Tampereen ammattikorkeakoulu

AWS Cogniton integraatio React web-sovellukseen

Niko Hienonen

OPINNÄYTETYÖ
Marraskuu 2020

Tietojenkäsittely
Web-tekniikat

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittely
Web-tekniikat

HIENONEN NIKO:
AWS Cogniton integraatio React web-sovellukseen

Opinnäytetyö 36 sivua
Marraskuu 2020

Käyttäjähallinnan tarve kasvaa jatkuvasti web-maailmassa. Kehityksen myötä yhä useampi sovelluksista tarvitsee keinoja antaa käyttäjien räätälöidä näkemäänsä sisältöä ja käyttämiänsä palveluita. Siinä missä sovellusten kehittäjät keksivät uusia ratkaisuja tehdä sovelluksista parempia, tehokkaampia ja hienompia, hakkerit ja muut pahaa tarkoittavat internetin käyttäjät keksivät uusia keinoja murtaa sovellusten tietoturvaa ja saada käsiinsä käyttäjien dataa. Tämän vuoksi käyttäjähallinnasta on tarpeellista kehittää toimiva sekä tietoturvallinen.

Toiminnallisen opinnäytetyön tarkoituksena on kehittää Cybercom Finlandin sisäiseen projektiin käyttäjähallinta AWS Cognitolla. Cybercom on ohjelmistotuotantoon ja pilvipalveluihin erikoistunut konsulttiyritys. Kyseessä oleva sisäinen projekti on NewbieMaker-sovellus, jolla taloon tulevat uudet työntekijät voivat tarkastella orientaatiotehtäviään. Sovelluksessa oli valmiina käsin tehty käyttäjähallinta, joka haluttiin korvata AWS Cognitolla. Toteutuksen tuli olla sillä tasolla, että sitä voitaisiin esitellä malliratkaisuna muille projekteille.

Opinnäytetyön raportissa pohditaan käyttäjähallinnan toimintaa ja vaatimuksia. Lisäksi käsitellään tietoturvan merkitystä ja toteutusta käyttäjähallinnan toteutuksessa. Raportissa esitellään myös yksinkertainen moderni web-sovellus, AWS Cognito, ja sen toiminta, jonka jälkeen käsitellään AWS Cogniton integraatiota moderniin web-sovellukseen.

Lopun pohdinnassa mietitään toteutuksen lopputulosta, prosessia itseään, mitä olisi voinut tehdä eri tavoin ja mitä kannattaa ottaa huomioon tulevaisuudessa jatkuvasti muuttuvassa web-maailmassa.

Asiasanat: aws, cognito, käyttäjähallinta, reactjs

ABSTRACT

Tampere University of Applied Sciences
Business Information Systems
Web technologies

HIENONEN NIKO:
Integrating AWS Cognito into a React Web Application

Bachelor's thesis 36 pages
November 2020

The need for user management is growing continuously in the web world. Due to technical development more applications need ways to allow the end users to customize the content they see and the services they use. Where application developers optimize their applications in looks and performance, malicious users and hackers find new ways to break the cyber security of applications and access user data. This raises the need to develop functioning and secure user management solutions in applications.

The meaning of this practice-based thesis is to develop a user management system with AWS Cognito for an internal project for Cybercom Finland. Cybercom is a consultant-based software development company specializing in cloud technology. The internal project in question is NewbieMaker, an application that new employees use to review their orientation tasks. The application had a hand-build user management system that was to be replaced with AWS Cognito.

The thesis report considers user management, its operation and requirements, how cyber security relates to it and how it can be implemented. A simple modern web application, AWS Cognito and its functionality is also presented in the report. After these topics, the report presents the integration of Cognito into the application.

In the reflection in the end there is pondering about how the result turned out, how the process went, what could have been done differently and more food for thought about the future of the ever-changing web world.

