Tampereen ammattikorkeakoulu

Koko pinon mobiilisovelluksen kehitystyö

Pauli Mänty

OPINNÄYTETYÖ
Joulukuu 2020

Tieto- ja viestintätekniikka
Ohjelmistotekniikka

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tieto- ja viestintätekniikka
Ohjelmistotekniikka

MÄNTY, PAULI
Koko pinon mobiilisovelluksen kehitystyö

Opinnäytetyö 21
Joulukuu 2020

Työn tarkoituksena oli toteuttaa mobiilisovellus, jonka avulla voidaan tallentaan hoivakodissa asuvien ihmisten muistoja teksti- ja kuvamuodossa tietokantaan. Sovellus rakennettiin käyttäen monipuolisia ja moderneja sovelluskehyksiä sekä nykyaikaisempia tekniikoita ja teknologioita.

Sovellusksen käyttöliittymä toteutettiin React Native:lla ja palvelinpuoli toteutettiin Node.js kehyksellä, Express.js kirjastolla MongoDB -tietokannalla ja Keycloak käyttäjienhallintaohjelmistolla. Tietoja kuljetetaan turvallisesti ja tietoja näytetään vain tunnistautuneille käyttäjille.

Työn tuloksena on yksinkertainen mobiilikäyttöliittymä ja palvelintoteutus, joiden avulla voidaan tallentaa ihmisten muistoja tietokantaan ja katsella näitä tietoja.

Asiasanat: kirjoita sanat pienillä alkukirjaimilla

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
ICT-Engineering
Software Engineering

MÄNTY, PAULI:
Full-stack mobile software development

Bachelor's thesis 21 pages
December 2020

Start to write here.

Next section starts here.

Key words: kirjoita sanat pienillä alkukirjaimilla

SISÄLLYS

ERITYISSANASTO

Full Stack	Koko pinon sovellus, joka sisältää sovelluksen etu- ja takapään
CRUD	Create, read, update and delete, Jatkuvan tallennuksen perustoiminnot
HTTP	Hypertext Transfer Protocol, selainten ja WWW-palvelinten tiedonsiirtoon hyödyntämä protokolla
HTTPS	Hypertext Transfer Protocol Secure, on HTTP-protokollan ja TLS/SSL -protokollan yhdistelmä
TLS/SSL	Transport Layer Security/Secure Sockets Layer, salausprotokolla, jolla voidaan suojata tietoliikennettä IP-verkkojen yli
JSON	JavaScript Object Notation, Avoimen standardin tiedostomuoto
JWT	JSON Web Token, Avoimen standardin menetelmä käyttöoikeustietueiden hallinnoimiseen eri ohjelmistojen välillä
API	Application Programming Interface -ohjelmointirajapinta, jonka avulla voidaan vaihtaa tietoja ohjelmistojen välillä
REST	Representational State Transfer – HTTP-protokollaan perustuva arkkitehtuurimalli ohjelmointirajapintojen luomiseen
NPM	Node Package Manager. Paketinhallintajärjestelmä Node.js -ympäristön paketteja varten
interface	Tapa kuvata tietoa TypeScript ohjelmointikielessä

1 JOHDANTO

Opinnäytetyössä toteutettiin koko pinon sovellus Node.js ja React Native -teknologioilla. Työ tehtiin yhteistyössä Pepron Oy:n kanssa.

Pepron Oy:n asiakkailla oli tarve saada mobiilisovellus, jonka avulla he voisivat tallentaa hoivakodin potilaiden muistoja teksti- ja kuvamuodossa mobiilikäyttöliittymän avulla. Omaiset ja potilaat voisivat luoda sovellukseen muistoja potilaiden aiemmasta elämästä, ja hoitajat voisivat nähdä nämä muistot tämän sovelluksen avulla. Kun hoitaja tulee hoitamaan potilasta, on hänen helpompi luoda potilaan olo turvalliseksi ja nostattaa luottamusta puhumalla potilaalle hänelle tutuista asioista muistojen avulla.

