepaid-kortilla nostaa käteisrahaa, maksaa laskuja tai siirtää rahaa tilien välillä. (Investopedia, 2023)

Itseilmoittautumisautomaatteja on monenlaisia, mutta nykyään suurin osa niistä on kosketusnäytöllisiä vapaasti seisovia automaatteja, joissa on viivakoodin lukuun soveltuva lukija ja tulostin. Joskus lisänä voi olla myös fyysinen näppäimistö kosketusnäytön lisäksi. (Lux, 2004, s. 29; Rhoads & Drazen, 2009, s. 7) Vapaasti seisovalla automaatilla tarkoitetaan noin 1,5–2 metrin korkuisia kioskeja, jotka voidaan asentaa minne tahansa lattialle. Vaihtoehtoisia malleja on seinälle tai tasolle asennettavat automaatit sekä liikuteltavat automaatit. (Rhoads & Drazen, 2009, s. 9)

2.2 Star Micronics

Star Micronics on jo vuonna 1950-luvulla perustettu japanilainen elektroniikkakomponenttien ja kuittitulostimien valmistamiseen erikoistunut yhtiö. Perustettaessa Star Micronicsin tuotanto kohdistui rannekelloihin ja kameroiden osiin, kun taas tulostimien valmistaminen aloitettiin

myöhemmin vuonna 1979. Nykyään Star Micronics toimii monikansallisesti myös monessa muussa Aasian maassa, Euroopassa sekä Pohjois-Amerikassa. (Star Micronics, n.d. a)

Yksi Star Micronicsin päätuotelajeista on nykyään POS-tulostimet (point of sale). Niiden pääasiallinen käyttötarkoitus on kuittien, laskujen ja muiden pienien dokumenttien tulostaminen kaupoissa, sairaaloissa ja muilla toimialoilla. Star Micronics tarjoaa laajan valikoiman kuittitulostimia, joihin lukeutuu lämpötulostimet, pistematriisitulostimet sekä mobiilitulostimet. (Star Micronics , n.d. b)

Lämpötulostimet ovat yleisin kuittitulostinmalli, jotka käyttävät lämpöä tekstin tai kuvan luomiseksi paperille tarjoten hyvän laatuisia ja tahrattomia tulosteita, jotka ovat ideaalisia kuiteille. Koska lämpötulostimet eivät käytä mustetta, vaan sen sijaan käyttävät lämpöä hyväksi siirtääkseen tekstiä sitä varten tarkoitetulle paperille, se on erittäin nopea tarvittavan kuitin tulostamiseen. (Techopedia, 2014)

Tässä opinnäytetyössä käytetty TSP654II-tulostin on lämpötulostin, jonka pääpiirteisiin lukeutuu markkinoiden huippuihin kuuluva 60 kuitin minuutissa -tulostamisnopeus. TSP650-sarjan tulostimet sisältävät myös USB ja Ethernet yhdistämismuodot. Uudemmissa malleissa on myös Bluetooth yhteys, joten Android-laitteen yhdistäminen näihin tulostimiin ei ole ongelma. (Star Micronics, 2023)

2.3 Back end kehitysympäristöt

Web-kehityksessä back end on ohjelman palvelinpuoli, joka on vastuussa muun muassa vuorovaikutuksista tietokannan kanssa. Yleensä back endiin sisältyy API:t, jotka ovat yhteydessä front endiin halliten tietokantaa, palvelimia ja turvallisuutta. Käyttäjällä ei ole pääsyä back endiin suoraan, vaan sen tietoja voi mahdollisesti nähdä front endin tarjoaman käyttöliittymän kautta. (TechTarget, 2019) Opinnäytetyössä luodaan täysin uusi palvelin pilveen, johon yhteys muodostetaan Visual Studio nimisellä ohjelmointiympäristöllä luodulla ohjelmalla.

2.3.1 Visual Studio

Visual Studio on Microsoftin alun perin vuonna 1997 julkaisema ohjelmointiympäristö. Ohjelmointiympäristönä se on tarkoitettu ohjelmoijalle suunnittelemaan, kehittämään ja muokkaamaan nettisivuja, mobiili- ja verkkosovelluksia sekä pilvipalveluita. Sen pääasiallisena ohjelmointikielenä toimii C#, joka toimii myös tässä opinnäytetyössä Visual Studio -osuuden ohjelmointikielenä. Sen muita tuettuja kieliä ovat mm. C, C++, Python ja verkko-ohjelmointiin tarkoitetut kielet kuten HTML, CSS ja JavaScript. Vuodesta 2017 lähtien Visual Studio ei ole enää tukenut Javaa. (FreeCodeCamp, 2023)

Tässä projektissa Visual Studiolla kehitetään täysin uusi back end, eli sovelluksen palvelin, jossa tarvittavat tiedot säilyvät pilveen luodussa tietokannassa. Tämän avulla pääsemme epäsuorasti käsiksi ulkoisella sovelluksella näihin tietoihin, jotka tässä tapauksessa ovat sairaalassa asioivien henkilöiden käyntitiedot sekä ajanvaraukset.

Tämän tietokannan luomiseen käytetään Visual Studiossa tuettua ASP.NET Core Web API -tukiympäristöä, joka takaa sen, että on helppoa rakentaa HTTP-palvelu, johon saa yhteyden mistä tahansa selaimesta tai mobiililaitteesta. (TutorialsTeacher, n.d.)

2.3.2 Swagger UI

Visual Studio käyttää valmiiksi HTTP-palvelujen suorittamiseen Swagger-nimistä kehityskalustoa. Swagger on vuonna 2010 julkaistu API-kehittäjille tarkoitettu työkalusarja. Swagger UI on osa Swaggerin sarjaa, joka antaa mahdollisuuden visualisoida ja toimia API:n resurssien kanssa ilman, että tarvitsee välittää taustalla pyörivästä koodista. (Swagger, n.d.) Swagger UI:ta käytetään tässä opinnäytetyössä helppona välikätenä tietokannan ja kehittäjän välillä uusien henkilötietojen lisäämiseksi pilvipalveluun.

2.3.3 PostgreSQL

Swagger UI:ssa tapahtuvat muutokset syötetään suoraan PostgreSQL -nimiseen avoimen lähdekoodin tietokantahallintajärjestelmään. PostgreSQL on nimensä mukaisesti SQL, eli

Structured Query Language, jolla voi tehdä lisäyksiä, muutoksia ja hakuja relaatiokantaan. Relaatiokanta on tietokanta, joka pohjautuu relaatiomalliin, eli se sisältää tauluja, joiden välille voi luoda yhteyksiä (relaatioita) tunnisteilla ja viiteavaimilla. (AWS, n.d.)

PostgreSQL julkaistiin ensimmäisen kerran vuoden 1995 puolessa välissä (silloin nimellä POSTGRES, nykyinen nimi otettiin käyttöön vuoden 1996 puolessa välissä), mutta se ei ole minkään yksittäisen yrityksen tai henkilön ohjaama, vaan se perustuu eri yrityksien ja ohjelmoijien tekemään yhteistyöhön ympäri maailmaa. Tällä yhteistyöllä on kuitenkin tietty pääryhmä, joka viimeisenä päättää, otetaanko muutokset mukaan virallisiin julkaisuihin. (The PostgreSQL Global Development Group, 2023, s. 32–34)

PostgreSQL on ohjattavissa monilla eri ohjelmilla, mutta tähän opinnäytetyöhön valittiin PostgreSQL:n tietokantoja ohjaamista varten luotu pgAdminin neljäs julkaisu. (PostgreSQL, 2017)

2.3.4 PgAdmin 4

PgAdmin 4 on Pythonilla ja JavaScriptillä luotu avoimeen lähdekoodiin perustuva työkalu, jolla ohjataan PostgreSQL:ää ja sen tietokantoja. (PgAdmin 4, 2023a) PgAdmin 4 toimii siis käyttöliittymänä PostgreSQL tietokantojen luomista varten. (PgAdmin 4, 2023b)

PgAdmin 4:llä on helppo luoda uusia yhteyksiä palvelimelle, jossa tietokantoja voi hallita. (The pgAdmin 4 Development Team, 2023, s. 99–104) Yhteyden muodostamisen jälkeen Swaggerissa tehdyt muutokset tulevat suoraan näkyviin pgAdmin 4:ssä näkyviin tietokantatauluihin ja myös pgAdminissa suoraan tauluihin tehtävät muutokset ovat myös nähtävissä Swaggerissa, tai missä tahansa, mistä palvelimeen tallennettuja tietoja tullaan myöhemmin lukemaan opinnäytetyön front end osiossa.

2.4 Front end kehitysympäristöt ja teknologiat

Web-kehityksessä front end on web-sovelluksen käyttöliittymä (UI, eli User Interface englanniksi). Tämä tarkoittaa, että käyttäjän web-selaimessa näkyvät elementit, käyttäjän syötteet ja

kommunikointi back endin kanssa tapahtuu käyttöliittymässä. (TechTarget, 2019) Opinnäytetyössä tullaan hyödyntämään kahta eri ohjelmistoympäristöä front endin luomiseen, joka tulee näkymään Android-sovelluksessa.

2.4.1 Visual Studio Code

Visual Studio Code (tunnetaan myös nimellä VS Code) on Microsoftin vuonna 2015 julkaisema avoimen lähdekoodin kevyt tekstieditori ohjelmoijia varten. VS Code tukee alustavasti Node JS -ajoympäristöä, jolla suoritetaan palvelimella myös VS Coden natiivisti tukemia TypeScriptiä sekä JavaScriptiä, joista jälkimmäinen toimii tässä opinnäytetyössä front endin ohjelmointikielenä. (FreeCodeCamp, 2023; NodeJS, n.d.)

Yksi VS Coden suurimpia vahvuuksia on kuitenkin sen joustavuus eri ohjelmointikielille. Edellä mainittujen kielien lisäksi se tukee myös suurinta osaa muistakin kielistä, joista suosituimpiin lukeutuu muun muassa Python, Java, CSS ja C++. Tämän takia VS Code on suosituimpien koodieditorien joukossa, koska se toimii hyvin ohjelmoijien keskuudessa, jotka työskentelevät monien kielien kanssa, tai vaihtavat eri projektien välillä usein. (Visual Studio Code, 2023a)

Eri ohjelmointikielien lisäksi VS Code tukee myös laajan kattaman muitakin laajennuksia, joilla ohjelmoija pystyy muokkaamaan ja laajentamaan editorin toimintoja helposti. Näihin lukeutuu esimerkiksi debugging-työkalut, tekstin helppo formatointi ja automaattiset koodin pätkät. (Visual Studio Code, 2023b)

2.4.2 React

React (tunnetaan myös nimellä React.js) on laajalti käytössä oleva JavaScript-kirjasto, joka on mullistanut tavan luoda web-sovelluksien käyttöliittymiä. Reactin suosio voidaan johtaa sen tehokkaaseen tapaan hallita sitä, mitä käyttöliittymässä tulee näkymään käyttäjille sekä sen modulaarisista ja helposti uudelleen käytettävistä komponenttirakenteista. Myös yksi Reactin suosituimmista piirteistä on sen virtuaalinen DOM (Dokumenttiobjektimalli), joka on nopea ja kevyt tapa päivittää käyttöliittymää muutosten tapahtuessa. (A-Team Global, 2023)

React julkaistiin alun perin vuonna 2011 Facebookin toimesta ja se on nykyään noussut yhdeksi suosituimmaksi front end kehitystyökaluksi. Facebookin ja muiden Metan omistuksessa olevien sovellusten kuten Instagramin lisäksi Reactia käyttää myös monet muut suuret yhtiöt, joiden sovelluksia tai palveluja miljoonat käyttävät päivittäin, kuten Netflix, Reddit, Pinterest ja Discord. (Medium, 2021; TatvaSoft, 2022)

Virtuaalinen DOM on kevyt versio oikeasta DOMista. Kun React-koodissa tekee muutoksia, React luo uuden virtuaalisen DOMin ja vertaa sitä edelliseen. Sen jälkeen se päivittää vain tehdyt muutokset oikeaan DOMiin, jonka ansiosta muutosten näkyminen käyttöliittymässä on paljon nopeampaa. Tämä myös johtaa käyttäjäkokemuksen paranemiseen sovelluksen nopeuden takia. (React, n.d.)

2.4.3 Android

Android on erittäin yleinen mobiilikäyttöjärjestelmä, joka perustuu Linux ytimeen eli kerneliin ja sitä tuottaa Google. Ensimmäinen Android-mobiililaite; T-Mobile G1, joka tunnetaan myös nimellä HTC Dream, julkaistiin vuonna 2008 ja se on siitä lähtien noussut kärkisijalle puhelimien, tablettien ja muiden mobiililaitteiden käyttöjärjestelmänä. Androidin suuri suosio perustuu sen joustavuuteen, muokattavuuteen ja avoimuuteen sekä siihen, että se toimii suurella määrällä eri valmistajien laitteita. (Britannica, 2022)

Yksi Androidin avainominaisuuksista on se, että se perustuu avoimeen lähdekoodiin, jonka ansiosta kehittäjät voivat muokata käyttöjärjestelmää vapaasti tarpeidensa mukaan. Tästä on seurannut suuri määrä kolmannen osapuolen sovelluksia sekä yhteisö, joka koostuu kehittäjistä ja harrastelijoista, jotka osallistuvat alustan päivittämiseen ja parantamiseen. (Android Developer Fundamentals, 2018)

Android oli kirjoitettu alun perin Java-ohjelmointikielellä, mutta vuodesta 2019 lähtien Google vaihtoi suositellun ohjelmointikielen Kotliniin, vaikkakin Java toimii edelleen yhtä hyvin. Näillä kielillä kirjoitetut sovellukset kootaan tavukoodiksi, jota suoritetaan DVM:llä, eli Dalvik Virtual Machinella, joka on Android-laitteilla sovellusten suorittamiseen tarkoitettu virtuaalikone. Lisäksi

Androidissa on suuri määrä kirjastoja ja API:ta, joiden kautta kehittäjillä on pääsy laitteen omiin ominaisuuksiin, kuten kamera, GPS ja sensorit. (New Media Aid, 2015)

2.4.4 Android Studio

Android Studio on Android-sovelluksien kehittämiseen tarkoitettu IDE. Sen ensimmäinen versio julkaistiin vuonna 2013 ja siitä on tullut nykyään suosituin työkalu Android-sovellusten kehittämiseen. Siitä löytyy kattava määrä juuri Androidille suunniteltuja työkaluja ja ominaisuuksia, kuten lähes minkä tahansa Android-käyttöjärjestelmän emulointi. (Android Developers, 2023a)

Android Studio on rakennettu Java-kehittämiseen luodun ohjelmistoympäristön; IntelliJ IDEA:n päälle. Se antaa käyttöön kaikki työkalut, mitä Android-sovellusten kehittämiseen tarvitsee, mukaan lukien koodieditorin, visuaalisen käyttöliittymäeditorin, debuggerin sekä nopean emulaattorin sovellusten testaamiseen kehitystyön ohessa. (Android Developers, 2023a)

Android Studio tukee pääasiallisesti Javaa, Kotlinia ja C++ ohjelmointikielinä. Tällä tavalla monet eri kehittäjät pääsevät nopeasti käyttämään Android Studiota ilman, että heidän välttämättä tarvitsee opetella täysin uutta kieltä. Android Studio myös tunnistaa automaattisesti kielen, jolla koodi on kirjoitettu, joten jos netistä löytyy sopiva koodin pätkä, jonka haluaa kopioida suoraan omaan koodiinsa, mutta se on kirjoitettu esimerkiksi Javalla, kun muu koodi on Kotlinia, Android Studio muuttaa liitetyn koodin Kotlin-kielelle. (Android Developers, 2023a)

3 REST-palvelu

REST (engl. REpresentational State Transfer) on web-palvelulle tietopalvelun rajapinta. Muutoin REST-palvelu on kuin normaali HTTP-palvelu, mutta paluuviestinä ei ole tyypillistä HTML-sivua, vaan takaisin tulee vastaus JSON-muodossa. Tämä vastaus on helppo tallentaa suoraan tietokantaan ilman muutoksia. Myöhemmin tätä dataa pystyy hyödyntämään helposti JavaScriptissä, jolla JSON:in tiedot luetaan tulevaa käyttöä varten. (Red Hat, 2020; Pilvikoodarin blogi, 2016)

Koska tätä opinnäytetyötä tehdessä ei ole mahdollisuutta saada yhteyttä oikeiden ihmisten henkilötietoihin ja ajanvarauksiin, joita sairaaloissa käytettäisiin, luodaan sitä varten C#-ohjelmointikielellä REST-palvelu henkilötietojen simuloimiseen. REST-palvelulla voi luoda uusia henkilöitä, muokata olemassa olevia henkilötietoja, poistaa henkilöitä tietokannasta tai vain yksinkertaisesti lukea henkilöiden tietoja. Kaikki tämä tapahtuu aluksi luomalla uudet henkilötiedot JSON-tiedostomuodossa, joka tallentuu tätä projektia varten luotuun tietokantaan. (Pilvikoodarin blogi, 2016)