Key words: aws, cognito, user management, reactjs

SISÄLLYS

ERITYISSANASTO

AWS	Amazon Web Services, Amazonin tarjoamat verkkopalvelut.
Cognito	AWS:n tarjoama käyttäjähallinta-, ja autentikaatiojärjestelmä.
Käyttöliittymä	Web-sovelluksen osa, jonka käyttäjä näkee ja jolla käyttäjä käyttää sovellusta.
Taustaohjelma	Web-sovelluksen osa, jota käyttäjä ei näe. Käyttöliittymä yleensä keskustelee taustaohjelman kanssa datan vaihdossa.
Tietokanta	Kokoelma tietoa tietokoneella, josta voi noutaa dataa ja muuttaa sitä.
HTTP	Hypertext Transfer Protocol, web-palvelimien käyttämä tiedonsiirtoprotokolla.
Pyyntö-vastaus	HTTP-pohjainen kommunikointimenetelmä, jossa yksi osapuoli pyytää dataa, ja toinen vastaa pyyntöön.
JavaScript	Monikäyttöinen ohjelmointikieli, jolla voi esimerkiksi ohjelmoida käyttöliittymän toiminnallisuutta.
React	JavaScript-pohjainen käyttöliittymä ohjelmointikirjasto.
TypeScript	Microsoftin ohjelmointikieli. Tiukasti syntaktinen pääjoukko JavaScriptistä, joka tarjoaa esimerkiksi staattisen tyypityksen.
Node.js	JavaScript-kehys, joka ajaa selaimen ulkopuolista koodia. Käytetään yleensä taustaohjelmissa.
Express	Node.js-kehys, jolla voi rakentaa web- ja mobiiliapplikaatioita.
JWT	JSON Web Token. Standardisoitu työkalu datan turvalliseen vaihtamiseen osapuolien välillä.
MongoDB	Skaalautuva tietokantapalvelu.
OAuth	Delegoitu REST/API-käyttöoikeuksien kehys.
Redux	Avoimen koodikannan omaava JavaScript-kirjasto sovelluksien tilan hallintaan.

Flux-arkkitehtuuri	Facebookin kehittämä käyttöliittymäarkkitehtuuri. Perustuu yksisuuntaiseen datan virtaukseen.

1 JOHDANTO

Käyttäjähallinnan tarve kasvaa jatkuvasti web-maailmassa. Kehityksen myötä yhä useampi sovelluksista tarvitsee keinoja antaa käyttäjien räätälöidä näkemäänsä sisältöä ja käyttämiänsä palveluita. Siinä missä sovellusten kehittäjät keksivät uusia ratkaisuja tekevät sovelluksista parempia, tehokkaampia ja hienompia, hakkerit ja muut pahaa tarkoittavat internetin käyttäjät keksivät uusia keinoja murtaa sovellusten tietoturvaa ja saada käsiinsä käyttäjien dataa. Tämä nostaa tarpeen kehittää käyttäjähallinnasta toimiva sekä tietoturvallinen.

Käyttäjähallinnalla lyhyesti tarkoitetaan niitä keinoja ja käytäntöjä, joilla käyttäjiä ja heidän aktiviteettejansa sovelluksessa hallitaan. Tähän kuuluvat mm. käyttäjätilin luonti, sisäänkirjautuminen, uloskirjautuminen, evästekäytäntöjen hallinta, käyttäjätietojen muokkaus ja käyttäjätilin poistaminen. Käyttäjähallinnan voi suunnitella ja toteuttaa itse, jolloin saa täyden vapauden toteuttaa sen juuri niin kuin haluaa. Toinen vaihtoehto on käyttää valmista käyttäjähallintapalvelua, jonka hyödyntäminen kehityksessä rajoittaa kehityksen vapautta, mutta antaa vakautta ja tietoturvallisuutta olettaen, että valittu työkalu on luotettavasti kehitetty.