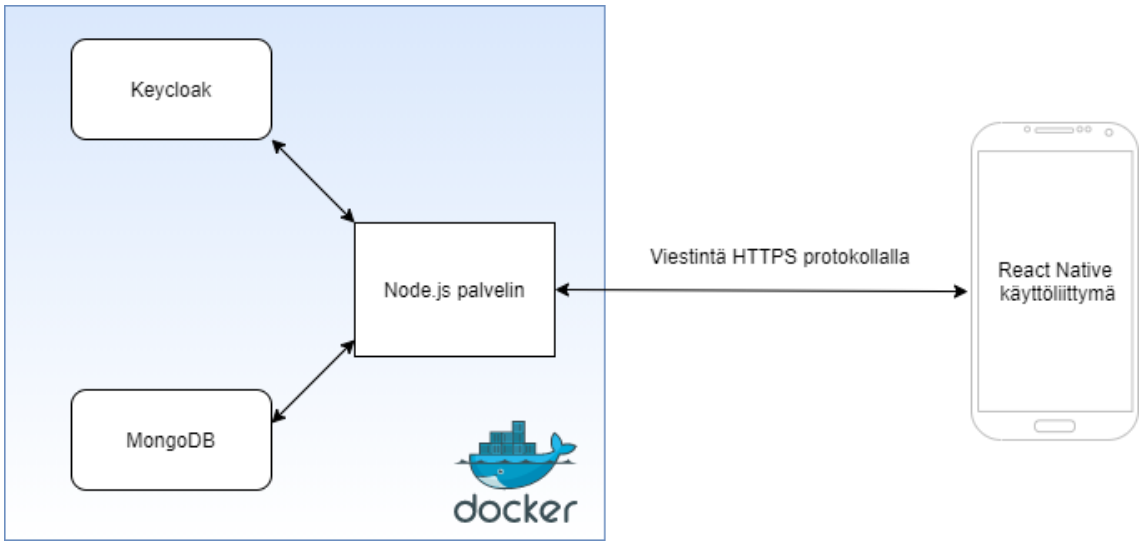
Sovelluksen käyttäjien täytyy pystyä kirjautumaan sisään mobiilikäyttöliittymän kautta sovellukseen. Eri käyttäjillä on mahdollista olla eriarvoisia käyttöoikeuksia sovelluksessa.

Asiakkaan toimesta vaatimuksena oli, että sovellus toimii sekä Googlen Android- että Applen iOS-alustalla. Tämä on sovelluksen ainoa teknologiavalintoja rajoittava vaatimus asiakkaan toimesta. Sovelluksen käyttöliittymä kehitettiin React Native:lla, koska React Native kääntyy sekä Android että iOS alustalle.

Sovelluksen on toimittava kahdella eri alustalla, eli käyttöliittymän kehittämiseen on valittava cross-platform-kirjasto. Sovelluksen käyttöliittymän kautta olisi mahdollista tallentaa kuvia ja tekstiä verkkoyhteyden yli. Sovellus tarvitsee siis tietokannan, ja rajapinnan siirtämään tietoa tietokannan ja käyttöliittymän välillä. Kaikki edellä mainitut sovelluksen osat täytyy olla käytettävissä vain tunnistetuille käyttäjille. Työssä esitellään tarvittavat teknologiat toteutusta varten.

2 SOVELLUKSEN ARKKITEHTUURI JA SUUNNITTELU

Sovelluksen takapää on huomattavasti monimutkaisempi arkkitehtuuriltaan ja toiminnallisuuksiltaan kuin käyttöliittymä.



KUVIO 1. Sovelluksen arkkitehtuuri pelkistettynä

Sovelluksen takapää koostuu kolmesta eri mikropalvelusta: Node.js-palvelin, Keycloak käyttäjienhallinta ja MongoDB -tietokanta. Mikropalveluiden käyttöönottoon käytetään Dockeria, jotta tarvittaessa sovelluksen koko takapään mikropalveluineen voidaan pystyttää lähes mihin tahansa palvelimelle, joka tukee Dockeria.

Sovelluksen etupää, eli käyttöliittymä on toteutettu React Nativella ja kommunikoi yllä mainituista mikropalveluista vain Node.js-palvelimen kanssa. Viestintä Node.js palvelimen kanssa täytyy toimia HTTPS-protokollalla, sillä HTTP-protokolla ei ole tarpeeksi turvallinen sovelluksen käyttötarkoituksessa. Pelkällä HTTP-protokollalla yhteys ei ole suojattu. Tällöin kuka tahansa voisi kuunnella liikennettä käyttöliittymän ja palvelimen välillä, ja näin anastaa käyttäjätunnuksia. Puolestaan HTTPS-protokollalla yhteys on salattu, joka tarkoittaa sitä, että käyttäjätunnukset ovat myös salattuja lähettäessä ja verkossa kuuntelevat osapuolet eivät näe salattuja tietoja selkokielisenä (Cloudflare: HTTP vs HTTPS 2020).