3.1 Web API REST-palvelun luonti

Projektin taustalle tarvitaan jatkuvasti suoritettavaksi palvelu, josta voidaan hakea tarvittavat tiedot, kun henkilö ilmoittaa itsensä lääkärin vastaanotolle itseilmoittautumislaitteella. Opinnäytetyössä tällainen palvelu luodaan Microsoftin Visual Studiolla, jossa käytetään ASP.NET ja verkkokehittämis-komponenttipakettia. Luodessa siis uutta projektia Visual Studiolla, valitaan ASP.NET Core Web API -mallipohja, jonka jälkeen varmistetaan, että projektin Framework on .NET 7.0 tai myöhempi ohjelman toimivuuden varmistamiseksi. Koska kyseessä on MVC-arkkitehtuuriin (Model-view-controller) perustuva web API, pitää myös varmistaa, että Use controllers -valintaruutu on valittuna. (Microsoft, 2023a)

Visual Studio luo projektin mukana automaattisesti malli API:n, johon on tallennettu valmiiksi JSON-muotoon muutaman päivän säätiedot vuodelta 2019. Tätä dataa pääsee tarkastelemaan painamalla Ctrl + F5, jolloin Visual Studio käynnistää sovelluksen ilman debuggeria. Visual Studio

kysyy tässä vaiheessa, että luotetaanko projektiin, koska sillä ei ole SSL-sertifikaattia, jolloin HTTP-protokollaa ei voi suojata. Hyväksymällä kaksi turvallisuusvaroitusta, pääsääntöiseen nettiselaimeen aukeaa Swagger-nimisen API:n kehityskaluston tukemana uusi nettisivu. (Microsoft, 2023a)

Kun Visual Studiolla valitaan ASP.NET Core Web API -mallipohja, se luo samalla myös tarvittavan CRUD:n, jolla säätietoja voi lisätä, lukea, muokata tai poistaa halutessa. Swaggerissa nämä kaikki ovat kätevästi näkyvillä käyttöliittymässä, joka tekee JSON:n muokkaamisesta erittäin helppoa. (Microsoft, 2023a)

CRUD on neljästä operaatiosta koostuva akronyymi, joka tulee sanoista create, read, update ja delete. Sitä käytetään ohjelmoinnissa kuvaamaan näitä neljää edellä mainittua toimintoa, joilla esimerkiksi tietokantaan voidaan tehdä muutoksia. Tarkempi selitys jokaisesta CRUD:n toiminnosta esitetään taulukossa 1. (FreeCodeCamp, 2022)

Taulukko 1 CRUD-operaatiot ja niitä vastaavat HTTP-metodit. (FreeCodeCamp, 2022)

CRUD-operaatiot	**HTTP-metodit**	**Selvitys**
Create	POST	Tietojen lisääminen tietokantaan.
Read / Retrieve	GET	Tietojen lukeminen tietokannasta.
Update	PUT / PATCH	Valmiiden tietojen päivittäminen tietokannassa.
Delete	DELETE	Tietojen poistaminen tietokannasta.

3.2 Oman tietokannan luonti

Koska projektia varten tarvitsemme käyttöön jotain muita tietoja, kuin säätietoja neljän vuoden takaa, pitää nämä uudet tiedot luoda itse. Tätä varten tässä luvussa kerrotaan, kuinka Microsoft Visual Studiossa luodaan uusi tietokantakonteksti.

3.2.1 Tietokantakontekstin luonti Microsoft Visual Studiossa

Visual Studiossa on oikealla Solution Explorer, jossa näkyy kaikki projektiin sisältyvät kansiot ja tiedostot. Kansioiden joukossa on Models-kansio, johon luodaan tietokantakontekstissa käytettävä tiedosto, joka sisältää luokan. Tässä tapauksessa luodaan luokka sairaalan asiakkaille, eli potilaille. Hyvien nimeämisperiaatteiden mukaan luokka kannattaa nimetä englanniksi, eli Patients.cs. (Microsoft, 2023a)

Kun uusi luokka on luotu, sinne lisätään ohjelmakoodi 1.

Ohjelmakoodi 1 Parametrien luonti potilasluokkaan. (Microsoft, 2023a)

```
namespace AttuneIlmoBack.Models
{
      public class Patients
      {
           public long Id { get; set; }
           public string Name { get; set; } = null!;
           public string BarCode { get; set; } = null!;
           public bool IsCheckedIn { get; set; }
           public string? Description { get; set; }
           public string Room { get; set; } = null!;
           public DateTime StartTime { get; set; }
           public long QueueNumber { get; set; }
      }
}
```

Luokkaan lisätään tarvittavat tiedot, mitä kuittiin halutaan tulostaa, eli toimenpiteelle varattu huone, ohjeet kyseiseen huoneeseen pääsemiseen, ajanvarauksen alkamisen aika sekä vuoronumero. Lisäksi koodissa on henkilön tiedot tunnistamista varten, vaikkei niitä kuittiin tulostetakaan sekä viivakoodille oma parametri, jonka avulla potilaat ja heille varatut ajat pystytään suodattamaan henkilökortin skannauksen yhteydessä. (Microsoft, 2023b)

Tarvittaessa tähän lisättäisiin myös kartalle oma kuva tietotyyppi, esim. ”BLOB”, joka kuitenkin jää opinnäytetyöstä pois varattuna mahdolliselle jatkokehittämiselle. (Binary Large OBject) (MySQL, 2023)

Seuraavaksi Solution Exploreriin tehdään kokonaan uusi kansio tietokantakontekstia varten. Tässä tapauksessa se nimetään DB-nimiseksi eli database (tietokanta). Sen sisälle luodaan jälleen uusi luokka, jonka nimeksi annetaan ApplicationDBContext.cs. Kun uusi tietokantakontekstiluokka luodaan, se sisältää automaattisesi ohjelmakoodi 2:n. (Microsoft, 2023a)

Ohjelmakoodi 2 ApplicationDBContext-luokka, joka ottaa käyttöönsä aiemmin luodun Patients-luokan parametrit. (Microsoft, 2023a)

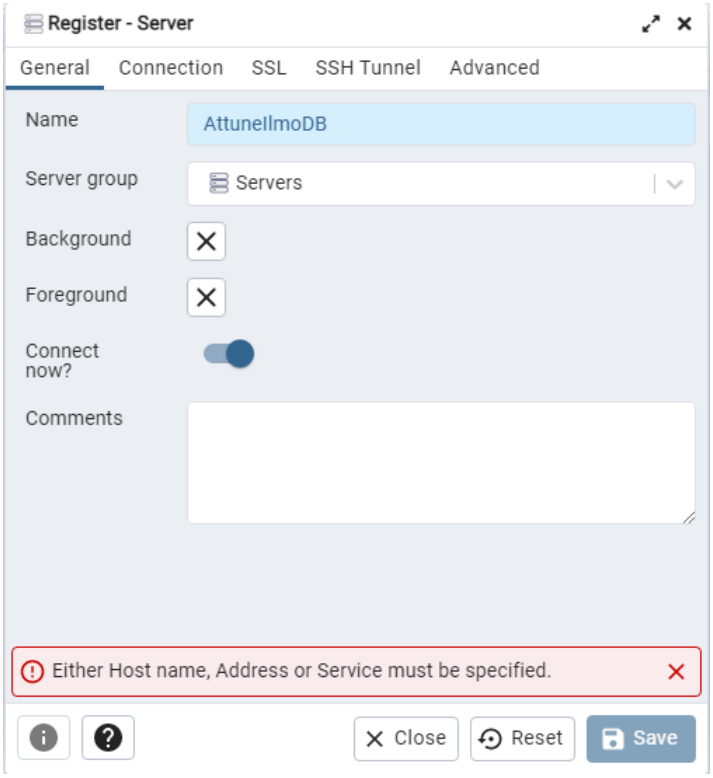
```
using AttuneIlmoBack.Models;
using Microsoft.EntityFrameworkCore;

namespace AttuneIlmoBack.DB
{
	public class ApplicationDBContext : DbContext
	{
		public ApplicationDBContext
		(DbContextOptions<ApplicationDBContext> options) :
		base(options) {}

		public DbSet<Patients> Patients { get; set } = null!;
	}
}
```

Ohjelmakoodi 2 määrittää tietokantakontekstiluokan palvelimelle, jonka avulla ohjelma kykenee olemaan yhteydessä Patients-tietokantataulun kanssa. (Microsoft, 2023a)

3.2.2 Uusi palvelin ja tietokanta PostgreSQL:llä

Uutta tietokantaa varten luodaan palvelin pgAdmin 4 käyttöliittymässä PostgreSQL:ään. Käyttöliittymän vasemmassa reunassa näkyy Servers, jonka kautta voi lisätä uuden palvelimen, josta aukeaa kuvan 1 mukainen näkymä. (The pgAdmin 4 Development Team, 2023, s. 99)

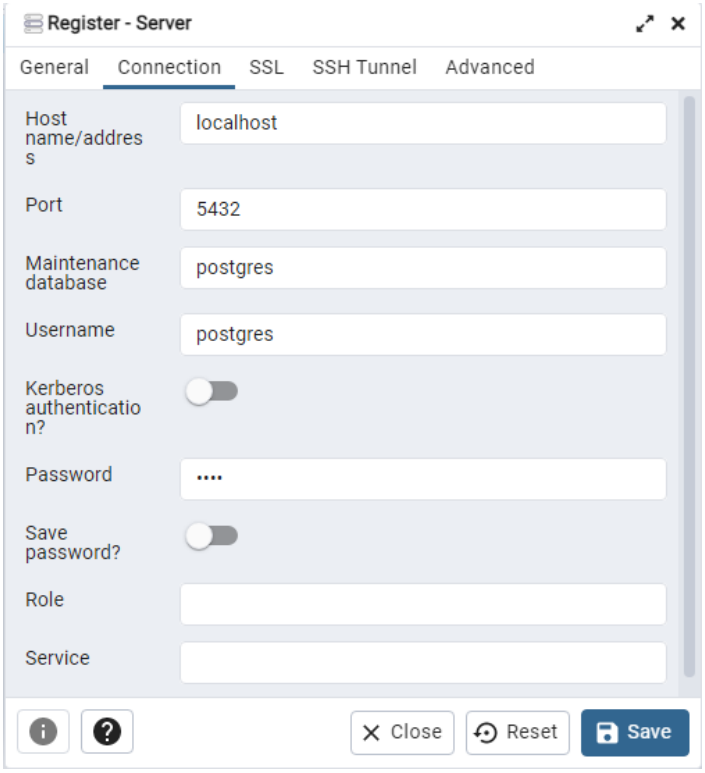
Kuva 1 Palvelimen nimeäminen ja muut yleiset tiedot.



Uutta palvelinta luodessa ensimmäisenä avautuu kuvassa 1 näkyvä ikkuna, johon täytetään yleiset tiedot, kuten palvelimen nimi ja mihin palvelinryhmään palvelin tulee kuulumaan. Nimellä ei ole väliä, kunhan se on tunnistettavissa ja palvelinryhmänä toimii Servers, johon alun perin tätä palvelinta alettiin luomaan. Background ja foreground määrittävät värin palvelimelle, jotka voi tässä tapauksessa jättää vakioksi, eli tyhjiksi. Myös connect now? -valintaruutu voidaan jättää päälle, jolloin ohjelma yrittää automaattisesti yhdistää palvelimeen, kun sen luominen on valmista. Jos käytössä on monia palvelimia ja monet ihmiset pääsevät käsiksi niihin, on joskus hyvä jättää myös kommentti selvittämään, mikä kyseisen palvelimen tarkoitus on. (The pgAdmin 4 Development Team, 2023, s. 100–101)

Palvelimen luontiprosessissa avatussa ikkunassa on yläreunassa viisi välilehteä, joista ensimmäisenä on automaattisesti aukeava general. Seuraavana on connection, josta päästään muokkaamaan tietoja, joiden avulla palvelimeen saadaan yhteys. Tämä yhteys toimii niin pgAdminissa, kuin sen ulkopuolellakin. Kyseinen välilehti näkyy kuvassa 2.

Kuva 2 Palvelimeen yhdistämiseen tarvittavat tiedot.



Kuvassa 2 ensimmäisenä määritetään palvelimelle IP-osoite, tai isäntänimi/-osoite. Koska tässä tapauksessa ei ole tarvetta päästä käsiksi kuin vain paikalliseen palvelimeen, se voidaan nimetä localhostiksi. Seuraavana listassa näkyy portin numero, joka on PostgreSQL tietokantasysteemissä vakiona 5432. (Apple, 2022) Portin numeron saa itse määrittää haluamakseen, kunhan kyseinen numero ei ole jo jonkun muun palvelun käytössä. Tässä projektissa tullaan käyttämään kahta muuta porttia numeroilla 3000, joka on React-kehityspalvelimen oletus portti ja 5140, joka on Swaggerin oletus HTTP-portti, joten tämän palvelimen portin voi jättää myös alkuperäiseksi.

Kaksi seuraavaa kohtaa ovat vakiona nimetty postgres-nimellä, joka kelpaa tähän projektiin hyvin, mutta isoimmissa projekteissa kannattaa käyttää sopivampia nimiä. Maintenance database -kenttä määrittää nimen alustavalle tietokannalle, mihin ohjelma yhdistää. Username-kenttä on nimelle, jota käytetään todentamisessa, kun palvelimelle haetaan yhteyttä. (The pgAdmin 4 Development Team, 2023, s. 101)

Palvelimelle kannattaa antaa salasana, jotta sinne ei pääse käsiksi ihan kuka tahansa, projektia varten yksinkertaisuuden vuoksi salasanaksi valitaan ”1234”. Seuraava valintaruutu määrittää, tallennetaanko annettu salasana tulevaisuutta varten. (The pgAdmin 4 Development Team, 2023, s. 101)

Seuraavalla välilehdellä voidaan määrittää tarvittaessa erilaisia parametreja yhteydelle, kuten salasanalle oma tiedosto ja sen sijainti, mutta tässä projektissa niitä, tai muidenkaan välilehtien asetuksia ei tarvitse muuttaa. (The pgAdmin 4 Development Team, 2023, s. 102)

Kun uusi palvelin on tallennettu, sen sisään luodaan uusi tietokanta, samalla tavalla vasemmalla olevasta listasta. Tästä avautuu uusi ikkuna samalla tavalla, kuin palvelinta luodessa ja sen ensimmäisellä välilehdellä on kolme muokattavaa asetusta. Ensimmäisenä tietokannalle annetaan nimi, joka tässä tapauksessa voi olla vaikka ”PatientList”. Tällä pystytään kertomaan tulevalle Web API:lle, että localhostista pitäisi löytyä PatientList niminen tietokanta. Seuraavana listassa on pudotusvalikko, josta valitaan omistaja tälle tietokannalle, ja vakiona oleva ”postgres” on hyvä tähän projektiin. Viimeisenä tietokannalle voi jälleen antaa kommentin tulevaisuutta varten, jos sen kokee tarpeelliseksi. Seuraavilla välilehdillä olevia tietoja ei tarvitse tässä projektissa muuttaa. (The pgAdmin 4 Development Team, 2023, s. 134–135)

3.2.3 Yhteyden muodostaminen tietokantaan

Tietomallien muuttuessa, kun uusia ominaisuuksia lisätään ja poistetaan, tietokantamalleja pitää muuttaa samalla asianmukaisesti. Migraation avulla tietokantaa pystyy päivittämään sovelluksen tietomallin mukaan samalla säilyttäen vanhat olemassa olevat tiedot. Kun tietomalliin tulee muutoksia, kehittäjä voi käyttää hyödykseen Microsoftin Entity Framework Coren (lyh. EF Core) työkaluja lisätäkseen vastaavan migraation pitääkseen tietokantamallin synkronoituna. EF Core vertaa nykyistä mallia vanhaan malliin ja toteaa niiden erot ja luo migraatiolle lähdetiedostot ja lisää ne projektin muiden tiedostojen sekaan mahdollista myöhempää tutkimista varten. Kun uusi migraatio on luotu, se voidaan lisätä tietokantaan seuraavalla tavalla. (Microsoft, 2023b)

Kun Microsoft Visual Studio luo Web API:n, se luo myös automaattisesti Program.cs-nimisen tiedoston, jonka avulla aikaisemmin mainittu Swagger käynnistyy. Tätä koodia muokkaamalla saamme myös oman tietokantamme sisällöt näkyviin Swaggeriin.