Toiminnallisen opinnäytetyön tarkoituksena on kehittää Cybercom Finlandin sisäiseen projektiin käyttäjähallinta AWS Cognitolla. Cybercom on ohjelmistotuotantoon ja pilvipalveluihin erikoistunut konsulttiyritys. Kyseessä oleva sisäinen projekti on NewbieMaker, sovellus, jolla taloon tulevat uudet työntekijät voivat tarkastella orientaatiotehtäviään. Sovelluksessa oli valmiina käsin tehty käyttäjähallinta, joka haluttiin korvata AWS Cognitolla. Toteutuksen tuli olla sillä tasolla, että sitä voitaisiin esitellä malliratkaisuna muille projekteille.

2 OPINNÄYTETYÖN TAVOITE JA TARKOITUS

Opinnäytetyöni tavoitteena on kehittää Cybercom Finlandille toimiva käyttäjähallintaratkaisu Cognitolla. Työn tarkoituksena on suunnitella ja toteuttaa uusittu käyttäjähallinta NewbieMaker-sovellukseen, jonka pohjana on Cognito. Uuden käyttäjähallinnan toiminnallisuus ja tietoturva tulee olla ammattimaisella tasolla. Ammattimaisella tasolla tarkoitetaan tässä kontekstissa sitä, että työtä voi esitellä organisaatiossa muille osapuolille havainnollistavana malliratkaisuna ja siitä voidaan ottaa mallia muissa projekteissa tarvittaessa.

Toiminnallisuuden tulee noudattaa hyviä design-käytäntöjä, esimerkiksi käyttöliittymän tulee olla selkeä, koodin tehokkuus nopeaa ja toteutus selkeästi dokumentoitu, jotta uudet tekijät ymmärtävät, miten toteutus toimii, jos projektin tekijät vaihtuvat.

Tietoturvan toteutus helpottuu Cogniton ottaessa salasanojen ja käyttäjänimien hallinnan itselleen, ja sovellukseen jää vain yleistä tietoa käyttäjistä. Mikään tietoturvariski ei siltikään ole hyväksyttävä, ja toteutus suoritetaan sen mukaan.

3 OPINNÄYTETYÖN TOTEUTUS

3.1 Aiheen valinta

Suoritin tutkintoni harjoittelun Cybercom Finlandilla. Työskentelin pääosin NewbieMaker-sovelluksen kehityksessä. NewbieMaker on web-sovellus, jossa taloon tulevat uudet työntekijät saavat orientaatiotehtäviä, ja jossa heidän esimiehensä voivat tarkastella orientaation sujumista. Sovelluksen käyttäjähallinta haluttiin modernisoida vanhasta, itsetehdystä ratkaisusta. Uudella ratkaisulla tavoiteltiin käytettävyyttä, ylläpidon helppoutta ja pilviteknologian preferointia.

Cybercom tunnetaan pohjoismaiden suurimpana AWS:n yhteistyökumppanina. Sen vuoksi NewbieMakerin käyttäjähallinta haluttiin toteuttaa AWS:n tarjoamalla Cognitolla. Projektissa toteutin uuden käyttäjähallinnan ja tein samalla opinnäytetyöni aiheesta, sillä työn tavoitteet ja tarkoitus olisivat selviä ja voisin toteuttaa opinnäytetyötäni töideni lomassa.

3.2 Tiedonhankinta

Aloitin opinnäytetyöni taustatutkimuksen tutustumalla projektin tekniikkojen dokumentaatioon ja jatkoin etsimällä tietoa enemmän abstrakteista aiheista, kuten käyttäjähallinnan filosofiasta ja käytännöstä sekä tietoturvallisesta ohjelmistokehityksestä. Suunnittelin työtä yksityiskohtaisesti ennen käyttäjähallinnan toteuttamista, koska ohjelmistotuotannossa huolellinen suunnittelu säästää aikaa kehitystyössä. Käytin tiedonhakuun pääasiallisesti Tampereen Yliopistojen sähköisten aineistojen palvelua Andoria.