3 TEKNOLOGIAT

3.1 JavaScript ja TypeScript -ohjelmointikielet

JavaScript on prototyyppipohjainen, yksisäikeinen, dynaaminen ohjelmointikieli. JavaScript tukee monia ohjelmointitapoja kuten olio-ohjelmointia, imperatiivista sekä funktionaalista ohjelmointityyliä. JavaScript tunnettaan parhaiten verkkoselaimessa toimivana kielenä, mutta sitä käytetään paljon myös ei-selainpohjaisissa ympäristöissä kuten Node.js. (Mozilla Developer: Javascript 2020)

TypeScript on avoimen lähdekoodin kieli, joka perustuu JavaScriptiin, joka on yksi maailman eniten käytettyjä työkaluja, lisäämällä staattisia tyyppimäärityksiä. Tyypit tarjoavat tavan kuvata objekteja, tarjota parempaa dokumentaatiota ja antaa TypeScriptille mahdollisuuden tarkistaa, että koodisi toimii oikein.

Kaikki kelvollinen JavaScript koodi on myös TypeScript koodia. Saatat saada tyypin tarkistusvirheitä, mutta se ei estä sinua suorittamasta JavaScriptiä. TypeScript koodi muunnetaan JavaScript koodiksi TypeScript kääntäjän tai Babelin avulla. Tämä JavaScripti on puhdas, yksinkertainen koodi, joka toimii kaikkialla, missä JavaScript toimii. (TypeScript: TypeScript 2020)

3.2 Docker-konttiteknologia

Docker on avoin alusta sovellusten kehittämiseen, toimittamiseen ja käyttämiseen. Dockerin avulla voit erottaa sovelluksesi infrastruktuuristasi, jotta voit toimittaa ohjelmistoja nopeasti. Dockerin avulla voit hallita infrastruktuuria samalla tavalla kuin hallitset sovelluksiasi. Hyödyntämällä Dockerin menetelmiä koodin testaamiseen, lähettämiseen ja käyttöönottoon, voit vähentää merkittävästi viivettä koodin kirjoittamisen ja tuotantoon siirtämisen välillä. (Docker: Docker Docs 2020)

Kuten kuvio 1 osoittaa, sovelluksen takapäätä ajetaan Docker ympäristössä. Dockerin avulla kyseessä olevan infrastruktuurin ylläpitäminen ja kehittäminen on tehty yksinkertaiseksi.

Dockerissa jokaisesta sovelluksen mikropalvelusta luodaan ensin Docker-kuva. Docker-kuva on tiedosto, jonka tarkoituksena on toimia ohjekirjana sille, kuinka Docker-kontti pystytetään. Docker-kuvat luodaan Dockerfilen avulla.

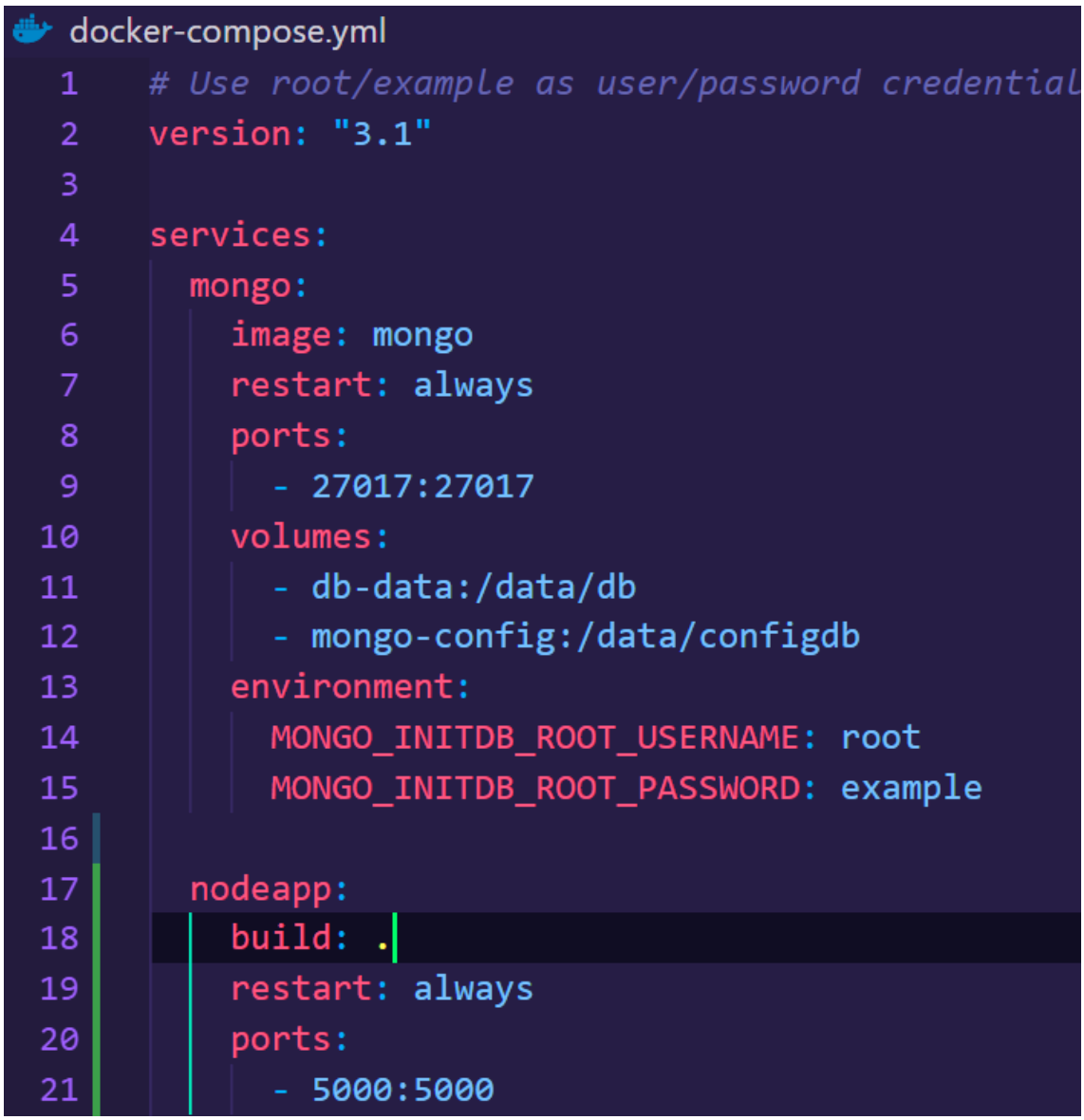
Dockerfile

```
1   FROM node:latest
2
3   WORKDIR /usr/app
4
5   COPY package.json .
6   COPY tsconfig.json .
7
8   RUN npm install
9   RUN npm install -g typescript
10
11  COPY . .
12
13  RUN tsc
14
15  EXPOSE 5000
16  CMD [ "node", "build/index.js" ]
```