Ohjelmakoodi 3 Program.cs-tiedoston sisältöä. (Microsoft, 2023a)

```
using Microsoft.EntityFrameworkCore;
using AttuneIlmoBack.DB;

var MyAllowSpecificOrigins = "_myAllowSpecificOrigins";

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddCors(options =>
{
      options.AddPolicy(name: MyAllowSpecificOrigins,
                  policy =>
                  {
                              policy.WithOrigins(
                              "http://xxx.xxx.x.xxx:3000",
                              "http://localhost:3000")
                              .AllowAnyHeader()
                              .AllowAnyMethod();
                  });
});
```

Ohjelmakoodissa 3 tuodaan ensin tarvittava nimiavaruus (namespace) EF Corelle, jonka jälkeen tuodaan Patients.cs-tiedoston parametrit tietokantakontekstin kautta Program-tiedoston käyttöön. (Microsoft, 2020)

Seuraavaksi luodaan uusi ”WebApplicationBuilder” -instanssi, johon syötetään ”args”-parametri, joka toimii komentorivi argumenttina. (Microsoft, 2022a)

Tämän jälkeen koodiin lisätään palveluita konttiin (container) kutsumalla AddCors()-metodin WebApplicationBuilder-esiintymän Services-ominaisuutta. Options-konfiguraation lambdailmaisu syötetään tähän metodiin parametrina. (Savonia, n.d.; Microsoft, 2023c)

Lopulta lambdailmaisun myötä AddPolicy()-metodi kutsutaan määrittämään CORS (Cross-Origin Request) käytäntö, joka nimettiin alussa MyAllowSpecificOrigins. Tämä käytäntö sallii pyynnöt kahdesta eri alkuperästä. Ensimmäinen näistä alkuperistä koostuu nettiyhteyden IP-osoitteesta, eli x:t korvattaisiin numeroilla. Tällä tavalla yhteys toimii miltä tahansa laitteelta, joka on yhteydessä

samassa verkossa, kuin tietokone, jolta tätä palvelinta suoritetaan. Sen sijaan localhost toimii vain paikalliselta laitteelta. IP-osoitteen perässä on kaksoispisteen jälkeen numero 3000, joka on myöhemmin luotavan React-käyttöliittymän vakio porttinumero, jota kautta React yhdistää kehityspalvelimelle. Tämän päätteeksi annetaan kaikille HTTP-otsikoille (headers) ja -metodeille lupa ottaa yhteyttä sivuun pyynnön yhteydessä. Myös AllowAnyOrigin()-metodi olisi toiminut localhostin ja IP-osoitteen sijaan, mutta tässä tilanteessa ei haluta, että kuka tahansa pääsee mistä tahansa omalla laitteellaan tietokantaan. (Microsoft, 2023d)

Lopuksi koodi päivitetään ohjelmakoodin 4 mukaisesti.

Ohjelmakoodi 4 Yhteyden muodostus tietokantaan. (Microsoft, 2023e)

```
builder.Services.AddDbContext<ApplicationDBContext>(conf =>
      conf.UseNpgsql("Server=localhost;" +
      "Database=PatientList;" +
      "Username=postgres;" +
      "Password=1234");
```

Viimeisenä tällä koodilla käytetään EF Corea lisäämään tietokantakonteksti ASP.NET Core sovellukseen. Aiemmin luodun builder-instanssin AddDbContext-metodia kutsutaan rekisteröimään ohjelman käytettäväksi oma tekemä ”ApplicationDBContext”, joka lukee tarvittavat parametrit sairaalan asiakkaita varten. Sen jälkeen lambdailmaisulla tarkennetaan, että kontekstin tulisi käyttää PostgreSQL tietokantaa, jonka palvelimen sijainti on localhostissa nimellä PatientList, joka määritettiin aiemmin. Koodilla myös kerrotaan, että tietokantaan saa yhteyden käyttäjänimellä ”postgres” ja salasanalla ”1234”, jotka määritettiin aiemmin pgAdmin 4:ssä, kun uutta palvelinta luotiin. (Npgsql, n.d.; Microsoft, 2023e) Oikeassa tilanteessa näitä tietoja ei kannata jättää näkyviin tähän, vaan ne kannattaa lukea jostain muualta, kuten XML- tai JSON-tiedostosta. (Microsoft, 2022c)

Tämän jälkeen kaikki on valmiiksi luotua koodia, jossa ainoa muutos, joka pitää itse tehdä, on poistaa, tai suositellummin kommentoida ”app.UseHttpsRedirection();”-rivi. Tällöin ohjelma ei suorita Swaggeria HTTPS-protokollan kautta, vaan käyttää suoraan HTTP-protokollaa, jolloin tulevaisuudessa saamme helposti yhteyden Android Studiosta tietokantaamme ilman sertifikaatteja. Tässä tapauksessa sovelluksesta tulee riskialtis hyökkäyksille, koska HTTP-verkkoliikenne ei ole salattua, mutta tässä väliaikaisratkaisussa sillä ei ole väliä. (Microsoft, 2022b)

3.3 Web API -ohjaimien päivittäminen

ASP.NET web -sovelluksiin voi lisätä helposti valmiiksi luodun koodin, jonka avulla tietomallien kanssa pystyy olemaan yhteyksissä. Tätä koodia kutsutaan scaffold-termillä englanniksi, suomeksi tälle ei ole oikeaa virallista termiä. Sillä luodaan uusi luokka, jossa on API-ohjain ominaisuus, joka osoittaa, että ohjain vastaa web API-pyyntöihin. (Microsoft, 2022a)

Visual Studiolla tällaisen luokan luominen onnistuu hyvin helposti, kun projektia tehdessä automaattisesti luotuun Controllers-kansioon lisää uuden Scaffolded Itemin. Sen jälkeen valitaan ”API Controller with actions, using Entity Framework”, johon valitaan malliksi aikaisemmin ohjelmakoodissa 1 luotu Patients-luokka ja tietokontekstiksi ApplicationDBContext-luokka. Tämän lisäämällä generoituu koodi, joka merkitsee luokan [ApiController]-ominaisuudella, joka kertoo ohjelmalle, että luokka ja kaikki siitä periytyvät datatyypit tarjoavat HTTP API-pyyntöjä, joka näytetään ohjelmakoodissa 5. (Microsoft, 2023a)

Ohjelmakoodi 5 Web API -ohjaimen kontekstin määritys sekä tietokannasta jokaisen potilastiedon hakemiseen tarkoitettu funktio. (Microsoft, 2023a)

```
[Route(“api/[controller]”)]
[ApiController]
public class PatientsController : ControllerBase {
      private readonly ApplicationDBContext _context;

      public PatientsController(ApplicationDBContext context) {
                  _context = context;
      }

      [HttpGet]
      public async Task<ActionResult<IEnumerable<Patients>>> GetPatients()
{
                  if (_context.Patients == null)
                  {
                              return NotFound();
                  }
                  return await _context.Patients.ToListAsync();
      }
```

Ohjainluokkaan generoidussa koodissa aluksi luodaan HTTP-päätepisteen reitti, eli mitä esimerkiksi normaalin nettisivun URL-osoitteeseen tulee verkkotunnuksen ja kauttamerkin jälkeen. Tässä tapauksessa reitti on määritetty ”api/Patients”, koska ohjaimen nimi on PatientsController. (Microsoft, 2022e)

API-ohjain merkinnän jälkeen peritään kaikki yleisimmät ohjaintoiminnot ControllerBase-luokasta PatientsController-nimiseen luokkaan, jonka sisällä ensin annetaan ohjaimelle pääsy luotuun tietokontekstiin, jotta toimintojen suorittaminen tietokannassa onnistuu mutkitta. Tämän jälkeen luodaan ensimmäinen varsinainen toiminto ohjaimelle, joka etsii tietokannasta kaikki sinne tallennetut rivit, eli jokaisen potilaan henkilötiedot ja palauttaa ne listana JSON-muodossa. (Microsoft, 2022a)

Kuitenkin tätä projektia varten pitää luoda itse yksi uusi toiminto, joka tuo esiin ainoastaan sen potilaan tiedot, joka ilmoittautuu automaatilla sisään. Tämä toiminto luodaan ohjelmakoodissa 6.

Ohjelmakoodi 6 Potilaan suodatus tietokannassa viivakoodin mukaan. (Microsoft, 2023a)

```
[HttpGet("{barcode}")]
public async Task<IActionResult> GetPatient(string barcode) {
      var patientList = from a in _context.Patients orderby a.Id descending
      select a;

      if (!String.IsNullOrEmpty(barcode)) {
                patientList =
      (IOrderedQueryable<Patients>)patientList.Where(
                i => i.Barcode.Equals(barcode));
      }
      if (patientList == null)
                {
                          return NotFound();
                }
                return Ok(await patientist.ToListAsync());
      }
```

Ohjelmakoodissa 6 määritetään aluksi uusi reitti sivulle, joka tässä tapauksessa tarkoittaa, että mikä ikinä potilaan viivakoodi onkaan, se tulee edellisessä kohdassa mainitun reitin perään, eli tässä tapauksessa ”api/Patients/viivakoodi”. (Microsoft, 2022e)

Tämän jälkeen luodaan uusi muuttuja nimeltään patientList, johon tallennetaan kaikkien potilaiden tiedot Patients-luokassa määritetyn ID:n mukaan laskeutuvaan järjestykseen. Sen jälkeen koodi vertaa sille annettua viivakoodia tietokantataulussa olevien potilaiden Barcode-muuttujien kanssa, jonka jälkeen se tallentaa osumat patientList-muuttujaan ja lopulta palauttaa ne jälleen listassa JSON-muodossa. (Microsoft, 2022a)

Web API -ohjaimen luontiprosessissa tulee mukana myös automaattisesti CRUD:in mukaiset toiminnot tietojen lisäykselle, muokkaamiselle ja poistamiselle, mutta niille ei tarvitse tehdä tässä tilanteessa muutoksia.

3.4 Tietojen siirto tietokantaan

Kun palvelin ja tietokanta on luotu PostgreSQL:ään ja kaikki tarvittava koodi on kirjoitettu, on aika siirtää Visual Studiossa luodut parametrit pgAdmin 4:n. Siihen on helppo tapa, joka tapahtuu Visual Studion terminaalin kautta, joka löytyy ohjelman alareunasta. Jos terminaalia ei näy, sen saa auki Windowsilla vakiona Ctrl + Ö näppäinyhdistelmällä.

Ennen kuin terminaaliin kirjoitettavat komennot datamigraatiota varten saadaan toimimaan, pitää asentaa paketteja projektiin Visual Studiossa. Asentaminen hoituu helposti Visual Studion sisältä, joko terminaali komennolla, tai NuGet Package Managerin kautta. (Microsoft, 2023f)

Komento 1 Visual Studio PowerShell terminaalikomento PostgreSQL-paketin asentamiseen. (Microsoft, 2023b)

```
Install-Package Microsoft.EntityFrameworkCore.Tools
Install-Package Npgsql.EntityFrameworkCore.PostgreSQL
```

Komento 1 sisältää kaksi komentoa, joista ensimmäinen asentaa paketin ja kaikki tarvittavat riippuvuudet, joka mahdollistaa EF Core komentorivityökalut, joihin lukeutuu myös dotnet-komennot, jota käytetään pääasiallisesti tietokantojen migraatioissa ja tietokantakontekstien luonnissa. (NuGet, 2023a)

Seuraava komento asentaa paketin, joka tarjoaa EF Coreen PostgreSQL-tuen. Tämä varmistaa, että EF Core pystyy kommunikoimaan PostgreSQL-tietokannan kanssa tarvittavien asennettujen luokkien ansiosta. (NuGet, 2023b)

Vaihtoehtoisesti PostgreSQL-paketin pystyy asentamaan myös korvaamalla komennon alun ”dotnet add package”:lla, jonka käyttö mahdollistettiin juuri komennolla 1. Visual Studiossa näillä

komentomuodoilla ei ole juurikaan eroa, mutta dotnet on myös käytettävissä muuallakin, kuin Visual Studiossa. (Microsoft, 2022f; Microsoft, 2022g)

Seuraavaksi terminaaliin syötetään komento 2, joka aloittaa migraatioprosessin.

Komento 2 Visual Studio PowerShell terminaalikomento datamigraation luomiseen. (Microsoft, 2023b)

```
Add-Migration InitialCreate
```

Komennolla 2 EF Core luo uuden kansion oikealla näkyvään Solution Exploreriin nimeltään Migrations. Sinne generoituu uusi C#-luokka, jonka nimessä on aikaleima sekä komennossa annettu nimi. Tässä tapauksessa migraation nimenä on InitialCreate kertomaan, että kyseessä on alkuperäinen migraatio. (Microsoft, 2023b)

Tämän jälkeen migraatio pystytään yhdistämään tietokantaan komennolla 3.

Komento 3 Tietokannan päivittäminen nykyisillä tiedoilla. (Microsoft, 2023b)

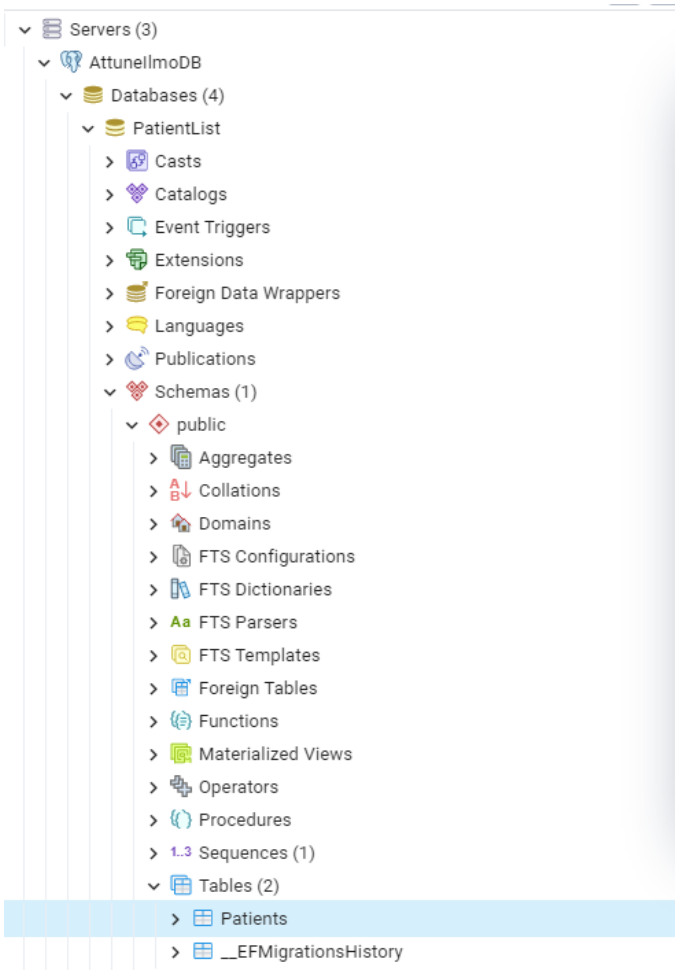
```
Update-Database
```

Komento 3 ottaa viimeisimmän migraation tiedot, eli tähän asti luotu ja yhdistää ne tietokantaan, joka asetettiin ohjelmakoodissa 4. Sen jälkeen kaikki ohjelmakoodin 1 eli Patients.cs-luokan parametrit ovat nähtävissä pgAdmin 4:ssä. Kun pgAdmin 4:ssä olevan PatientList-tietokannan päivittää, sen sisältä löytyy uusi tietokantataulukko, joka on saanut nimensä Patients.cs-luokasta. Tämä taulukko löytyy kuvan 3 osoittamasta polusta. (Microsoft, 2023b)

Tulevaisuudessa, jos tietokantamallia täytyy päivittää, se onnistuu helposti tekemällä halutut muutokset esimerkiksi Patients.cs-luokkaan, jonka jälkeen terminaaliin annetaan uudestaan samanlainen komento, kuin komento 2, paitsi komennon jälkimmäinen osuus, eli nimi, muutetaan esimerkiksi kuvaamaan tehtyjä muutoksia. Tällöin EF Core vertaa uutta päivitettyä mallia vanhan mallin tilannekuvaan, joka löytyy Migrations-kansiosta. Vertailussa havaittuihin eroihin perustuen EF Core lisää tarvittavan migraation. Tämän jälkeen terminaaliin annetaan komento 3 uusiksi ja EF Core päivittää PostgreSQL:ssä olevan tietokantataulukon sopivaksi. (Microsoft, 2023b)

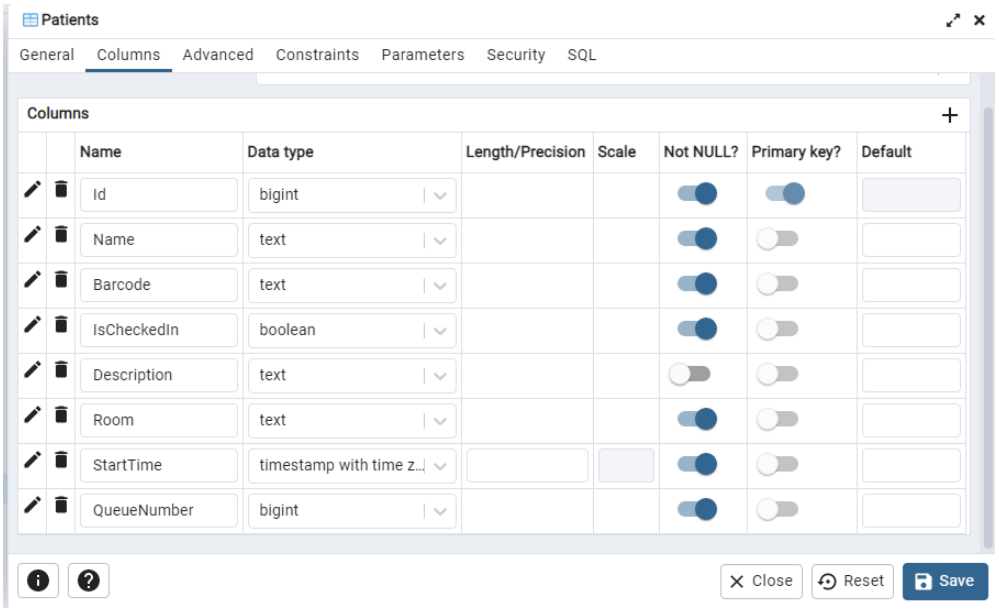
Kuvassa 3 korostetussa kohdassa olevan taulukon ominaisuuksia pystyy tarkastelemaan klikkaamalla taulukkoa hiiren oikealla painikkeella ja valitsemalla Properties-napin avautuneen listan pohjalta, jolloin aukeaa uusi ikkuna, jonka ensimmäisellä välilehdellä näkyy kentät, joista taulukon nimeä, omistajaa, kaavaa ja muuta voi muuttaa.