4 KÄYTTÄJÄHALLINNAN RAKENNE JA TOIMINTA

4.1 Käyttäjähallinnan toiminta

Käyttäjähallinta web-applikaatioissa mahdollistaa datan tallentamisen käyttäjästä. Tällaisia asioita ovat esimerkiksi nimi, sijainti, lempisarjat jne. Käyttäjähallinnan kulmakivenä toimivat autentikointi ja valtuutus. Dasgupta, Roy ja Nag (2017, 4) kirjoittivat että autentikoinnin tarkoituksena on varmistaa käyttäjän identiteetti ja rajoittaa varmistamattomien käyttäjien pääsyä järjestelmään. Autentikointi voidaan tehdä esimerkiksi salasanalla, kaksivaiheisella varmistuksella, sormenjälkitunnistuksella tai avainpareilla. Useimmissa tapauksissa kolme ensimmäistä vaihtoehtoa riittävät. Pelkän salasanan lisäksi kaksivaiheinen varmistaminen estää salasanan väärinkäytön ja tuo lisäturvaa. Kaksivaiheinen varmistus voidaan suorittaa esimerkiksi ensin varmistamalla käyttäjän salasanan ja tämän jälkeen lähettämällä käyttäjän tilin sähköpostiin varmistuspyyntö. Autentikoinnin jälkeen käyttäjä saa valtuudet käyttää sovellusta tilinsä käyttöoikeuksien mukaisesti. Jos käyttäjä on niin sanottu ylläpitäjäkäyttäjä (admin user), hän voi mahdollisesti muokata sovellusta tai hallita muita käyttäjiä. Perustason käyttäjä ei yleensä voi nähdä muiden käyttäjien käyttäjätietoja, vaan voi muokata vain omia käyttäjätietojaan ja käyttää sovellusta.

4.2 Hyvän käyttäjähallinnan ominaisuuksia

Hyvän käyttäjähallinnan kulmakivet ovat toiminnallisuus, käytettävyys ja tietoturvallisuus. Toiminnallisuus on kolmesta edellä mainitusta hankalin määritellä yleisesti, sillä se vaihtelee jokaisen projektin mukaan, mutta ainakin sovelluksen tehokkuus, koodin ylläpidettävyys ja projektilta vaadittujen ominaisuuksien toimitus ovat avainasemassa. Jos käyttäjähallinta toimii suunnitellusti, se on tietoturvallinen ja helppokäyttöinen.

4.2.1 Käytettävyys käyttäjähallinnassa

Käytettävyydestä on kirjoitettu lukuisia teoksia jo itsessään ja sen roolia kaikessa sovelluskehityksessä ei voi aliarvioida. Kiteytettynä käytettävyys kuvailee loppukäyttäjän käyttökokemusta. Huonoa käyttäjäkokemusta kuvailevat seuraavat kysymykset: Latautuvatko sivut hitaasti? Onko käyttöliittymä sekava ja epämiellyttävän näköinen? Kysytäänkö käyttäjänluonnissa käyttäjältä epärelevantteja asioita?

Jos verkkosivujen halutaan olevan tehokkaita, niiden pitää suorittaa suurin osa taioistaan vilkaisulla. Ja paras keino tämän saavuttamiseen on luoda verkkosivuja, jotka ovat itsestään selviä tai vähinäänkin itsestään selittäviä. (Krug 2013, 1). Käytettävyys voi siis tarkoittaa loppukäyttäjän tietoisen ajattelun minimointia sivun käytön suhteen.

4.2.2 Tietoturva käyttäjähallinnassa

Jotta salasanaa voidaan käyttää autentikointiin, salasanat tulee tallentaa turvallisesti (Dasgupta, Roy & Nag 2017, 6). Salasanojen salaus hajautuksella (hashing) on yleinen keino tallentaa salasanat järjestelmään turvallisesti. Hajautuksen ideana on se, että järjestelmä osaa autentikoida oikean salasanan ilman että salasanaa tallennetaan tekstimuodossa tietokantaan (Dasgupta, Roy & Nag 2017, 6). Myös tietokannan tulee olla turvallisessa sijainnissa, jolloin käyttäjätietojen vuotaminen on epätodennäköisempää. Tämä on kriittisen tärkeää, jos käyttäjistä tallennetaan arkaluontoista tietoa sovelluksen käyttöä varten, kuten esimerkiksi henkilötunnus, pankkitunnuksia, osoitetietoja jne.