KUVA 1. Esimerkki Dockerfilestä

Dockerfilessä määritellään Docker-kuvalle pohja. Tähän pohjaan tässä tapauksessa otettiin pohjaksi uusin Node.js. Seuraavaksi määritetään WORKDIR ja sinne kopioitavat tiedostot COPY komennolla. Kun tarvittavat komennot on annettu, voidaan lopuksi käyttää komentoa CMD, jolla käsketään konttia käynnistämään Node.js palvelimen koodi.

Dockerin avulla voidaan tehdä hieman Dockerfileä vastaava deskriptiivinen tiedosto, jolla voidaan määrittää koko sovelluksen infrastruktuuri. Tämä tehdään docker-compose.yml tiedostossa.



KUVA 2. Esimerkki docker-compose.yml tiedostosta

Kuvassa 2 on määritetty “services” sanan alle sovelluksen infrastruktuuriin kuuluvat palvelut. Tässä esimerkissä docker-compose.yml tiedostoon on määritetty palveluiksi MongoDB ja Node.js sovellus. MongoDB:lle on annettu valmis kuva Dockerhubista, joka on palvelu mihin voidaan julkaista valmiita Docker-kuvia. Node.js sovellukselle on määritetty oma Dockerfile, jolla kuva muodostetaan. Tässä vaiheessa voidaan myös määritellä muita Docker-kohtaisia asioita kuten, mistä portista konttiin voidaan yhdistää, halutaanko kontin käynnistävän itsensä uudelleen vikatilanteissa tai ympäristömuuttujia, kuten käyttäjätunnuksia ja salasanoja.

3.3 Keycloak

Keycloak on avoimen lähdekoodin ohjelmisto käyttäjien hallintaan. Keycloak tarjoaa matalan kynnyksen käyttäjien tunnistamisen ilman suurta määrää koodin kirjoittamista. (Keycloak: About 2020)

Keycloak toimii parhaiten, jos se halutaan integroida selainpohjaiseen sovellukseen. Keycloak tarjoaa tähän oman sisäänkirjautumisikkunan, johon käyttäjä syöttää tunnuksensa, jonka jälkeen Keycloak hoitaa loput asioista taustalla ilman, että kehittäjän tarvitsee huolehtia tunnistamiseen liittyvistä teknisyyksistä. Keycloak tukee suurinta osaa sosiaalisen median alustojen kirjautumisen mahdollisuuksista.

Keycloak tarjoaa rajapinnan, jossa on päätepisteet kaikille toiminnallisuuksille. Tätä kautta voidaan ohittaa Keycloakin oman sisäänkirjautumisikkunan käyttäminen ja tehdä kirjautuminen suoraan rajapintakutsulla mobiilikäyttöliittymästä.