Kuva 3 Patients-taulukko hakemistopuussa pgAdmin 4:ssä.



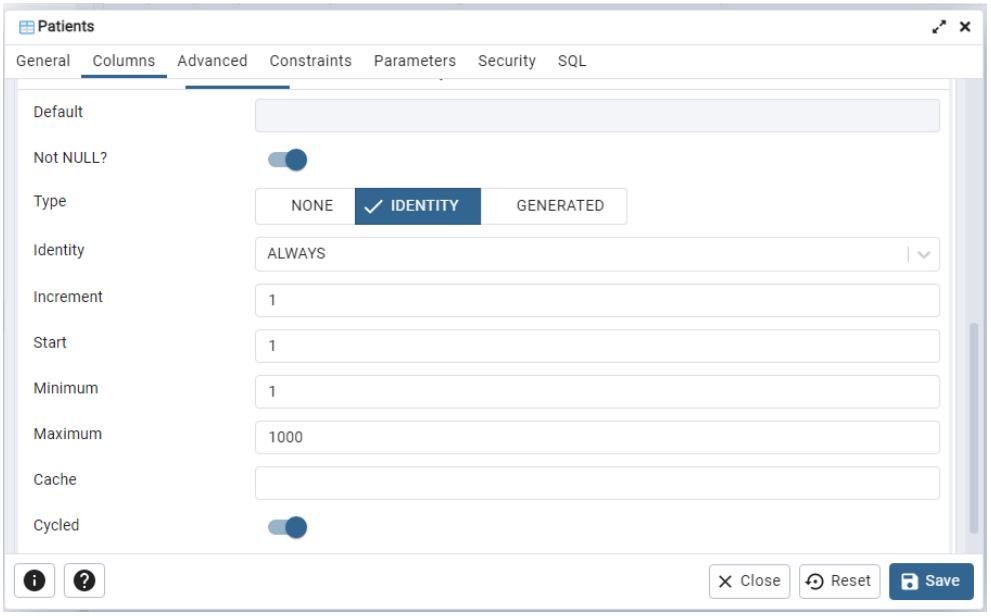
Seuraavalla välilehdellä näkyy taulukon parametrit, joista muodostuu sarakkeet taulukolle, kun tietokantaan syötetään uutta tietoa, niin kuin kuvassa 4 näkyy.

Kuva 4 Patients.cs-luokan parametrit, joista taulukon sarakkeet muodostuvat.



Tietokantataulukon sarakkeet koostuvat aikaisemmin ohjelmakoodissa 1 luoduista parametreistä. PgAdmin 4 tunnistaa Id-parametrin automaattisesti, jolloin se tekee siitä viiteavaimen. Myös kaikki datatyypit ovat menneet kuvan 4 mukaan oikein, joten ainoa lisäys, mikä voidaan tehdä manuaalisesti, on QueueNumber-parametrin asettaminen kasvamaan yhdellä joka kerta, kun taulukkoon syötetään uutta tietoa, eli tässä tapauksessa uusi varattu aika potilaalle. Sarakkeiden listassa vasemmalla puolella on kynän kuva ja tätä klikkaamalla valitun sarakkeen alapuolelle aukeaa uusia välilehtiä ja kenttiä muokattavaksi. Ensimmäisenä avautuu yleinen välilehti, josta sarakkeen nimeä voi vaihtaa ja sitä voi tarvittaessa kommentoida. Seuraavana on Constraints-välilehti, joka näkyy kuvassa 5.

Kuva 5 QueueNumber-parametrin automaattinen kasvaminen joka rivin lisäyksellä.



Constraints-välilehdellä sarakkeen tyypin pystyy vaihtamaan tyhjästä joko identiteetiksi tai generoiduksi. Jos sarakkeen tyyppinä on identiteetti, tietokanta luo automaattisesti tälle sarakkeelle uuden arvon seuraavien kenttien sääntöjen mukaan. Kuvassa 5 näkyy seuraavana kenttänä nousu, joka kertoo, kuinka suurella luvulla sarakkeen arvo nousee jokaisen uuden rivin kohdalla. Koska kyseessä on jonotusnumero, on hyvä, että arvo nousee joka kerta yhdellä. Tämän kentän jälkeen tulee aloitusluku, joka voidaan myös aloittaa yhdestä. Seuraavana on minimiarvo, joka voidaan määrittää samaksi, kuin aloitusnumero. Viimeinen muokattava kenttä on maksimiarvo. Tässä tapauksessa ei tarvitse odottaa, että sairaalassa on samaan aikaan odottamassa vuoroaan yli 1000 ihmistä, joten maksimiksi voidaan asettaa 1000. Kuvassa 4 pohjalla ”Cycled”-valintaruutu on asetettu päälle, jolloin maksimiarvon saavuttaessa parametrin numerot alkavat uudestaan nousemaan minimiarvosta. (The pgAdmin 4 Development Team, 2023, s. 292–293)

Generoituna näkyviin tulee uusi kenttä, johon voi kirjoittaa oman funktion ilmaisuna. Tämä ilmaisu voi viitata taulukon muihin sarakkeisiin, kunhan ne eivät ole myös generoituja. Myöskään toisiin tauluihin ei saa viitata tällä. Esimerkiksi ilmaisuna toimisi ”Id + 100”, jolloin QueueNumber olisi aina 100 numeroa isompi, kuin saman rivin Id-parametri. Tässä projektissa kyseistä tyyppiä ei tulla kuitenkaan käyttämään. (The pgAdmin 4 Development Team, 2023, s. 293)

3.5 Potilastietojen lisääminen tietokantataulukkoon

Seuraavana vaiheena on potilaiden luonti tietokantaan. Koska tässä projektissa on vain yksi taulukko käytössä, potilaiden tiedot ja ajanvaraukset ovat kaikki samassa taulukossa, mutta normaalisti olisi järkevintä olla taulukko sairaalan potilaille ja lisäksi oma taulukko varatuille ajoille. Koska PostgreSQL on relaatiomalliin pohjautuva tietokanta, eli relaatiokanta, sen sisällä olevia taulukoita voi yhdistää toisiinsa uniikeilla viitteillä. Tässä projektissa henkilöitä suodatetaan henkilökortin viivakoodin avulla, se voisi toimia viiteavaimena eri taulukoiden väleillä, jolloin yhdellä viivakoodilla voisi viitata moneen varattuun aikaan, eli jokaiseen aikaan, joka tämän henkilökortin omistajalle on varattu, kun taas jokaisella ajan varauksella voisi olla viite vain yhteen viivakoodiin.

Visual Studiossa avataan uudestaan Swagger UI uudessa selaimessa ilman debuggeria painamalla Ctrl + F5 näppäinyhdistelmää. Tällä kertaa Swaggeriin on tullut uusia ominaisuuksia, nimittäin luvussa 3.3 luodut Web API -ohjaimet potilaiden lisäämiseen (post), poistamiseen (delete), muokkaamiseen (put) ja hakemiseen (get). Avaamalla potilastietojen lisäämiseen tarkoitettu sarake näkyviin tulee ruutu, jossa on Try it out -nappi, jota painamalla aukeaa kenttä, jossa on JSON-muodossa ohjelman parametrit, kuten kuvassa 6 on näkyvillä.

Kuva 6 Uusi potilas ja ajanvaraus tälle Swagger UI:ssa.

```
{
  "id": 0,
  "name": "Olli Mappilas",
  "barcode": "string",
  "isCheckedIn": false,
  "description": "Käänny vasemmalle",
  "room": "Huone 252",
  "startTime": "2023-02-27 11:56:12.672+02",
  "queueNumber": 0
}
```

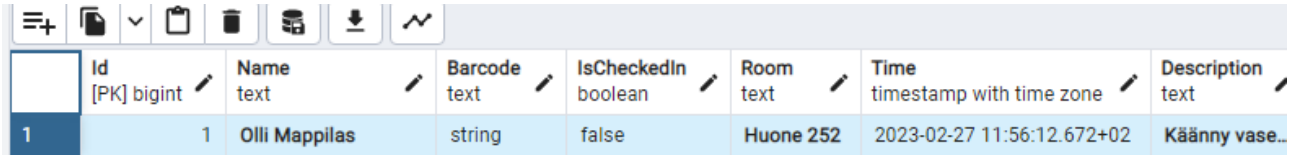
Kuvassa 6 potilaalle annetaan nimi, viivakoodi string-muodossa, kulkuohjeet huoneeseen ja huoneen nimi. Loput parametrit täyttyvät itsestään tarvittaessa. JSON:ssa pitää olla tarkkana, että jokainen parametri on heittomerkkien sisällä, jonka jälkeen tulee kaksoispiste ja sen jälkeen taas heittomerkkien sisään haluttu tieto, jos kyseessä on string tai muu vastaava datatyyppi. Pelkkien

numeroiden ja boolean-arvojen kohdalla ei tarvita jälkimmäisiä heittomerkkejä. Jokaisen paitsi viimeisen rivin lopussa tulee myös olla pilkku. Kaikkea tätä ympäröi aaltosulut. (HubSpot, 2022)

Kun luotavan potilaan tiedot ovat kohdillaan, JSON-kentän alapuolella on suuri Execute-nappi, jota klikkaamalla Swagger lähettää pyynnön, joka sisältää muokatun JSON:n määritettyyn päätepisteeseen, eli juuri luotuun PostgreSQL tietokantaan. Kun tieto on lähetetty, ohjelma odottaa vastausta palvelimelta, joka ilmoitetaan statuskoodilla, jotka ovat yleensä joko 200 onnistumiselle tai 400 epäonnistumiselle. Kun palvelimelta on saatu vastaus, Swagger näyttää tekstikentän alapuolella ”Response Body”-osion, jossa näkyy HTTP-pyynnön yksityiskohdat. Vastauksen epäonnistuessa tämä on hyvä työkalu selvittämään missä vika on. (Microsoft, 2023a) Monesti vika saattaa olla tiedon virheellisessä muodossa, eli JSON:sta puuttuu jostain kohtaa heittomerkki tai muuta vastaavaa.

Kun vastaus on onnistunut, Swaggerista voidaan siirtyä takaisin pgAdmin 4:n tutkimaan työn jälkeä. Vasemmalla olevassa listassa Patients-taulukkoa oikealla klikkaamalla tulee näkyviin lista, jonka alapäässä on ”View/Edit data”-valinta. Tästä voi valita tietyn määrän rivejä tutkittavaksi, tässä tapauksessa voidaan ottaa kaikki, koska rivejä on vain yksi tällä hetkellä. PgAdminiin aukeaa uusi välilehti, jonka alhaalla näkyy juuri luomamme henkilö ja hänen ajanvarauksensa kuvan 7 mukaisesti.

Kuva 7 Potilastiedot pgAdminissa.



	Id [PK] bigint	Name text	Barcode text	IsCheckedIn boolean	Room text	Time timestamp with time zone	Description text
1	1	Olli Mappilas	string	false	Huone 252	2023-02-27 11:56:12.672+02	Käänny vase...

Tästä voidaan päätellä, että Visual Studiossa luomamme koodi saa yhteyden PostgreSQL-tietokantaamme hyvin. Kuvassa 7 näkyvää riviä voi joko muokata tai sen voi myös poistaa kokonaan, jonka jälkeen Swaggerissa potilaita hakiessa voi havaita, että kyseisen potilaan tiedot ovat myös poistuneet sieltä, joka tarkoittaa, että yhteys toimii molempiin suuntiin.

3.6 Käynnistysasetusten muuttaminen hyväksymään tarvittavat yhteydet

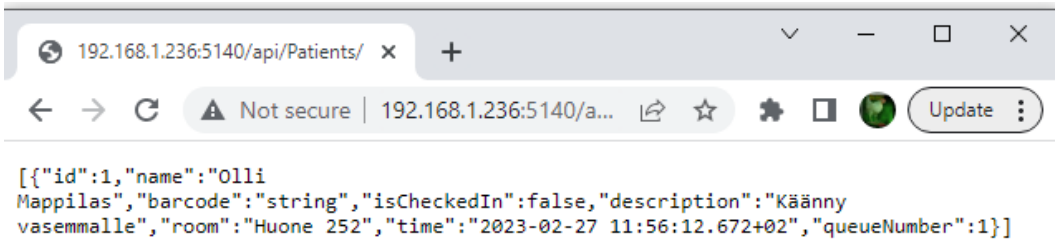
Viimeisenä tarvittavana muutoksena aukaistaan launchSettings.json-tiedosto, joka löytyy oikealta Solution Explorerista ”Properties”-kansiosta. Tämän tiedoston pohjalta löytyy ”https”-osio, jonka sisällä on ”applicationUrl”. Tämän perään lisäämällä ”http://(IP-osoite):5140;” tietokantaan saadaan yhteys suojaamattomalta nettisivulta laitteella, joka on yhteydessä samaan nettiin kuin tietokantakin. LaunchSettings.json pitäisi loppujen lopuksi näyttää kuva 8:n mukaiselta, ainoana erona IP-osoite muuttuu nettiyhteyden mukaan.

Kuva 8 LaunchSettings.json-tiedoston muuttaminen suojaamattomien yhteyksien hyväksymiseksi.

```
"https": {
    "commandName": "Project",
    "dotnetRunMessages": true,
    "launchBrowser": true,
    "launchUrl": "swagger",
    "applicationUrl": "https://localhost:7209;http://192.168.1.236:5140;",
    "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
    }
```

Muutoksen jälkeen nettiselaimessa voi avata sivun, jonka URL:na toimii juuri lisätty omasta IP-osoitteesta koostuva osoite. Tämän perään voi kirjoittaa ”/api/Patients/”, jolloin kaikki tietokantataulukkoon lisätyt rivit tulevat JSON-muodossa näkyviin, niin kuin kuvassa 9 on esitetty.

Kuva 9 Potilastiedot JSON-muodossa.



4 React-käyttöliittymä

Opinnäytetyön käyttöliittymän luomisen ohjelmointiympäristönä toimii Microsoftin VS Code. Tällä kertaa koodin kirjoituskielenä käytetään pääasiallisesti käyttöliittymien kehittämiseen tarkoitettua JavaScript-kirjastoa; Reactia.

Toimeksiantajalla on olemassa oma käyttöliittymä, joka sisältää jo valmiiksi tuen muutamalle eri puhekielelle sekä viivakoodin lukuun soveltuvat funktiot ja potilastietojen haun omista tietokannoistaan. Kuten aikaisemmassa kappaleessa mainittiin, tässä opinnäytetyössä oikeisiin tietokantoihin pääsy ei ole mahdollista, joten VS Codella luodaan uusi hyvin yksinkertainen käyttöliittymä, jonka tarkoituksena on hakea edellisessä kappaleessa luodusta pilvestä potilaiden tiedot ja lähettää nämä tiedot eteenpäin Android-sovellukselle nappia painaessa. Tässä tapauksessa käyttöliittymässä oleva nappi simuloi viivakoodinlukijan toimintaa.

4.1 Uusi React-projekti

Tässä opinnäytetyössä React-sovelluksen varmaan toimimiseen tarvitaan Node.js:n 14. tai uudempi versio. Node.js on JavaScript-ajoympäristö, jonka avulla JavaScript koodia pystyy suorittamaan palvelimen puolella, sen sijaan, että kyseinen koodi toimisi vain web-selaimessa. (NodeJs, n.d.)