Tietoturvallisuuteen kuuluu tiedon salaamisen lisäksi myös tietoihin pääsyn rajoittaminen, vuotaneen tiedon jäljittäminen ja turvaaminen ja koodin laadukkuus. Ohjelmistovirheet voivat sallia avoimet ovat haitallisille tunkeilijoille (Amoroso 2007, 160). Tietoturvallisuus kattaa koko tuotantoketjun ohjelmistokehityksessä, jonka takia kaikkien kehityksen osapuolien tulee kantaa vastuu oman tekemisensä turvallisuudesta.

5 YKSINKERTAINEN MODERNI WEB-SOVELLUS

5.1 NewbieMaker teknologia

NewbieMaker on toteutettu käyttäen ReactJS:ää käyttöliittymässä, Node.js:ää taustaohjelmassa ja MongoDB-tietokantaa. Myös muita teknologioita on käytetty sovelluksen kehityksessä, mutta raporttiin on sisällytetty vain minimalistinen versio sovelluksen toiminnallisuudesta ja yksinkertaistetut osat koodista, joiden avulla lukija ymmärtää sovelluksen toiminnan. Käyttöliittymän koodi on kehitetty erikseen raporttia varten ja ei ole NewbieMaker-sovelluksesta, mutta käyttää samoja periaatteita.

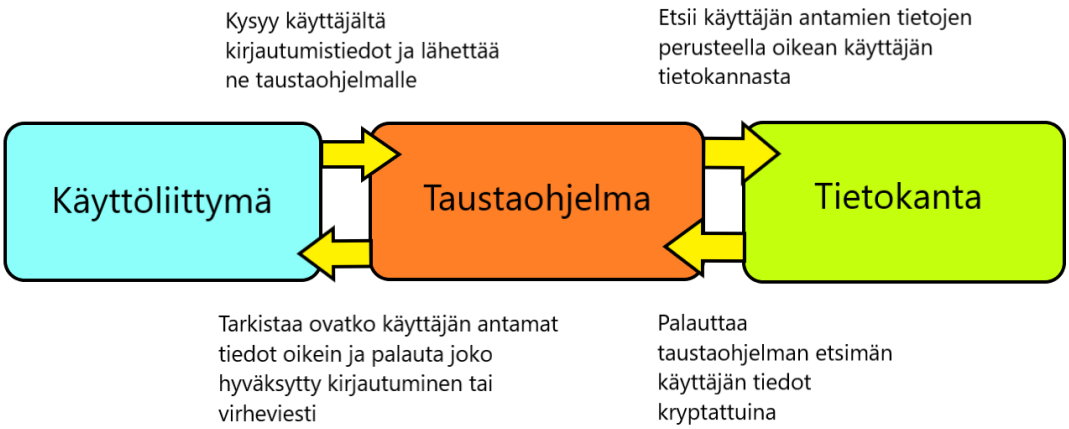
5.2 Datan kulku web-sovelluksessa

Data kulkee web-sovelluksessa HTTP:n välityksellä. HTTP toimii pyyntö-vastaus-protokollalla käyttöliittymän ja taustaohjelman välillä. Käyttöliittymästä lähetetään pyyntö taustaohjelmalle ja taustaohjelma lähettää vastauksen. Jos sovelluksessa hyödynnetään tietokantaa, taustaohjelma voi ennen vastauksen lähettämistä tarkastella tai muokata tietokannan dataa pyynnön perusteella ja sen jälkeen palauttaa vastauksen. Data kulkee siis kumpaankin suuntaan (kuva 1).

KUVA 1. Datan kulku ohjelmien välillä.