3.4 JSON Web Tokens -teknologia

JSON Web Tokens on avoimen lähdekoodin ja standardin RFC 7519 mukainen ohjelmistokehys käyttöoikeustietueiden välittämiseen kahden osapuolen välillä (JSON Web Tokens 2019). JSON Web Tokensin avulla on helppo luoda, allekirjoittaa ja varmentaa tietueita, joiden avulla voidaan välittää käyttäjälle kirjautumistunnuksien perusteella yksilöllinen avain. Jokaisen seuraavan kutsun yhteydessä avain lähetetään uudelleen ja palvelin varmentaa avaimen. Avaimen varmistuksen onnistuessa käyttäjälle palautetaan pyydetyt resurssit. Muussa tapauksessa tietojen palauttaminen pysäytetään ja käyttäjää pyydetään kirjautumaan sisään uudelleen saadakseen uuden avaimen.

JSON Web Token koostuu kolmesta osasta, jotka ovat erotettu pisteellä.
Nämä kolme osaa ovat:

- ylätunniste
- hyötykuorma
- allekirjoitus

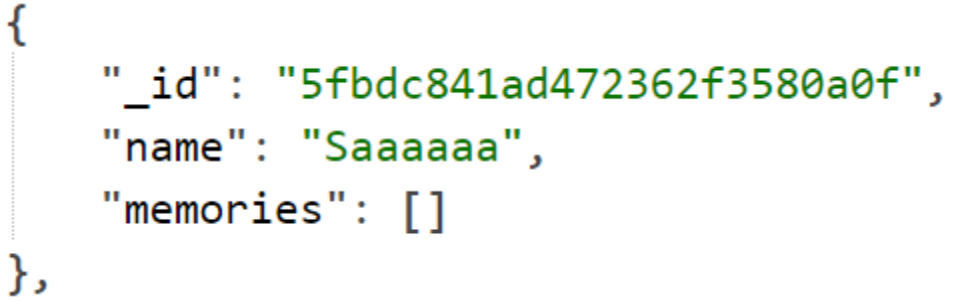
KUVA 3. Esimerkki JSON Web Tokenista (JSON Web Token 2020)

Kuvassa kolme punainen osa kuvaa avaimen ylätunnistetta. Ylätunnisteessa ilmoitetaan käytetty algoritmi ja tietuetyyppi. Seuraavaksi violetilla tekstillä ilmoitettuna on vapaamuotoinen hyötykuorma. Tämän sisälle voidaan Keycloakin käyttötapauksessa tuoda tiedot käyttäjän oikeuksista sovelluksen sisällä. Viimeisenä sininen teksti esittää allekirjoitusta.

3.5 MongoDB

MongoDB on dokumenttitietokanta, joka on suunniteltu kehittämisen ja skaalaamisen helpottamiseksi (MongoDB 2020). MongoDB on eräs nykypäivän suosituimmista tietokannoista sen helppokäyttöisyyden vuoksi.

MonboDB tietokantaan kirjoitetaan JSON (JavaScript Object Notation) dokumentteja. Dokumenttien rakenteella ei ole väliä, joka tekee tietokannan muodostamisesta ja käyttämisestä helppoa. Tästä on eniten hyötyä erityisesti sovelluksen kehitysvaiheessa, koska käytettävät tietomallit voivat muuttua useaan otteeseen kehitystyön aikana.



KUVA 4. Dokumentti MongoDB-tietokannassa. Dokumentissa on 3 saraketta, _id, name ja memories.

3.6 Node.js-kehys,

Node.js on asynkroninen tapahtumapohjainen sovelluskehys JavaScript ohjelmointikielelle. Node.js on suunniteltu rakentamaan skaalautuvia verkkopalveluja. (Node.js: About 2020)

Node.js palvelimen tehtävänä on toimia sovelluksen ohjelmointirajapintana eli API:na. API:n tarkoitus on mahdollistaa pyyntöjen tekeminen ja tietojen vaihtaminen eli keskustella sovelluksen muiden osien kanssa. Tämän työn tapauksessa keskustelu tapahtuu pääosin sovelluksen käyttöliittymän kanssa.

Node.js on avoimen lähdekoodin cross-platform kirjasto, joka ajaa JavaScript koodia verkkoselaimen ulkopuolella. Node.js mahdollistaa palvelimien kehittämisen JavaScriptillä lähes jokaisessa käyttöjärjestelmässä.

Node.js sovelluksiin voidaan lisätä ulkoisia kirjastoja NPM:n avulla. NPM on Node.js:n oma pakettienhallintatyökalu, jolla on mahdollista asentaa ja päivittää kolmansien osapuolien tekemiä kirjastoja.