Uuden projektin luominen aloitetaan VS Coden terminaalista (avautuu myös oletuksena Ctrl + Ö näppäinyhdistelmällä), vaihtoehtoisesti voi myös käyttää esimerkiksi Windowsin omaa komentokehotetta. Ensin kannattaa varmistaa, että terminaalin käytössä oleva kansio on oikein. Kansiota pystyt vaihtamaan cd (Change Directory)-komennolla kirjoittamalla halutun kansiopolun cd-komennon perään. Tämän jälkeen terminaaliin syötetään kolme eri komentoa, jotka näkyvät komennossa 4. (W3Schools, n.d. a; Microsoft, 2023g)

Komento 4 Terminaalikomennot uuden React-projektin luomiseen. (W3Schools, n.d. a)

```
npx create-react-app attune-ilmo
cd attune-ilmo
npm start
```

Komento 4:n ensimmäinen rivi käyttää Node.js-pakettien suorittajaa, eli npx (Node Package Execute) luomaan uuden React sovelluksen, joka nimetään attune-ilmoksi. Seuraavalla rivillä cd-komennolla siirrytään juuri luotuun attune-ilmo projektikansioon. Viimeisen rivin komento käynnistää kehityspalvelimen React-sovellukselle oletusselaimessa. Tämä sovellus päivittyy automaattisesti selaimessa aina kun koodissa tapahtuneet muutokset tallennetaan. Kehityspalvelimen porttina toimii oletuksena 3000, joka on myös määritetty ohjelmakoodissa 3, jolloin nettiselaimeen aukeaa tämän komennon jälkeen ”http://localhost:3000”-sivu. Localhost-osoitteen voi vaihtaa IP-osoitteeksi, jonka löytää Windowsin komentokehotteesta ”ipconfig”-komennolla. (W3Schools, n.d. a)

VS Codessa pääsee nyt tutkimaan juuri luodun attune-ilmo-projektin tiedostoja ja kansioita. Src-nimisen kansion sisältä löytyy App.js-tiedosto, joka määrittää, mitä nettiselaimessa näkyy projektin ollessa käynnissä. Tässä tapauksessa tästä tiedostosta voi poistaa kaiken jättäen jäljelle ohjelmakoodin 7 näyttämän pätkän. (W3Schools, n.d. a)

Ohjelmakoodi 7 Automaattisesti luotu App.js-tiedosto kaiken ylimääräisen poistamisen jälkeen.

```
import React from 'react';
import './App.css';

function App() {
      return (

      );
};

export default App;
```

Ohjelmakoodissa 7 tuodaan ensin Reactin pääkirjasto nykyiseen tiedostoon, eli App.js-tiedostoon. Tämä on erittäin ratkaisevassa roolissa React-käyttöliittymää luodessa, sillä sen avulla kehittäjät voivat käyttää Reactin ydintoimintoja ja luoda komponentteja käyttöliittymälle tehokkaasti. Tämän ansiosta kehittäjät voivat myös käyttää JSX-laajennusta JavaScript syntaksille, jonka avulla JavaScript-tiedostoihin pystyy kirjoittamaan HTML-tyyppistä koodia käyttöliittymän muokkaamisen helpottamiseksi. (W3Schoold, n.d. b)

Seuraavaksi tuodaan automaattisesti luodun App.css-tiedoston sisältö React-komponenttiin. App.css-tiedosto sisältää tiedostopäätteensä mukaisesti CSS-koodia (Cascading Style Sheets), jolla

voi muokata HTML- tai XML-kielellä kirjoitetun dokumentin ulkoasua ilman, että itse HTML- tai XML-koodia tarvitsee päivittää. (MDN Web Docs, 2023a)

Lopulta itse koodiin jätetään vain yksi toiminnollinen komponentti, jonka nimi on ”App”. Se palauttaa tällä hetkellä tyhjän JSX-elementin, jonka takia nettiselaimelle auennut sivu on nyt täysin tyhjä Reactin nopeasti toimivan virtuaalisen DOMin ansiosta. Lopuksi App-komponentti määritetään käytettäväksi muissa tiedostoissa tarpeen mukaan. (MDN Web Docs, 2023b)

Reactissa komponentit ovat kuin rakennuspalikoita käyttöliittymälle. Komponentti on itsenäinen ja uudelleen käytettävä pala koodia, joka määrittää osan käyttöliittymän ulkoasusta sekä käyttäytymisestä. Komponentit voidaan renderöidä HTML-elementteinä, jolloin niitä voidaan käyttää luomaan käyttöliittymä, joka koostuu monesta pienemmästä käyttöliittymäelementistä. (React, 2023)

4.2 Nappi viivakoodinlukijan simuloimiseksi

Jotta tuleva Android-sovellus tietää, kenen mukaan ajanvaraukseen liittyvät tiedot lähetetään tulostimelle tulostettavaksi, tarvitaan joku, joka kykenee suodattamaan potilaat heidän henkilökortissansa olevan viivakoodin mukaan, sillä se on uniikki jokaista Suomen kansalaista kohti. Tätä viivakoodinlukijaa tullaan simuloimaan erittäin yksinkertaisella napilla, joka sijaitsee käyttöliittymän yläreunassa keskellä.

JSX-laajennuksen ansiosta tähän koodiin voi kirjoittaa sujuvasti HTML-koodia, jolloin käyttöliittymän muokkaaminen haluamansa näköiseksi on helppoa. Ohjelmakoodi 8 lisätään ohjelmakoodin 7 ”return (”-rivin ja seuraavan sulkeutuvan sulkumerkin väliin.

Ohjelmakoodi 8 HTML-koodi sivun otsikon ja napin luomiseksi.

```
<div className = “Page”>
      <h2>Lue henkilökortti</h2>
      <button>Lue viivakoodi</button>
</div>
```

HTML:ssä -tunniste määrittää osion nettisivulle. Osioiden sisällön ulkonäköä pystyy muuttamaan helposti kerralla muokkaamalla sille määritettyä CSS-koodia. Ohjelmointikoodissa 8 luodun osion nimeksi annetaan ”Page”, koska se tulee olemaan käyttöliittymän ainoan sivun ainoa osio. Tähän nimeen viittaamalla App.css-tiedostossa pystyy muuttamaan kaikkia osion sisällä olevia elementtejä kerralla tarvittaessa. Tässä opinnäytetyössä se ei ole oleellista, joten nettisivu jää kauniin valkoiseksi ja yksinkertaiseksi. (W3Schools, n.d. c)

Seuraavalla rivillä <h2>-tunnisteiden sisällä on otsikko, joka tulee sivun yläreunaan isolla fontilla. H2 tarkoittaa header 2, joka on HTML:ssä toiseksi suurin vakio otsikkofontti. Otsikon perässä on </h2>, joka tarkoittaa, että otsikko loppuu tähän ja fontti palaa takaisin oletukseksi. (W3Schools, n.d. d)

Seuraavaksi tulee <button>-tunnisteet, jolloin sivulle tulee näkyviin nappi, jolla ei vielä tällä hetkellä ole toimintoa. Napin sisälle tulee teksti ”Lue viivakoodi”, jolloin käyttäjä tietää, mitä toiminnallisuutta voi odottaa nappia painaessa. Lopuksi Page-osio suljetaan -tunnisteella. (W3Schools, n.d. e)

Jotta nappia painaessa tapahtuisi jotain, sen tunnisteen sisään lisätään onClick-tapahtuma, niin kuin Ohjelmakoodissa 9 näkyy.

Ohjelmakoodi 9 Toiminto napille seulomaan tietokanta viivakoodin mukaan. (W3Schools, n.d. f)

```
<button onClick={() => barcodeFilter(“string”)}>Lue viivakoodi</button>
```

Lisäämällä ohjelmakoodissa 9 esitetyt muutokset koodiin, nappia painaessa suoritetaan lambdailmaisun kautta barcodeFilter-niminen muuttuja, jolle tässä tilanteessa syötetään kuva 6:ssa määritetty potilaan viivakoodi, eli ”string”. (W3Schools, n.d. f) Tällä hetkellä tätä funktiota ei ole vielä olemassa, joten seuraavaksi on aika luoda se.

4.3 Potilastietojen haku tietokantataulusta viivakoodin mukaan

Kun potilaiden tiedot ja ajanvaraukset halutaan tallentaa käytettäväksi, ne tallennetaan muuttujaan. Tätä varten Reactissa on olemassa useState-niminen ”hook”, joka palauttaa arvot

kahdella elementillä: tämänhetkinen arvo sekä funktio, joka muuttaa tämänhetkistä arvoa. (React, 2023b) Tallennettavat tiedot saadaan fetch-nimisellä metodilla, jolle syötetään haluttu osoite, joka on tässä tapauksessa ohjelmakoodissa 6 luotu päätepisteen reitti. Ohjelmakoodissa 10 näkyy, miltä tietojen haku ja tallennus näyttää kokonaisuudessaan. Ohjelmakoodi 10 tulee ennen ohjelmakoodin 7 ”return(”-riviä.

Ohjelmakoodi 10 Potilastietojen suodatus viivakoodin mukaan tietokannasta.

```
const [patients, setPatients] = useState([]);

const barcodeFilter = (barcode) => {
      fetch(`http://192.168.1.236:5140/api/Patients/${barcode}`)
                  .then(response => response.json())
                  .then((patients) = > {setPatients(patients);
                  })
      .catch((err) => {console.log(err.message);
      });
}
```

Ohjelmakoodin 10 ensimmäinen rivi luo muuttujan, johon potilaiden tiedot tullaan tallentamaan useState-hookilla. Muuttuja koostuu kahdesta elementistä, joista ensimmäinen on tämänhetkinen arvo ja seuraavana on arvoa muuttava funktio. Näiden perässä annetaan alustava arvo useState():n sulkumerkkien sisään. Koska potilastiedoissa on mahdollista olla monta riviä tietoa samaan aikaan, eli useita ajanvarauksia, mutta alustavasti muuttujaan ei tarvitse tallentaa mitään tietoja, asetetaan muuttujan arvoksi tyhjä taulukko hakasuluilla. Tällöin koodi tietää, että patients-muuttuja on taulukko, ja että sinne voidaan tallentaa enemmän kuin yksi arvo. (React, 2023b)

VS Coden pitäisi lisätä UseState-hookin automaattisesti koodin alkuun tässä vaiheessa, mutta jos näin ei käy, se tuodaan koodin ensimmäisellä riville Reactin jälkeen aaltosulkujen sisään. Koodin ensimmäinen rivi näyttää nyt tältä: ”import React, { useState } from ‘react’;”

Seuraavalla rivillä luodaan funktio, jota kutsutaan ohjelmakoodissa 9 nappia painaessa, eli barcodeFilter. Tämä funktio ottaa vastaan barcode-argumentin, joka ohjelmakoodissa 9 on kovakoodattu ”string”-sanaksi esitys syistä, koska se on ainoa viivakoodi tietokannassa tällä hetkellä. Funktion sisällä lähetetään HTTP GET-pyyntö fetch-metodilla kuvassa 8 määritettyyn osoitteeseen, jonka reitin päätepisteenä toimii ohjelmakoodissa 6 määritetty osoite. ${barcode}-termillä reitin viimeinen osa korvataan kovakoodatulla ”string”-sanalla, jolloin fetch()-metodi

hakee tietokannasta kaikki rivit, joiden barcode-muuttujan arvona on ”string”. (CodingTheSmartWay, 2022)

Tämän jälkeen koodi muuttaa haetut rivit JSON-muotoon ja tallentaa ne patients-muuttujaan, jolloin patients-muuttujassa olevat taulukon rivit pystytään pilkkomaan tarvittaessa pienempiin osiin. Lopulta mahdollisten virheiden ilmetessä, kuten yhteyden katketessa, konsoliin tulostetaan virheilmoitus. (CodingTheSmartWay, 2022)

Halutessaan barcodeFilter-funktion sisään voi lisätä esimerkiksi ”console.log(patients[0].name)”, jolloin nettisivulla nappia painaessa selaimen konsoliin tulostuu ”Olli Mappilas”, eli tietokantataulussa ”string”-barcode-muuttujan omaavaan potilaan nimi. Patients.name koostuu taulukon sisältävästä patients-muuttujasta, jonka sisälle tallentuu API:sta haetut potilastiedot sekä tässä potilastiedoista koostuvasta taulukossa olevasta ”name”-parametristä, joka määritettiin tallennettavaksi tietokantaan jo ohjelmakoodissa 1. Hakasulkujen sisällä oleva numero tarkoittaa patients-taulukossa olevien rivien järjestysnumeroa nollasta alkaen, eli 0 on taulukon ensimmäinen rivi. (MDN Web Docs, 2023c)

5 Android-sovellus

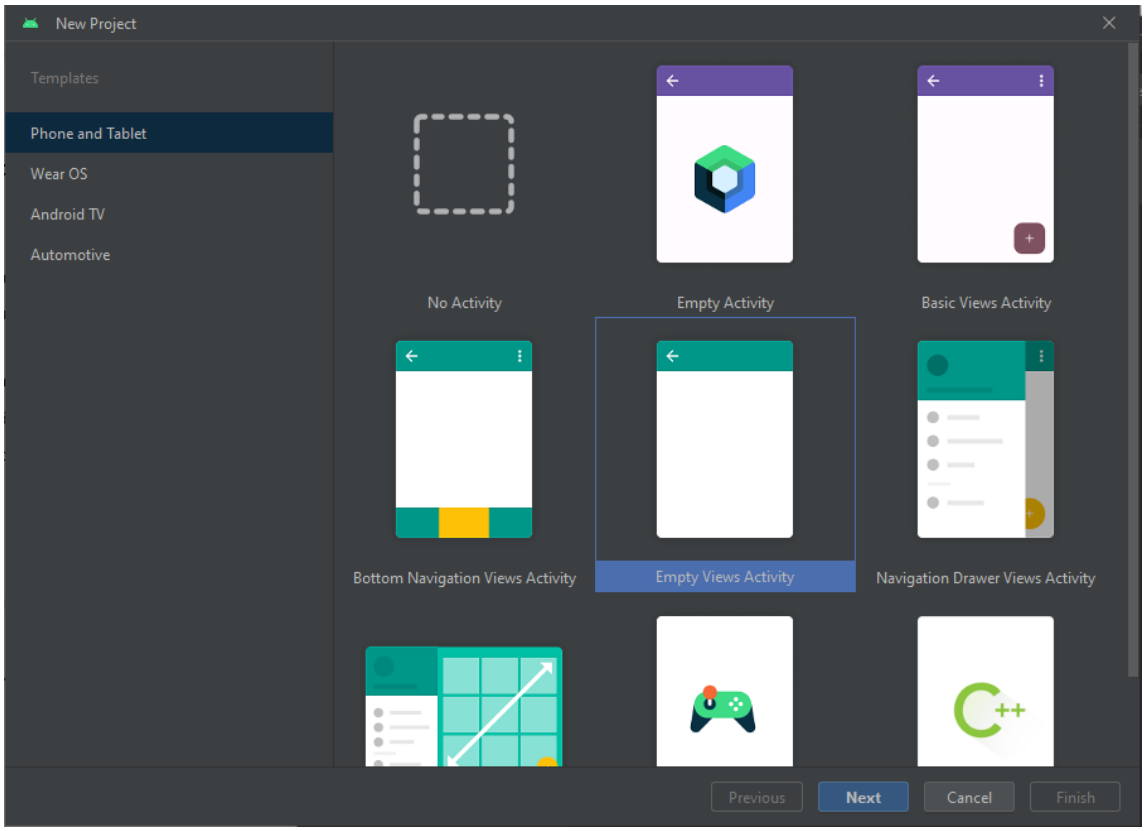
Android-sovellus on koko opinnäytetyön pääaihe, sillä projektin toimeksiantajalta löytyy jo React-käyttöliittymä omasta takaa sekä potilastiedot haetaan ennalta määritetystä paikasta, mutta potilaiden ajanvarauksen tulostamiseen soveltuvaa Android-sovellusta ei ole vielä kehitetty.

Android-osio kehitetään Googlen julkaisemalla Android-laitteille tarkoitetulla virallisella kehitysympäristöllä; Android Studiolla. Ohjelmointikielenä toimii Kotlin, joka on vuodesta 2019 ollut Googlen suosittelema kieli Android-sovelluksia kehittäessä.

5.1 Uusi Android Studio projekti

Android Studio antaa paljon vaihtoehtoja sovelluksen pohjaksi uutta projektia luodessa. Tämä sovellus tulee käyttöön suurelle Android-tabletille, joten mallipohjista valitaan puhelimet ja tabletit. Valmiista aktiviteeteistä valitaan ”Empty Views Activity”, niin kuin kuvassa 10 on valittu.

Kuva 10 Malliaktiviteetin valinta uutta Android-projektia luodessa.



Aktiviteetit ovat erittäin tärkeä komponentti Android-sovelluksissa. Sen tehtävänä on näyttää sovelluksessa tapahtuvat asiat laitteen ruudulla. Todellisuudessa aktiviteetti on Android-sovelluksen käyttöliittymä, joten tässä opinnäytetyössä käyttöliittymän sisällä näytetään toinen käyttöliittymä, joka luotiin Reactissa. (Android Developers, 2022a)

Aktiviteetin tarkoitus on vastaanottaa käyttäjän syöttöjä, kuten painalluksia ruudulla, näyttää informaatiota tai suorittaa joitakin toimintoja. Aktiviteetit voivat myös käynnistää muita aktiviteettejä, joko samassa tai eri sovelluksessa. Tämä tapahtuu yleensä käyttämällä ohjelmistomekanismia nimeltä Intent. Sen avulla voidaan myös siirtää dataa monen aktiviteetin välillä. (Android Developers, 2022a)

Kun oikea aktiviteettipohja on valittu, seuraavalla sivulla kysytään projektin nimeä, paketin nimeä, tiedostosijaintia, ohjelmointikieltä sekä minimi SDK-versiota, eli millä Android-versiolla sovellusta tullaan käyttämään. Opinnäytetyössä nimeämismalli jatkuu samalla linjalla, eli nimeksi tulee ”Attune Ilmo” toimeksiantajan kautta. Ohjelmointikieleksi valitaan Kotlin ja Android-versioksi valitaan API-taso 30, eli Android 11. Android 11 on opinnäytetyötä tehdessä yksi viimeisimmistä Android-versioista, mikä tarkoittaa sitä, että valtaosa Android-laitteista ei tule todennäköisesti tukemaan tätä sovellusta, mutta Star Micronicsin TSP654II-kuittitulostimen ohjaaminen ei onnistu vanhemmalla versiolla, kun käyttää Star Micronicsin omaa Java/Kotlin SDK:ta.