5.3 Käyttäjähallinnan dataliikenne

Yksi keskeisimmistä datan siirto-operaatioista käyttäjähallinnassa on sisäänkirjautuminen. Sisäänkirjautumisessa käyttöliittymä lähettää käyttäjän sisäänkirjautumistiedot taustaohjelmalle, joka noutaa tietokannasta käyttäjän kryptatut tiedot, vertaa niitä käyttöliittymän lähettämiin tietoihin ja palauttaa käyttöliittymälle joko hyväksytyn kirjautumisen tai virheviestin. Kuva 2 havainnollistaa edellä mainittua.



KUVA 2. Dataliikenne sisäänkirjautumisessa

5.4 Sisäänkirjautumislomake ja sen toiminta Reactissa

Jotta käyttäjä voi antaa ohjelmalle käyttäjätietonsa sisäänkirjautumista varten, tulee sovelluksen pystyä vastaanottamaan käyttäjän syötettä. Tähän tarkoitukseen on sisäänkirjautumislomake, joka antaa käyttäjän syöttää käyttäjätilinsä identifioivan tekijän (sähköposti, käyttäjänimi) ja salasanansa sovellukseen ja aloittaa autentikoinnin kuvan 2 mukaisesti.

Reactissa yksinkertaisten lomakkeiden tekeminen on vaivatonta, toiminnallisuuteen kuuluvat vain tekstisyötekomponenttien ja lomakekomponentin tilan hallinta, datan lähetys ja vastaanottaminen taustaohjelmalta ja kirjautumisen tuloksen näyttäminen käyttäjälle.

Kuvassa 3 on demo käyttöliittymän juuri, App.tsx. Reactissa sovelluksen juurena perinteisesti toimii App.js tai App.tsx riippuen siitä käytetäänkö sovelluksen kehityksessä TypeScriptia. Demossa esitetään käyttöliittymän sisäänkirjautuminen todella yksinkertaistetusti käyttämällä sovelluksen tilaa näyttämään joko sisäänkirjautumissivu tai kotisivu, jossa näytetään sisään kirjautuneen käyttäjän käyttäjänimi. Oikeassa sovelluksessa käytetään yleensä tilanhallintakirjastoa, esim. reduxia hallitsemaan aktiivisen käyttäjän tietoja.

```
function App() {
  const [userLoggedIn, setUserLoggedIn] = useState("");
  const login = (username: string) => setUserLoggedIn(username);
  const logout = () => setUserLoggedIn("");
  return (
    <div className="App">
      <h1>Login Demo</h1>
      {
        userLoggedIn
          ? <Landing username={userLoggedIn} logout={logout} />
          : <Login login={login} />
      }
    </div>
  );
}

export default App;
```

KUVA 3. Käyttöliittymän juuri, App.tsx

Kuvassa 4 on kotisivu, jonne päädytään onnistuneen kirjautumisen jälkeen. Sivulla on yksinkertainen tervetulotoivotus ja uloskirjautumispainike. Komponentti saa uloskirjautumisfunktion ja käyttäjänimen vanhemmaltaan, eli App.tsx:ltä.

```
interface LandingProps {
  logout: () => void;
  username: string;
}

const Landing: FC<LandingProps> = ({ logout, username }) => (
  <div>
    <h2>Welcome {username}!</h2>
    <button onClick={logout}>Log out</button>
  </div>
)

export default Landing;
```