Tärkein tässä työssä käytetty ulkoinen kirjasto on Express.js, jonka avulla Node.js palvelimesta tehdään REST-rajapinta. Express.js avulla koodissa voidaan määrittää palvelimelle päätepisteitä, ja mitä CRUD metodeja kyseessä olevissa päätepisteissä voidaan käyttää.

3.7 React Native -kehys

React Native on Facebookin kehittämä ja ylläpitämä avoimen lähdekoodin cross-platform-kehys. (React Native 2020). React Nativea kirjoitetaan JavaScriptilla ja käännetään natiiviksi.

Android-kehityksessä sovelluksen näkymiä kirjoitetaan Kotlinilla tai Javalla. IOS-kehityksessä käytät Swift tai Objective-C kieliä. React Nativen avulla voit kutsua näitä näkymiä React-komponenttien avulla. Ajon aikana React Native luo kyseisille komponenteille vastaavat Android- ja iOS-näkymät. Koska React

Native -komponentit tukevat samoja näkymiä kuin Android ja iOS, React Native -sovellukset näyttävät, tuntuvat ja toimivat kuten kaikki muut natiivisovellukset.

```
import React from 'react';
import { Text, View } from 'react-native';

const HelloWorldApp = () => {
  return (
    <View
      style={{
        flex: 1,
        justifyContent: "center",
        alignItems: "center"
      }}>
      <Text>Hello, world!</Text>
    </View>
  )
}
export default HelloWorldApp;
```

KUVA 5. React Native koodia

4 SOVELLUKSEN KEHITYSTYÖ

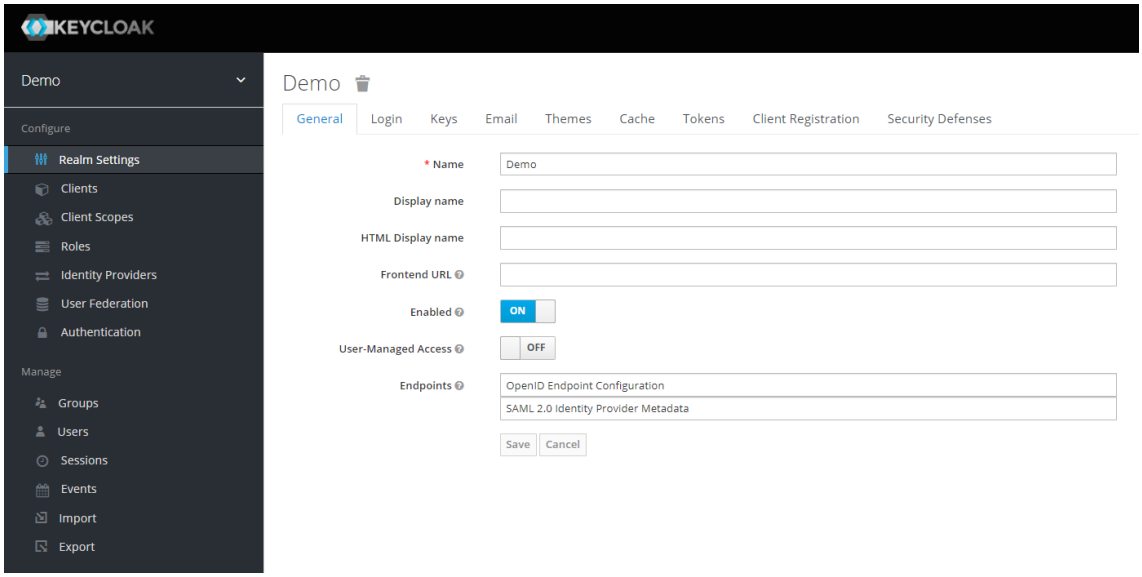
4.1 Keycloak

Keycloakin käyttöönotto Dockerilla on tehty erittäin helpoksi valmiilla kuvalla. Keycloakin saa pystyyn yhdellä komentorivikomennolla.

```
docker run -p 8080:8080 -e KEYCLOAK_USER=admin -e KEYCLOAK_PASSWORD=admin quay.io/keycloak/keycloak:11.0.3
```

KUVA 6. Komentorivikomento, jolla Keycloak pystytetään

Kuvassa 6 annetuilla parametreillä voidaan määrittää käyttäjätunnus ja salasana järjestelmävalvojalle. Järjestelmävalvojan paneeliin päästään käsiksi verkkoselaimen kautta.