5.2 React-käyttöliittymän näyttäminen Android-sovelluksessa

Uuden projektin luotua Android Studio kokoaa tarvittavat tiedostot, mitä Android-sovellus tarvitsee toimiakseen. Tämän jälkeen Android Studioon aukeaa kaksi välilehteä, activity_main.xml sekä MainActivity.kt, koska valitsimme aktiviteetin pohjaksi ”Empty Views Activity” kuvassa 10. Tämä helpottaa suuresti käyttöliittymän muokkaamista.

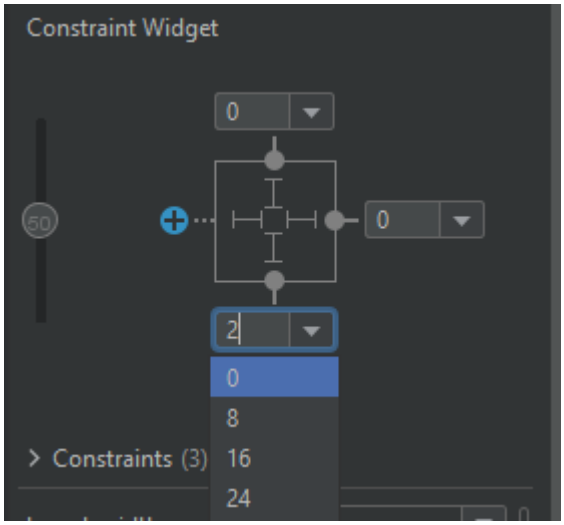
5.2.1 Käyttöliittymän asettaminen Android-laitteen ruudulle

Avaamalla activity_main.xml välilehden näkyviin tulee kaksi ruutua, jotka simuloivat puhelimen/tabletin ruutua. Oletuksena koko ruudun peittää valkoinen tausta, jossa lukee ”Hello

World!”. Tämä voidaan poistaa ja tilalle laitetaan vasemmalta paletista Widgets-kategoriasta WebView-komponentti vetämällä se jommankumman ruudun päälle, jolloin ruutu muuttuu harmaaksi ja sen keskellä lukee WebView.

WebView-komponentille annetaan rajoitteet asettelussa oikealta attribuuttivalikosta. Tämä tarkoittaa, että WebViewin reunat on aina tietyllä etäisyydellä emokomponentin reunoista. Koska WebView on ainoa komponentti käytössä, emokomponentiksi lasketaan Android-laitteen ruudun reunat. Kuvassa 11 näkyy, kun komponentin asettelulle määritetään rajoitteet.

Kuva 11 WebView-komponentin asettelun rajoitteet.



Koska WebView-komponentin on tarkoitus täyttää koko ruutu, rajoitteiksi asetetaan 0 joka suunnassa. Kuvan 11 mukaisesti sinistä plusmerkin sisältävää ympyrää painaen tulee näkyviin pudotusvalikko, josta voi vaihtaa rajoitteen määrää. Tämä tehdään yksitellen ylös, alas, vasemmalle ja oikealle, jotta komponentti täyttää Android-laitteen ruudun täysin.

Attribuuttivalikon ylimpänä kenttänä on id. Siitä komponentille voi antaa haluamansa tunnisteen, jota voidaan käyttää myöhemmin WebView-komponentin tunnistamiseen muussa koodissa. Tässä tapauksessa id-kenttään kirjoitetaan yksinkertaisesti ”webview”, koska se tulee olemaan ainoa komponentti koko projektissa.

5.2.2 React-käyttöliittymän lataaminen WebViewissä

Kun WebView-komponentti on saatu aseteltua paikoilleen laitteen ruudulle, siirrytään toiselle välilehdelle, joka aukeaa uuden projektin kanssa, eli MainActivity.kt. Projektin kaikki toiminnallinen koodi tulee tähän Kotlin-tiedostoon.

MainActivity.kt-tiedostossa on alustavasti koodia, jonka perään lisätään omat koodit. Ensimmäisenä tuodaan activity_main XML-tiedostossa luotu WebView koodin käyttöön ohjelmakoodissa 11.

Ohjelmakoodi 11 WebViewin asettaminen objektiksi Kotlin-koodissa. (Android Developers, 2023b)

```
val WebView: WebView = findViewById(R.id.webview)
```

Val-avainsanaa käytetään määrittämään muuttuja, jonka arvoa ei voi enää muokata myöhemmin koodissa. Tässä tapauksessa tähän muuttujaan tulee luvussa 5.2.1 lisätty WebView, joka löydetään ”findViewById()”-metodilla. Sulkujen sisällä oleva R-luokka ilmoittaa, että kyseessä on sovelluksen resurssi, kuten asetelma, string, tai sitten ID, jota käytettiin WebView-komponentille tunnisteen antamiseksi aikaisemmin. (Android Developers, 2023b)

Seuraavana Android-sovellus vaatii erikseen asetuksen, jolla Reactin JavaScript saadaan toimimaan. Se onnistuu helposti yhdellä rivillä koodia. Samalla voidaan asettaa WebViewClient, joka on vastuussa sivun lataamisesta ja virheilmoituksista. Myös WebView-komponenttiin voidaan lisätä yhteys JavaScript-käyttöliittymään, jonka ansiosta myöhemmin Android Studiossa luotu koodi pystyy olemaan yhteydessä VS Codessa luodun koodin kanssa. Nämä kolme riviä näkyvät ohjelmakoodissa 12.

Ohjelmakoodi 12 JavaScript-tuki Android-sovellukseen. (Android Developers, 2022b)

```
webView.settings.javaScriptEnabled = true
webView.webViewClient = WebViewClient()
webView.addJavascriptInterface(WebAppInterface(this), "Android")
```

Ohjelmakoodin 12 ensimmäisellä rivillä yksinkertaisesti annetaan WebView-komponentille lupa käyttää JavaScriptejä. Oletuksena JavaScriptit ovat poistettu käytöstä turvallisuussyistä.

Seuraavalla rivillä oletusselaimen aukeaminen yli kirjoitetaan niin, että kaikki mahdolliset toiminnot tapahtuvat tässä sovelluksessa. Ohjelmakoodin 12 viimeinen rivi mahdollistaa, että Android Studiossa tehtyjä metodeja pystytään kutsumaan JavaScriptillä käyttämällä Android-nimeä. Tätä tullaan käyttämään myöhemmin, kun VS Codessa muokataan JavaScriptiä hieman eteenpäin. (Android Developers, 2022b)

Viimeisenä, jotta WebView-komponenttiin saadaan näkyviin oikea nettisivu, lisätään yksi rivi koodia, joka näkyy ohjelmakoodissa 13.

Ohjelmakoodi 13 React-käyttöliittymän näyttäminen Android-sovelluksessa. (Android Developers, 2022b)

```
webView.loadUrl(http://IP-osoite:3000/)
```

Ohjelmakoodissa 13 WebView-komponenttiin ladataan näkyviin VS Codella luotu käyttöliittymä ottamalla yhteys omaan IP-osoitteeseen ja käyttämällä porttia 3000, jota React-kehityspalvelimet käyttävät oletuksena. Tämä osoite voidaan korvata millä tahansa osoitteella tarvittaessa. (Android Developers, 2022b)

5.2.3 Tarvittavat luvat Android-sovellukselle nettiin yhdistämiseen

Kuitenkaan tässä vaiheessa testatessa WebViewissä ei näy mitään, sillä sivuun ei saada yhteyttä. Se johtuu siitä, että sovelluksesta puuttuu tarvittavat oikeudet internettiin yhdistämiseen. Lupa nettiin yhdistämiseen on helppo antaa lisäämällä tarvittava rivi AndroidManifest.xml-tiedostoon, joka löytyy Android Studiosta vasemmalta projektilistasta ”manifests”-kansiosta. Tarvittava rivi näkyy ohjelmakoodissa 14, joka voidaan lisätä XML-tiedostoon ennen <application>-riviä.

Ohjelmakoodi 14 Oikeus Android-sovellukselle yhdistää verkkoon. (Android Developers, 2023c)

```
<user-permission android:name=”android.permission.INTERNET”/>
```

Lisäksi Android-sovellus ei Android-versio 9 jälkeen suostu oletuksena yhdistämään HTTP-osoitteisiin turvallisuussyistä. Koska aikaisemmin luotu tietokanta löytyy HTTP-palvelun takaa, Android-sovellukselle pitää antaa erillinen lupa, jotta tähän tietokantaan voidaan yhdistää. Tämä

käy myös helposti lisäämällä yhden rivin koodia samaan AndroidManifest XML-tiedostoon. Tällä kertaa koodirivi tulee <application>-komponentin sisään, kuten ohjelmakoodissa 15 näkyy. (GeeksforGeeks, 2022)

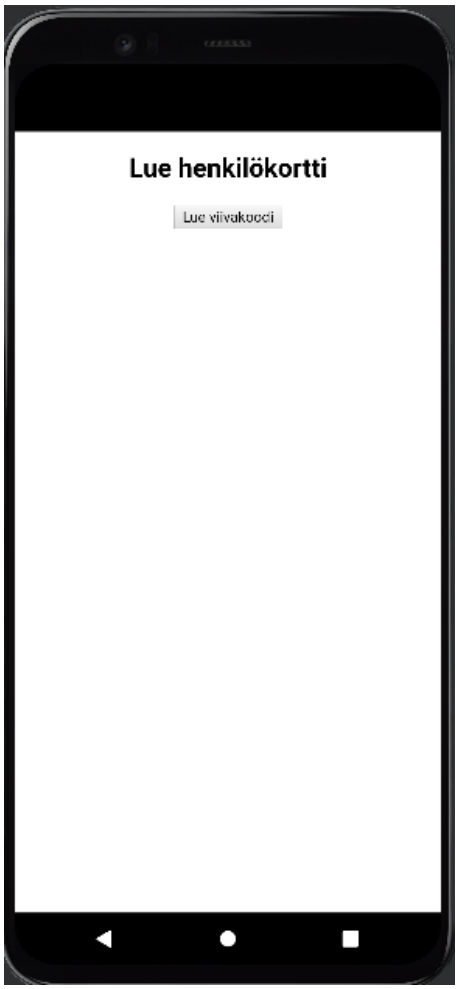
Ohjelmakoodi 15 XML-koodi, jolla Android-sovellus voi yhdistää HTTP-liikenteeseen. (GeeksforGeeks, 2022)

```
<application
      android:allowBackup=”true”
      ...
      android:usesCleartextTraffic=”true”
      ...
/application>
```

Ohjelmakoodissa 15 lihavoitu teksti lisätään XML-tiedostoon, jolloin Android-sovellus saa yhteyden HTTP-osoitteisiin samalla tavalla, kuin se saisi yhteyden turvallisempiin HTTPS-osoitteisiin. Kun toimeksiantaja ottaa sovelluksen käyttöönsä, tämä rivi poistetaan koodista turvallisuuden varmistamiseksi.

Tämän jälkeen Android Studiossa käynnistetään emulaattori, jolla sovellusta voi niin sanotusti esikatsella. Emulaattorilla suoritettavan laitteen teknisillä tiedoilla ei ole väliä, kunhan sen API-taso on tarpeeksi korkea. Sairaaloissa sovellusta käytetään tableteilla, joten yksi vaihtoehto on emuloida laitetta, jossa on suuri näyttö, mutta selkeyden takia kuvassa 12 emuloitavaksi laitteeksi valitaan Googlen Pixel 4-puhelin.

Kuva 12 React-käyttöliittymä Android-emulaattorilla.



Kuvasta 12 näkee, että Reactilla luotu käyttöliittymä tulee halutulla tavalla näkyviin Android-laitteelle. ”Lue viivakoodi”-nappia painaessa ei tapahdu vielä mitään, joten seuraavaksi sille annetaan toiminto, jolla JavaScriptissä pilvestä haetut tiedot saadaan tuotua Android-sovellukseen ja vietyä siitä eteenpäin tulostimelle.

5.3 Tulostimeen yhdistäminen Android-sovelluksella

Star Micronicsin kuittitulostimeen yhdistämistä varten pitää jälleen antaa uusia lupia Android Studiossa. Ensimmäinen lisäys tapahtuu Android Studion vasemmasta reunasta olevasta projektipuusta ”app”-kansion juuresta. Siellä on build.gradle-niminen tiedosto, jonka sisältä löytyy

muun muassa SDK-versiot ja sovelluksen riippuvuudet. Näiden riippuvuuksien loppuun lisätään ohjelmakoodi 16.

Ohjelmakoodi 16 Star Micronics SDK:n riippuvuuksien lisääminen Android-sovelluksen build.gradle-tiedostoon. (StarPRNT SDK Users Manual, n.d. a)

```
dependencies {
      ...
      implementation 'com.starmicronics:stario:2.10.1'
      implementation 'com.starmicronics:starioextenson:1.15.1'
}
```

Ohjelmakoodissa 16 lisätyillä riippuvuuksilla Android Studio löytää Star Micronicsin ulkoiset kirjastot ja tuo ne koodiin automaattisesti. (Android Developers, 2023e)

Build.gradle-tiedostoon lisättyjen riippuvuuksien jälkeen muokataan jälleen AndroidManifest.xml-tiedostoa. AndroidManifest.xml:n pohjalla on <intent-filter>-komponentti, jonka sisällä on jo pari riviä tekstiä. Näiden rivien perään ja </intent-filter>-komponentin jälkeen lisätään ohjelmakoodi 17.

Ohjelmakoodi 17 Android-sovellukselle lupa käyttää USB-yhteyttä. (StarPRNT SDK Users Manual, n.d. a)

```
<intent-filter>
      ...
      <action
android:name="android.hardware.usb.action.USB_DEVICE_ATTACHED" />
    <action
android:name="android.hardware.usb.action.USB_ACCESSORY_ATTACHED" />
</intent-filter>

<meta-data android:name="android.hardware.usb.action.USB_DEVICE_ATTACHED"
android:resource="@xml/device_filter" />
<meta-data
android:name="android.hardware.usb.action.USB_ACCESSORY_ATTACHED"
android:resource="@xml/accessory_filter" />
```

Kuittitulostin tullaan yhdistämään Android-laitteeseen USB-johdolla, joten sovellukselle pitää antaa lupa yhdistää USB-laitteisiin. Näiden lihavoitujen rivien lisäksi pitää myös tarkentaa tiedosto, joka määrittää kyseisten USB-laitteiden ominaisuuksia. Kun yhdistetty laite täsmää suodattimessa olevan laitteen kanssa, Android-laite kysyy lupaa yhdistää laitteeseen. Hyväksyessä tämän sovelluksella on lupa käyttää tulostinta, kunnes USB-liitäntä katkeaa. (Android Developers, 2021)

Lopulta tulostimien ominaisuudet lisätään XML-tiedostoon, josta AndroidManifest.xml suodattaa USB-yhteydessä olevan laitteen tiedot. Tätä varten luodaan device_filter.xml- ja accessory_filter.xml-tiedostot, Android Studion app/src/main/res/xml-tiedostopolkuun. Device_filter.xml tiedostoon voi lisätä niin monta laitetta, kuin haluaa, mutta tässä projektissa pärjätään yhdellä, joka näkyy ohjelmakoodissa 18.