KUVA 7. Keycloak-ympäristön oletusnäkymä

Keycloakissa eri sovelluskokonaisuudet erotellaan valtakunnilla (engl. “realm”). Valtakunnassa voidaan määritellä mitä protokollia käyttäjän tunnistautumiseen käytetään, voidaan luoda sovelluksia, jotka käyttävät kyseisen valtakunnan tunnistautumisen palveluita ja voidaan määrittää käyttäjärooleja ja luoda käyttäjiä.

4.2 Node.js palvelimen kehitys

Node.js palvelin on kehitetty TypeScriptillä, koska se helpottaa virheiden hallintaa ja lisää toimintavarmuutta sovellukseen.

```
1   export interface IResidentMemory {
2     title: string;
3     message: string;
4     isGoodMemory?: boolean;
5     pictureHash?: string;
6   }
7   export interface IResidentDetails {
8     name: string;
9     memories?: IResidentMemory[];
10  }
```

KUVA 8. Esimerkki sovelluksessa käytetyistä TypeScriptillä kirjoitetuista tyypeistä

TypeScriptissä määritetään omia tyyppejä avainsanalla interface, jotka kuvaavat sovelluksessa käytettäviä tietomalleja. Kuvassa yksi on määritelty kaksi interfacea, jotka mallintavat sovelluksessa käytettyjä tietoja. IResidentDetails kuvaa yksittäisen käyttäjän mallia tietokannassa. Käyttäjälle on tämän mallin mukaan pakko antaa nimi. Muistot eli kuvassa “memories?” kuvaa käyttäjän muistoja taulukkona IResidentMemory:jä. Muistot eivät ole pakollinen osa IResidentDetails interfacea, ja sitä merkitään kysymysmerkillä “memories?” perässä.

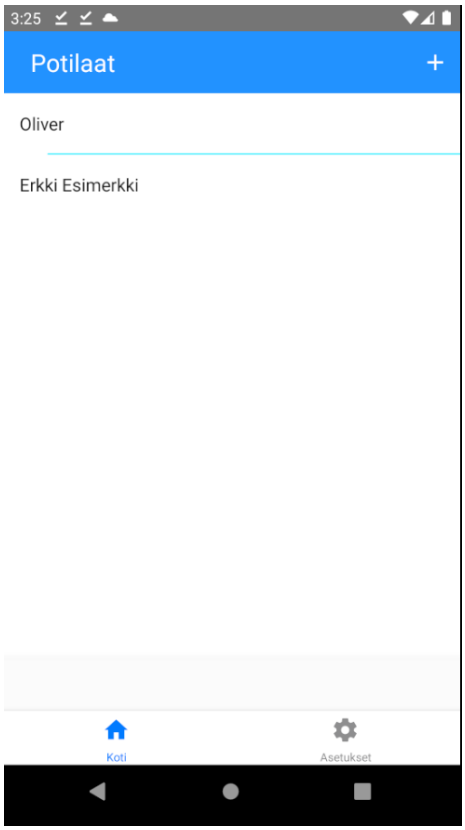
```
27  router.post("/", function (req: any, res: any) {
28    const patientName = req.body.patientName;
29    console.log(req.body);
30
31    if (patientName == null) {
32      throw new Error("INVALID NAME");
33    }
34
35    MongoClient.connect(mongoUrl, function (err: any, db: any) {
36      if (err) throw err;
37      const dbo = db.db("hetket");
38      const userToAdd: IResidentDetails = {
39        name: patientName,
40        memories: req.body.memories,
41      };
42      dbo
43        .collection("patients")
44        .insertOne(userToAdd, function (err: any, res: any) {
45          if (err) throw err;
46          console.log(`Added patient ${patientName}`);
47          db.close();
48        });
49    });
50    res.end(`Added patient ${patientName}`);
51  });
```

KUVA 9. Esimerkki Express.js-päätepisteessä ja TypeScriptin interfacen käytöstä koodissa

Yllä olevassa kuvassa (kuva 9) “router.post()” määrittää Express.js päätepisteen sovellukselle. Päätepiste “/” on palvelimen osoitteen juuri. Juureen voi siis tehdä POST pyynnön, jolla voidaan lisätä käyttäjä tietokantaan. Koodissa vakio “const userToAdd” käyttää aiemmin mainittua IResidentDetails interfacea. Tämä interfacen käyttö pakottaa tietokantaan kirjoitettavan tiedon olevan tietyn standardin mukaista, näin tietokannan sisältämät tietueet pysyvät rakenteellisesti yhdenmukaisena.

4.3 Käyttöliittymän kehitys

Sovelluksen käyttöliittymä on toteutettu React Nativella. Sovellus toimii Googlen Android- ja Applen iOS -käyttöjärjestelmillä. Sovelluksen päätoiminnot ovat tekstin ja kuvan lähettäminen ja hakeminen palvelimelta. Sovellukseen täytyy kirjautua sisään, jotta sitä voi käyttää.

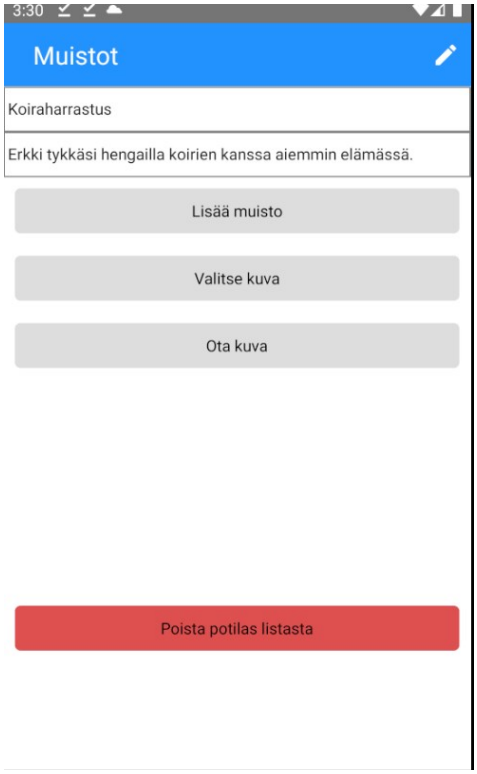


KUVA 10. Sovelluksen oletusnäkymä sisäänkirjautuneelle käyttäjälle

Sovelluksen oletusnäkymässä on lista potilaista, joiden muistoja sovelluksessa käsitellään. Klikkaamalla potilaan nimeä avautuu näkymä kyseisen henkilön muistoista.

KUVA 11. Näkymä potilaan muistoista

Muistonäkymään voidaan lisätä muistoja potilaan erilaisista elämäntilanteista. Muistoihin voidaan määrittää otsikko, selite, kuva ja onko muisto hyvä vai ei.



KUVA 12. Muiston lisääminen sovellukseen

5 POHDINTA

Opinnäytetyön tuloksena on yksinkertainen ja toimiva mobiilikäyttöliittymä sekä palvelintoteutus. Projektin alussa määritetyt ominaisuudet saatiin toteutettua. Suurin osa tekniikoista ja kehyksistä oli jo ennalta tuttuja, joten kehitystyö onnistui mutkitta tekniikoiden omien teknisten dokumentaatioiden avulla.

Vaikein ja mielenkiintoisin osuus oli koko pinon kehittäminen toimivaksi kokonaisuudeksi. Keycloakin integraatio oli vaikeampaa kuin dokumentaatio antoi olettaa, sillä Keycloakin oma React Native -kirjasto ei toiminut mainostetusti. Tämän vuoksi integraatio tehtiin Keycloakin rajapinnan kautta, joka vaati hieman enemmän työtä, sillä rajapinnan palauttama JSON Web Token täytyi hajottaa manuaalisesti käyttöliittymässä sekä rajapinnassa.

Tämän sovelluksen käyttö olisi tarkoitus aloittaa pienellä sisäisellä testiryhmällä ja jatkokehitys harkitaan tapauskohtaisesti.

Kokemuksena projekti oli erittäin opettavainen. Sovelluksen takapäätä kehittäessä tuli tutustuttua uusiin sovelluskehyksiin, joista tulee olemaan apua tulevaisuuden projekteissa. Projektia ja sen lopputulosta voidaan käyttää esimerkkinä ja apuna tulevaisuudessa vastaavia sovelluksia suunniteltaessa ja kehittäessä.

LÄHTEET

MongoDB. What is MongoDB. 2020. https://www.mongodb.com/what-is-mongodb

TypeScript. 2020. TypeScript. Luettu 06.12.2020. https://www.typescriptlang.org/

JSON Web Tokens. 2020. JWT. Luettu 06.12.2020. https://jwt.io/

Mozilla Developer. 2020. JavaScript. Luettu 06.12.2020. https://developer.mozilla.org/fi/docs/Web/JavaScript

Docker. 2020. Docker Docs. Luettu 06.12.2020 https://docs.docker.com/

Cloudflare. 2020. HTTP vs. HTTPS. Luettu 06.12.2020 https://www.cloudflare.com/learning/ssl/why-is-http-not-secure/

Keycloak. 2020. About. Luettu 06.12.2020. https://www.keycloak.org/abouta

React Navigation. 2020. Docs. Luettu 06.12.2020. https://reactnavigation.org/docs/getting-started

React Native. 2020. React Native. Luettu 06.12.2020. https://reactnative.dev/

Node.js. 2020. About Node.js. Luettu 06.12.2020. https://nodejs.org/en/about