Ohjelmakoodi 18 Tarvittavan tulostimen ominaisuuksien lisääminen device_filter.xml-tiedostoon. (StarPRNT SDK Users Manual, n.d. a)

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <usb-device class="255" subclass="66" protocol="1" />
    <usb-device vendor-id="1305" product-id="0003" />
</resources>
```

Toinen luotava tiedosto, eli accessory_filter.xml tulee näyttämään ohjelmakoodin 19 mukaiselta.

Ohjelmakoodi 19 Accessory_filter.xml-tiedoston sisältö. (StarPRNT SDK Users Manual, n.d. a)

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <usb-accessory model="Star TSP654II" manufacturer="STAR"/>
    <usb-accessory model="mC-Label3" manufacturer="Star Micronics"/>
</resources>
```

Ohjelmakoodissa 18 ja 19 luodaan uudet XML-tiedostot nimeltään device_filter ja accessory_filter, jossa ilmoitetaan Android-laitteella, että Star Micronicsin kuittitulostin on luotettava laite. Näillä lisäyksillä Android-laitteella saadaan yhteys Star Micronics kuittitulostimeen USB:n kautta. Tämän jälkeen on aika luoda itse sovelluksen yhdistäminen tulostimeen. (StarPRNT SDK Users Manual, n.d. a)

Jotta tulostimeen saadaan yhteys ja tulostimesta saadaan jotain ulos, pitää luoda uusi funktio Android-sovelluksessa. Yleensä olisi järkevintä tehdä kokonaan uusi Kotlin-tiedosto tätä varten, mutta yksinkertaisuuden vuoksi funktio rakennetaan MainActivity.kt-tiedoston pohjalle. MainActivity-luokan alle aloitetaan uusi luokka nimeltään WebAppInterface, joka määritettiin jo ohjelmakoodissa 12. Tämän jälkeen lisätään huomautus, jolla koodi ymmärtää, että tässä tullaan käyttämään JavaScript-metodeja. Huomautus merkataan @-merkillä, jonka jälkeen kirjoitetaan

JavascripInterface. Sen jälkeen luodaan uusi funktio tulostimen toiminnoille, kuten ohjelmakoodissa 20 näytetään. (Android Developers, 2023d)

Ohjelmakoodi 20 Uusi luokka ja metodi Android-sovelluksessa, jota kutsutaan JavaScriptillä.

```
class WebAppInterface(private val mContext: Context) {
          @JavaScriptInterface
          fun connectPrinter(
                    time: String,
                    room: String,
                    guide: String,
                    queue: Int
          ) {
          var port: StarIOPort? = null

          try {
                    port = StarIOPort.getPort("USB:", "", 10000, mContext)

                    val builder =
StarIoExt.createCommandBuilder(StarIoExt.Emulation.StarLine)
          var status = port.beginCheckedBlock()
```

JavaScriptInterface-huomautuksen jälkeen luodaan funktio, johon otetaan muuttujina string-muodossa päivämäärä, huone, ohjeistus huoneeseen sekä numeraalina vuoronumero. Funktion alussa ilmoitetaan, että port-nimisessä muuttujassa tulee olemaan Star Micronicsin SDK:sta luokka, jolla kommunikoidaan tulostimien kanssa. Try/catch-lausunnon sisällä avataan portti, joka hakee USB-yhteydellä 10000 millisekunnin sisään tulostinta. (StarPRNT SDK Users Manual, n.d. b) Kun tulostin löytyy, luodaan ”builder”-muuttuja, jota käytetään kuitille tulostettavan tiedon kirjoittamiseen selkokielellä. (StarPRNT SDK Users Manual, n.d. c)

Lopulta aloitetaan monitorointimetodi, joka päätetään tulostamisen jälkeen port.endCheckedBlock()-metodilla. Nämä metodit tarkistavat tulostamisen tilanteen, ja että data on tulostettu kokonaan. (StarPRNT SDK Users Manual, n.d. b)

5.4 Tulosteen luominen ja tulostaminen

Seuraavaksi sovellukselle kerrotaan, mitä halutaan tulostaa. Tulostettava materiaali kirjoitetaan rivi riviltä käyttäen builder-muuttujaa. Koko tulostettava koodi löytyy opinnäytetyön lopusta (Liite 3), mutta ohjelmakoodissa 21 on pari yksinkertaista esimerkkiä builder-muuttujan keskeisistä toiminnoista.

Ohjelmakoodi 21 Yksinkertainen esimerkki tulosteen luomisesta StarPRNT SDK:lla.

```
builder.beginDocument()

builder.appendAlignment(ICommandBuilder.AlignmentPosition.Center)
builder.append("Hello World").toByteArray())
builder.appendMultiple("Hello World").toByteArray(), 2, 2)
builder.appendCutPaper(ICommandBuilder.CutPaperAction.PartialCutWithFeed)

builder.endDocument()

val command = builder.commands

port.writePort(command, 0, command.size)

status = port.endCheckedBlock()

StarIOPort.releasePort(port)
```

Aluksi ohjelmakoodissa 21 ilmoitetaan, että tulosteen kirjoittaminen aloitetaan, ja kaikki builder-muuttujan komennot lisätään niin sanottuun tulostusjonoon. Seuraavalla rivillä määritetään tulosteen sisällön tasaus keskelle tulostuspaperia. Sen jälkeen jonoon lisätään kaksi ”Hello World”-tekstiä. Ensimmäisellä rivillä teksti lisätään oletusfontilla, mutta jälkimmäisellä rivillä tämä teksti lisätään lisämuuttujien kera. Näillä muuttujilla määritetään tekstin korkeus ja paksuus, jolloin suurempi numero tarkoittaa suurempaa tekstiä 6 ollessa maksimikoko. Kun haluttu teksti on lisätty tulostusjonoon, ilmoitetaan tulostimelle, että kuittipaperi pitää leikata, jotta se on helpompi ottaa ulos tulostimesta. Lopulta builder.endDocument()-metodilla ilmoitetaan, että tulostusjonon rakentaminen loppuu tähän. (StarPRNT SDK Users Manual, n.d. d)

Tämän jälkeen koko tulostusjono tallennetaan yhteen muuttujaan, joka lähetetään tulostimelle tulostettavaksi. Sen jälkeen ohjelmakoodissa 16 aloitettu tulostamisen monitorointi lopetetaan endCheckedBlock()-metodilla ja tulostimen kanssa kommunikointiin käytetty portti suljetaan. (StarPRNT SDK Users Manual, n.d. b)

Viimeisenä try/catch-lausunto suoritetaan loppuun virheilmoituksilla ja portin sululla, jotta tulostusjonoon ei jää mitään kummittelemaan seuraavaa sairaalaan ilmoittautuvaa potilasta varten. Koodi kokonaisuudessaan löytyy opinnäytetyön lopusta (Liite 3).

5.5 React-metodin yhdistäminen Android-sovellukseen

Pilvessä olevasta tietokantataulusta tietojen lähettämiseksi pitää palata takaisin VS Coden React-sovelluksessa olevaan koodiin. Kun käyttöliittymässä olevaa nappia painaa, sovellus niin sanotusti skannaa viivakoodin ja hakee tällä viivakoodilla merkityt rivit tietokannasta. Tämän jälkeen rivit tallentuu listaan, josta ne voi pilkkoa rivien ja sarakkeiden mukaan. Android-sovellukseen lähetetään ajanvaraus, huone, ohjeet huoneeseen ja jonotusnumero, joten nämä sarakkeet pitää eritellä. Tämä kaikki voidaan tehdä samassa funktiossa, kuin ajanvarausten hakeminen, eli barcodeFilter-funktiossa. Kun tietokannasta haetut rivit on tallennettu setPatients();-rivillä sen jälkeen lisätään ohjelmakoodi 22.

Ohjelmakoodi 22 Päivämäärän formatointi luettavaan muotoon sekä muiden ajanvaraustietojen lähettäminen Android-sovellukseen.

```
const dateString = patients[0].time;
                const date = new Date(dateString);
                const options = {
                          day: 'numeric',
                          month: 'long',
                          year: 'numeric',
                          hour: 'numeric',
                          minute: 'numeric',
                          timeZone: 'UTC,
                };
                const formatter = new Intl.DateTimeFormat('fi-FI',
                options);
                const formattedDate = formatter.format(date);

                window.Android.connectPrinter(
                          formattedDate,
                          patients[0].room,
                          patients[0].description,
                          patients[0].queueNumber
                )
```

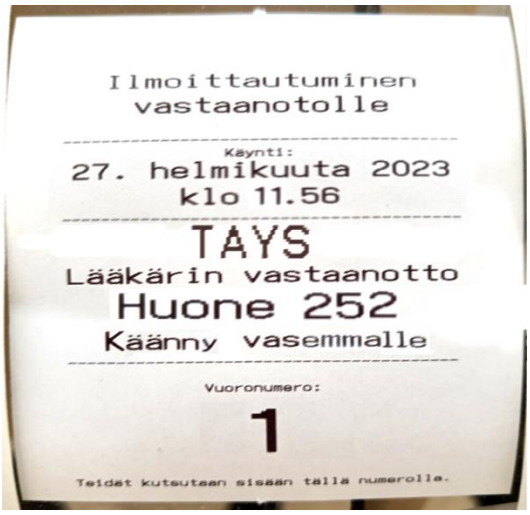
Ohjelmakoodissa 22 muutetaan aluksi tietokantataulusta saatu päivämääräsarake oikeanlaiseen muotoon. Oletuksena päivämäärä näyttäisi esimerkiksi tältä: 2023-02-27 11:56:12.672+02, joka saattaa olla sairaalassa asioivalle ihmiselle turhan sekava muoto. Tämä formaatti koostuu ensin vuodesta, jonka jälkeen tulee kuukausi, sitten päivä, tunti, minuutti, sekunti, millisekunti ja lopulta aikavyöhyke UTC/GMT aikaan nähden, eli Suomen tapauksessa kaksi tuntia edellä. Tämä merkintä tapa on kuitenkin helppo muuttaa haluttuun muotoon, eli 27. helmikuuta 2023 klo 11:56.

Aluksi tietokannasta haetusta patients-listaan tallennetuista tiedoista otetaan ensimmäinen rivi ja sen rivin time-sarake. Tämä tallennetaan uuteen muuttujaan, jonka jälkeen koodille kerrotaan, että kyseessä on päivämäärä. Options-objektissa ilmoitetaan, missä muodossa tämän päivämäärämuuttujan optiot ovat. Kaikki muut voidaan merkitä numeroiksi, paitsi kuukausi merkitään ”long”, jotta numeron sijaan kuukausi näkyy tekstinä. Tämän jälkeen luodaan uusi Intl.DateTimeFormat-objekti, jossa alueeksi ilmoitetaan Suomi sekä aiemmin määritetty Options-objekti parametreiksi. Näin koodi tietää, että tämä kyseinen päivämäärä on Suomen aikavyöhykkeellä UTC:n mukaan sekä kuukasi kirjoitetaan Suomeksi. (MDN Web Docs, 2023d)

Kun päivämäärä on saatu oikeaan muotoon, on aika lähettää tarvittavat sarakkeet Android-sovellukselle. Ohjelmakoodin 22 loppupäässä on ”window.Android.connectPrinter(”-rivi. Tällä rivillä WebViewissä näkyvät JavaScript elementit pystyvät olemaan yhteydessä muun Android-koodin kanssa. ConnectPrinter-metodilla viitataan Android Studiolla luotuun saman nimiseen metodiin, joka ottaa vastaan päivämäärän, huoneen nimen, ohjeet huoneeseen sekä jonotusnumeron. Nämä tarvittavat tiedot karsitaan tietokannasta saaduista tiedoista patients-muuttujan ensimmäiseltä riviltä. Lisäksi mukaan lisätään juuri formatoitu aikamuoto. React-käyttöliittymän koodi näkyy kokonaisuudessaan opinnäytetyön lopusta (Liite 2).

Lopulta Android Studiossa valitaan yläriviltä Build-valikko ja sen sisältä Build APK(s). Android Studio kokoaa tehdystä koodista APK-tiedostopäätteen omaavan tiedoston, joka on Android-sovellusten asentamiseen tarkoitettu pakettitiedosto. Tämän tiedoston voi asentaa samassa netissä olevalle fyysiselle Android-laitteelle, johon saa kiinni Star Micronicsin kuittitulostimen USB C-liitännällä. Kun sovellus on asennettu ja käynnistetty, Android-laitteen ruudulle tulee kuvasta 12 tuttu näkymä, mutta tällä kertaa nappia painaessa, tulostimesta tulostuu lappu, joka näkyy kuvassa 13. (Android Developers, 2023f)

Kuva 13 Android-laitteella tulostettu kuitti ajanvarauksesta.



Kuten kuvassa 13 näkyy, tulostimesta tuli ulos lappu, johon on tulostettu kaikki Android Studiossa kirjoitettu koodi, joka näkyy kokonaisuudessaan opinnäytetyön lopussa (Liite 3). Tällä lapulla asiakkaan pitäisi pystyä toimimaan täysin itsenäisesti sairaalan sisätiloissa.

6 Tulokset ja pohdinta

Opinnäytetyön tuloksena on pääasiallisesti Android-sovellus, joka hakee pilvessä sijaitsevasta tietokannasta potilaiden ajanvaraukset ja tulostaa ne paikallisella kuittitulostimella asiakkaalle ilmoittautuessa sairaalaan itseilmoittautumislaitteella. Hienoiseksi haittapuoleksi jäi se, että sovellus vaatii vähintään Androidin 11. version toimiakseen, jolloin tarvittavaa laitteistoa hankittaessa kulut voivat nousta enemmän, kuin toivoisi.

Vaikka kaksi kolmasosaa opinnäytetyöstä kului tietokannan ja käyttöliittymän luomiseen, nekin olivat hyviä oppimisen näytön kohteita palvelinpuolen ja selainpuolen luomisesta. Myöskään ilman niitä tämä opinnäytetyö ei olisi onnistunut lainkaan ja jos ne olisi jätetty pois työn kirjallisesta osuudesta, moni Android-sovellusta koskevasta kohdasta olisi jäänyt hyvin epäselväksi ilman kontekstia.

Projektin luonti oli loogisinta aloittaa aivan pohjalta, eli tietokannan rakentamisesta. Tämä tapahtui Microsoftin Visual Studiolla C#-ohjelmointikielellä, jonka käytöstä projektin alkaessa oli jo hieman kokemusta, mutta kaikki projektissa luotu oli uutta territoriota koodaamisen maailmassa. Tähän auttoi suunnattomasti Microsoftin omat ohjeet, joiden mukaan koko palvelinpuoli oikeastaan luotiin. Migraatiot olivat myös täysin uusi asia opinnäytetyön alussa, mutta sen oppiminen oli helppoa ja nopeaa.

React-käyttöliittymän luominen oli kaikkein eniten tuttua puuhaa koulussa opittujen asioiden ansiosta. Opinnäytetyötä kirjoittaessa moneen Reactilla luotuun asiaan ei ollut kovin selviä lähteitä, koska ne olivat yhdistelmiä useista koulussa opituista asioista tai suoraan päättelemällä muiden toimintojen perusteella.

Projektin aloittaessa Kotlin oli täysin uusi kieli. Toimeksiantaja sanoi, että ei ole väliä ohjelmoiko Android-sovelluksen Javalla; joka on jo entuudestaan erittäin tuttu, vai Kotlinilla; joka on Googlen suosittelema ohjelmointikieli Android-sovelluksia varten. Loppujen lopuksi projekti ohjelmoitiin Kotlinilla, joka ei tuonut mitään suuria haasteita projektin aikana muun muassa netissä olevan kattavan informaation ansiosta.

Suurimmat haasteet opinnäytetyötä tehdessä tulivat SSL-sertifikaattien ja HTTP-osoitteiden kanssa. Jokaisessa projektin kolmesta vaiheesta oli vähintään yksi suurempi este, joka johtui loppujen lopuksi vain tietokantaan yhdistämisestä turvattomalla yhteydellä, mutta kaikkiin näistä löytyi loppujen lopuksi ratkaisu, jotta opinnäytetyö saatiin loppuun. Sairaaloissa käyttöön otettaessa näitä samoja ongelmia tuskin tulee, koska sairaaloiden tietoturva on luultavasti ja toivottavasti parempaa kuin mitä tässä projektissa on esitetty.

Jatkossa tätä sovellusta voidaan kehittää toimimaan vanhemmilla Android-versioilla ja useammilla eri kuittitulostimilla, varsinkin eri valmistajien tulostimilla, kuten Zebra ja Toshiba. Lisäksi kuitista voi tehdä monipuolisemman sairaalakohtaisesti, esimerkiksi lisäämällä sairaalan logon yläreunaan ja tekstiohjeiden sijaan pistää kuitin alareunaan kartan, joka perustuu laitteen sijaintiin luetun ymmärtämisessä tapahtuvien virheiden minimoimiseen.

Android-sovellukseen lisätään myöhemmin toiminnot, joilla käyttöliittymään saadaan tarvittaessa näkyviin virheilmoitukset, kuten kuittipaperi on loppumassa tai tulostimessa on jotain muuta vikaa. Toimeksiantajan toivomus olisi myös, että jos henkilöllä on tietyn ajan sisään useampi kuin yksi ajanvaraus, laite ilmoittaisi hänet kaikille niille kerralla ja tulostaisi kaikkiin esimerkiksi kolmeen vastaanottoon laput, jolloin potilaan ei tarvitse ravata jatkuvasti laitteella ilmoittautumassa.

7 Yhteenveto

Alussa määritettyihin tutkimuskysymyksiin saatiin vastattua hyvin. Kuitin tulostaminen onnistuu simppelillä Android-sovelluksella, joka koostuu jo valmiiksi luodusta Star Micronicsin SDK:sta, joten kuitin sisällön muokkaaminen on erittäin helppoa tarvittaessa. Se helpottaa myös tulevaisuudessa terveysasemien osuutta, koska alustavan asennuksen jälkeen kuittia pystytään päivittämään etänä ja muutamassa sekunnissa päivittämään koodi halutuilla laitteilla, jolloin ne ovat taas valmiina käyttöön.

Tekijänä opin opinnäytetyöprosessin aikana paljon uutta. Lähes kaikki tässä projektissa oli uutta, React-käyttöliittymän perusteita lukuun ottamatta. Ohjelmointiin käytettävät kielet eivät kuitenkaan eroa liikaa toisistaan, jotta kielestä toiseen hyppiminen ei aiheuttanut suurta päänvaivaa syntaksien takia. Projektin myötä ymmärrys tietokannoista ja HTTP- sekä HTTPS-pyynnöistä selkeni erittäin paljon.

Jatkokehitykseen on monenlaisia ideoita eri tulostinvalmistajien tukemiseksi, vanhempien Android-laitteiden tukemiseksi sekä käyttäjäkokemuksen parantamiseksi ja helpottamiseksi.

Aihe kokonaisuutenaan oli erittäin mukava, vaikkakin toimeksiantajan näkökulmasta palvelinpuoli ja käyttöliittymä olivat niin sanotusti turhia, koska ne löytyvät heiltä jo entuudestaan. Kuitenkin opinnäytetyöksi koen, että tämä oli hyvän laajuinen näyttämään, että olen tosiaan oppinut koulussa jotain tämän kolmen vuoden aikana, ja että opittuja asioita pystyy hyödyntämään ja etenkin kehittämään työelämässä oikeiden projektien parissa.

Lähteet

Kioskindustry (2023) *Kiosks History*

https://kioskindustry.org/kiosk-about-2023/kiosk-history/

Yle (2013) *Itsepalvelu tulee sairaalaankin*

https://yle.fi/a/3-6483596

Finn-ID (n.d.) *Attune Ilmo*

https://info.finn-id.fi/attuneilmo

Redyref (2021) *What is a touch screen kiosk?*

https://redyref.com/touch-screen-kiosk/

Investopedia (2023) *What Is an ATM and How Does It Work?*

https://www.investopedia.com/terms/a/atm.asp

Lux, Cindy M. (2004) *Patient registration kiosk -patentti*

https://patentimages.storage.googleapis.com/7a/b4/da/fba0aaa14701a2/WO2004084034A2.pdf

Rhoads, Jared & Drazen, Erica (2009) *Touchscreen Check-In: Kiosks Speed Hospital Registration*

https://www.chcf.org/wp-content/uploads/2017/12/PDF-TouchscreenCheckInKiosks.pdf

Star Micronics (n.d. a) *Background*

https://star-m.jp/eng/company/history.html

Star Micronics (n.d. b) *About Us*

https://star-emea.com/about-us/

Techopedia (2014) *Thermal Printer*

https://www.techopedia.com/definition/3629/thermal-printer

Star Micronics (2023) *TSP650II Series*

https://star-emea.com/products/tsp650ii/

TechTarget (2019) *Front end and back end*

https://www.techtarget.com/whatis/definition/front-end

FreeCodeCamp (2022) *CRUD Operations – What is CRUD?*

https://www.freecodecamp.org/news/crud-operations-explained/

FreeCodeCamp (2023) *Visual Studio vs Visual Studio Code – What's The Difference Between These IDE Code Editors?*

https://www.freecodecamp.org/news/visual-studio-vs-visual-studio-code/

TutorialsTeacher (n.d.) *Learn ASP.NET Web API*

https://www.tutorialsteacher.com/webapi

Swagger (n.d.) *Swagger UI*

https://swagger.io/tools/swagger-ui/

AWS (n.d.) *What is SQL (Structured Query Language)?*

https://aws.amazon.com/what-is/sql/

The PostgreSQL Global Development Group (2023) *PostgreSQL 15.2 Documentation*

https://www.postgresql.org/files/documentation/pdf/15/postgresql-15-A4.pdf

PostgreSQL (2017) *pgAdmin*

https://www.postgresql.org/ftp/pgadmin/

PgAdmin 4 (2022) *Column Dialog*

https://www.pgadmin.org/docs/pgadmin4/6.18/column_dialog.html

PgAdmin 4 (2023a) *FAQ*

https://www.pgadmin.org/faq/

PgAdmin 4 (2023b) *Download*

https://www.pgadmin.org/download/

The pgAdmin 4 Development Team (2023) *pgAdmin 4 Documentation, Release 6.21*

https://ftp.postgresql.org/pub/pgadmin/pgadmin4/v6.21/docs/pgadmin4-6.21.pdf

NodeJS (n.d.) *About Node.js*

https://nodejs.org/en/about

Visual Studio Code (2023a) *Programming Languages*

https://code.visualstudio.com/docs/languages/overview

Visual Studio Code (2023b) *Extension Marketplace*

https://code.visualstudio.com/docs/editor/extension-marketplace

A-Team Global (2023) *Why is React so popular these days?*

https://a-team.global/blog/why-is-react-so-popular

Medium (2021) *ReactJS: A brief history*

https://medium.com/@sjarancio/reactjs-a-brief-history-3c1e969a477f

TatvaSoft (2022) *Top 10 companies that use React*

https://www.tatvasoft.com/outsourcing/2022/02/top-10-successful-companies-using-react-js.html

React (n.d.) *Virtual DOM and Internals*

https://legacy.reactjs.org/docs/faq-internals.html#what-is-the-virtual-dom

Britannica (2022) *Android*

https://www.britannica.com/technology/Android-operating-system

Android Developer Fundamentals (2018) *1.0: Introduction to Android*

https://urly.fi/38A4

New Media Aid (2015) *Android App Development*

https://www.newma.co.uk/android-app-development

Android Developers (2023a) *Meet Android Studio*

https://developer.android.com/studio/intro

Red Hat (2020) *What is React API?*

https://www.redhat.com/en/topics/api/what-is-a-rest-api

Pilvikoodarin blogi (2016) *REST palvelun pystytys*

http://pilvikoodari.net/?p=218

Microsoft (2013) *ASP.NET Scaffolding in Visual Studio 2013*

https://learn.microsoft.com/en-us/aspnet/visual-studio/overview/2013/aspnet-scaffolding-overview

Microsoft (2020) *Microsoft.EntityFrameworkCore Namespace*

https://learn.microsoft.com/en-us/dotnet/api/microsoft.entityframeworkcore?view=efcore-6.0

Microsoft (2022a) *WebApplication Class*

https://learn.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.builder.webapplication?view=aspnetcore-6.0

Microsoft (2022b) *Safe storage of app secrets in development in ASP.NET Core*

https://learn.microsoft.com/en-us/aspnet/core/security/app-secrets?view=aspnetcore-6.0&tabs=windows

Microsoft (2022c) *Configuration in ASP.NET Core, XML configuration provider*

https://urly.fi/36hd

Microsoft (2022d) *Enforce HTTPS in ASP.NET Core*

https://learn.microsoft.com/en-us/aspnet/core/security/enforcing-ssl?view=aspnetcore-6.0&tabs=visual-studio%2Clinux-ubuntu

Microsoft (2022e) *Overview of ASP.NET Core MVC*

https://learn.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-5.0

Microsoft (2022f) *dotnet add package*

https://learn.microsoft.com/en-us/dotnet/core/tools/dotnet-add-package

Microsoft (2022g) *Manage packages with the Visual Studio Package Manager Console (PowerShell)*

https://learn.microsoft.com/en-us/nuget/consume-packages/install-use-packages-powershell

Microsoft (2023a) *Tutorial: Create a web API with ASP.net Core*

https://learn.microsoft.com/en-us/aspnet/core/tutorials/first-web-api?view=aspnetcore-7.0&tabs=visual-studio

Microsoft (2023b) *Migrations Overview*

https://learn.microsoft.com/en-us/ef/core/managing-schemas/migrations/?tabs=vs

Microsoft (2023c) *DbContext Lifetime, Configuration, and Initialization*

https://learn.microsoft.com/en-us/ef/core/dbcontext-configuration/

Microsoft (2023d) *Enable Cross-Origin Requests (CORS) in ASP.NET Core*

https://learn.microsoft.com/en-us/aspnet/core/security/cors?view=aspnetcore-6.0

Microsoft (2023e) *EntityFrameworkServiceCollectionExtensions.AddDbContext Method*

https://urly.fi/36gz

Microsoft (2023f) *Entity Framework Core tools reference – Package Manager Console in Visual Studio*

https://learn.microsoft.com/en-us/ef/core/cli/powershell

Microsoft (2023g) *Cd*

https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/cd

MySQL (2023) *11.3.4 The BLOB and TEXT Types*

https://dev.mysql.com/doc/refman/8.0/en/blob.html

Apple (2022) *TCP and UDP ports used by Apple software products*

https://support.apple.com/en-us/HT202944

Savonia (n.d.) *THY23 ohjelmointi C#:lla, 4op*

http://webd.savonia.fi/home/ktrasse/mat_tiedostot/thy8s/thy23/ohjelmointi/olio.php

Npgsql (n.d.) *Npgsql Entity Framework Core Provider*

https://www.npgsql.org/efcore/

NuGet (2023a) *Microsoft.EntityFrameworkCore.Tools 7.0.5*

https://www.nuget.org/packages/Microsoft.EntityFrameworkCore.Tools

NuGet (2023b) *Npgsql.EntityFrameworkCore.PostgreSQL 7.0.4*

https://www.nuget.org/packages/Npgsql.EntityFrameworkCore.PostgreSQL

HubSpot (2022) *What Are JSON Files & How Do You Use Them?*

https://blog.hubspot.com/website/json-files

W3Schools (n.d. a) *React Getting Started*

https://www.w3schools.com/react/react_getstarted.asp

W3Schools (n.d. b) React JSX

https://www.w3schools.com/react/react_jsx.asp

W3Schools (n.d. c) *HTML Tag*

https://www.w3schools.com/tags/tag_div.ASP

W3Schools (n.d. d) *HTML <h1> to <h6> Tags*

https://www.w3schools.com/tags/tag_hn.asp

W3Schools (n.d. e) *HTML <button> Tag*

https://www.w3schools.com/tags/tag_button.asp

W3Schools (n.d. f) *onclick Event*

https://www.w3schools.com/jsref/event_onclick.asp

React (2023a) *Your First Component*

https://react.dev/learn/your-first-component

React (2023b) *UseState*

https://react.dev/reference/react/useState

CodingTheSmartWay (2022) *How To Fetch API Data With React*

https://www.codingthesmartway.com/how-to-fetch-api-data-with-react/

MDN Web Docs (2023a) *CSS: Cascading Style Sheets*

https://developer.mozilla.org/en-US/docs/Web/CSS

MDN Web Docs (2023b) *Export*

https://developer.mozilla.org/en-US/docs/web/javascript/reference/statements/export

MDN Web Docs (2023c) *Console: log() method*

https://developer.mozilla.org/en-US/docs/Web/API/console/log

MDN Web Docs (2023d) *Intl.DateTimeFormat*

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Intl/DateTimeFormat

Android Developers (2021) *USB host overview*

https://developer.android.com/guide/topics/connectivity/usb/host

Android Developers (2022a) *Introduction to activities*

https://developer.android.com/guide/components/activities/intro-activities

Android Developers (2022b) *Build web apps in WebView*

https://developer.android.com/develop/ui/views/layout/webapps/webview

Android Developers (2023b) *View*

https://developer.android.com/reference/android/view/View

Android Developers (2023c) *Connect to the network*

https://developer.android.com/training/basics/network-ops/connecting

Android Developers (2023d) *JavascriptInterface*

https://developer.android.com/reference/android/webkit/JavascriptInterface

Android Developers (2023e) *Add build dependencies*

https://developer.android.com/build/dependencies

Android Developers (2023f) *Build and run your app*

https://developer.android.com/studio/run

GeeksforGeeks (2022) *Android – Cleartext HTTP Traffic Not Permitted*

https://www.geeksforgeeks.org/android-cleartext-http-traffic-not-permitted/

StarPRNT SDK Users Manual (n.d. a) *2 How to add the library*

https://www.star-m.jp/products/s_print/sdk/starprnt_sdk/manual/android_java/en/configure_application.html

StarPRNT SDK Users Manual (n.d. b) *3.1. StarIOPort*

https://www.star-m.jp/products/s_print/sdk/starprnt_sdk/manual/android_java/en/api_stario_port.html?highlight=starioport

StarPRNT SDK Users Manual (n.d. c) *4.1. StarIoExt*

https://www.star-m.jp/products/s_print/sdk/starprnt_sdk/manual/android_java/en/api_starioext.html?highlight=starioext

StarPRNT SDK Users Manual (n.d. d) 4.2 *ICommandBuilder*

https://www.star-m.jp/products/s_print/sdk/starprnt_sdk/manual/android_java/en/api_starioext_i_command_builder.html

Liite 1: Aineistonhallintasuunnitelma

Kehitysprojektin suunnittelun ja kehityksen aikana kerätään muistiinpanoja vinkeistä ja ideoista toimeksiantajalta Teams-palavereissa. Näitä Teams-palavereja pidetään kaksi kertaa viikossa tiistaisin ja torstaisin niin kauan kuin nähdään tarpeelliseksi, eli siihen asti, kunnes Android-sovellus on valmis. Muistiinpanot säilyvät pääasiallisesti työpaikan kannettavalla tietokoneella, joista projektia luodessa on helppo katsoa, että millaisella tavalla jonkun tietyn ominaisuuden on toivottu toimivan. Varmuuskopioita näistä muistiinpanoista tehdään pilvipalveluun aika-ajoin aina muutosten tapahtuessa.

Kehitysprojektin aikana pidetyistä palavereista saaduista ideoista kootaan mahdollinen kokonaisuus, jonka luominen ja loppuun vieminen on vapaasti toteutettavissa. Tämä kokonaisuus tehdään myös kokonaan työpaikan kannettavalla tietokoneella lisenssien ja ohjelmistojen takia, mutta projektista on olemassa varmuuskopio pilvessä.

Kehitysprojektin aikana luodut muistiinpanot ja sovellukset tullaan säilyttämään ainakin vuoden ajan opinnäytetyön hyväksymisestä lähtien tulosten varmistamisen takaamiseksi.

Projektia luodessa sairaaloiden potilastietoihin tai ajanvarauksiin ei ole pääsyä, joten kaikki projektissa käytetty potilaisiin kohdistuva informaatio on kuvitteellista ja itse keksittyä.

Projektin pääasiallista sovellusta tullaan toivottavasti käyttämään useita vuosia sairaaloissa ja muissa terveydenhuoltoon liittyvissä toimipisteissä.

Liite 2: React-käyttöliittymän koodi kokonaisuudessaan

```
import React, { useState } from 'react';
import './App.css';

function App() {
      const [patients, setPatients] = useState([]);

      const barcodeFilter = (barcode) => {
                fetch(`http://192.168.1.236:5140/api/Patients/${barcode}`)
                .then(response => response.json())
                .then((patients) => {setPatients(patients);

                const dateString = patients[0].time;
                const date = new Date(dateString);
                const options = {
                          day: 'numeric',
                          month: 'long',
                          year: 'numeric',
                          hour: 'numeric',
                          minute: 'numeric',
                          timeZone: 'UTC,
                };
                const formatter = new Intl.DateTimeFormat('fi-FI',
                options);
                const formattedDate = formatter.format(date);

                window.Android.connectPrinter(
                          formattedDate,
                          patients[0].room,
                          patients[0].description,
                          patients[0].queueNumber
                )
                })
                .catch((err) => {console.log(err.message);
                });
      }

      return (
                <div className="Page">
                          <h2>Lue henkilökortti</h2>
                          <button onClick = {() =>
barcodeFilter("string")}>Lue viivakoodi</button>
                </div>
      );
}

export default App
```

Liite 3: Android-sovelluksen koodi kokonaisuudessaan

```
     class MainActivity : AppCompatActivity() {

           override fun onCreate(savedInstanceState: Bundle?) {
           setContentView(R.layout.activity_main)

           val webView: WebView = findViewById(R.id.webview)
           webView.settings.javaScriptEnabled = true
           webView.addJavascriptInterface(WebAppInterface(this, "Android")
           webView.loadUrl("http://192.168.1.236:3000/")
}

class WebAppInterface(private val mContext: Context) {
           @JavaScriptInterface
           fun connectPrinter(
                      time: String,
                      room: String,
                      guide: String,
                      queue: Int
           ) {
           var port: StarIOPort? = null

           try {
                      port = StarIOPort.getPort("USB:", "", 10000, mContext)

                      val builder =
StarIoExt.createCommandBuilder(StarIoExt.Emulation.StarLine)
           var status = port.beginCheckedBlock()

           val words = guide.split(" ")
           var line = ""
           var wrappedText = ""
           for (word in words) {
                      if (line.length + word.length > 23) {
                                 wrappedText += "$line\n"
                                 line = ""
                      }
                      line += "$word "
           }
           wrappedText += line.trim()

           val dateAndTime = time.replace(" klo", "\nklo")

           builder.beginDocument()

           builder.appendAlignment(ICommendBuilder.AlignmentPosition.Center)
           builder.appendMultiple("Ilmoittautuminen\nvastaanotolle\n".toByteArra
y(), 2, 2)
           builder.append("\n----------------------------------------
\n".toByteArray())
           builder.append("Käynti:\n".toByteArray())
           builder.appendMultiple("$dateAndTime\n".toByteAttay()2, 2)
           builder.append("\n----------------------------------------
\n".toByteArray())
           builder.appendMultiple("TAYS\n".toByteArray(), 3, 3)
           builder.appendMultiple("Lääkärin vastaanotto\n".toByteArray(), 2, 2)
```

```
            builder.appendMultiple("$room\n".toByteArray(), 2, 2)
            builder.appendMultiple("\n$wrappedText\n".toByteArray(), 2, 2)
            builder.append("\n----------------------------------------
\n\n".toByteArray())
            builder.appendMultiple("Vuoronumero:\n\n".toByteArray(), 1, 1)
            builder.appendMultiple("$queue".toByteArray(), 6, 6)
            builder.append("\n".toByteArray())
            builder.append("Teidät kutsutaan sisään tällä
vuoronumerolla.".toByteArray())

            builder.appendCutPaper(ICommandBuilder.CutPaperAction.PartialCutWithF
eed)
            builder.endDocument()

            val command = builder.commands
            port.writePort(command, 0, command.size)
            status = port.endCheckedBlock()
            StarIOPort.releasePort(port)

} catch (e: StarIOPortException) {
            Log.d("Connecting", "${e.message}")
} finally {
            StarIOPort.releasePort(port)
}
```