KUVA 4. Kotisivu, jossa näytetään käyttäjänimi ja sallitaan uloskirjautuminen

Kuvassa 5 on sisäänkirjautumissivu, jossa on sisäänkirjautumislomake. Sisäänkirjautumislomake on toteutettu skaalautuvasti, eli kaikki tekstisyötekomponentit käyttävät samaa onChange-funktiota lomakkeen datan muuttamiseen. Komponentissa on myös lomakkeen lähetysfunktio onSubmit, joka hyödyntää apiLogin-funktiota, joka löytyy kuvasta 6. Validointia demosovelluksesta ei löydy. Oikeassa maailmassa validointi on todella tärkeää, sillä se estää käyttöliittymän väärinkäytön.

```
const Login: FC<loginProps> = ({ login }) => {
  const [formState, setFormState] = useState({ username: '', password: '' });

  async function onSubmit(e: SyntheticEvent) {
    e.preventDefault();
    const loginSuccessful = await apiLogin(formState);
    loginSuccessful ? login(loginSuccessful) : alert("Error in login");
  };

  const onChange = (e: ChangeEvent<HTMLInputElement>) => {
    e.preventDefault();
    const { id, value } = e.target;
    setFormState((previousState) => ({
      ...previousState,
      [id]: value,
    }));
  };

  return (
    <form onSubmit={onSubmit}>
      <input type="text" id="username" placeholder="Username..." value={formState.username}
        onChange={onChange} />
      <input type="password" id="password" placeholder="Password..." value={formState.password}
        onChange={onChange} />
      <input type="submit" />
    </form>
  );
};
```

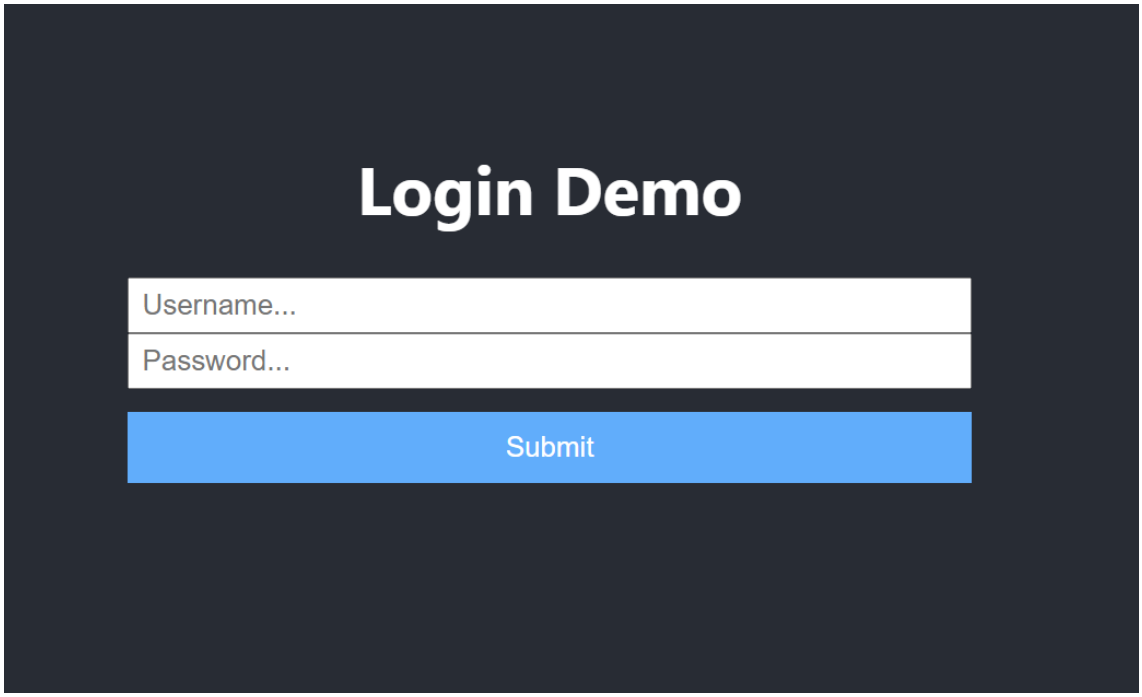
KUVA 5. Sisäänkirjautumissivu

Kuvasta 6 löytyy HTTP-apufunktiot, joissa tässä demossa on vain apiLogin, joka vastaanottaa sisäänkirjautumislomakkeen datan ja lähettää sen taustaohjelmalle. Vastauksen mukaan funktio palauttaa joko onnistuneen kirjautumisen käyttäjänimen tai tyhjän merkkijonon. Kuvan 5 mukaisesti tyhjän merkkijonon palautus apiLoginista laukaisee hälytyksen, kun taas muut palautukset laukaisevat login-funktion.

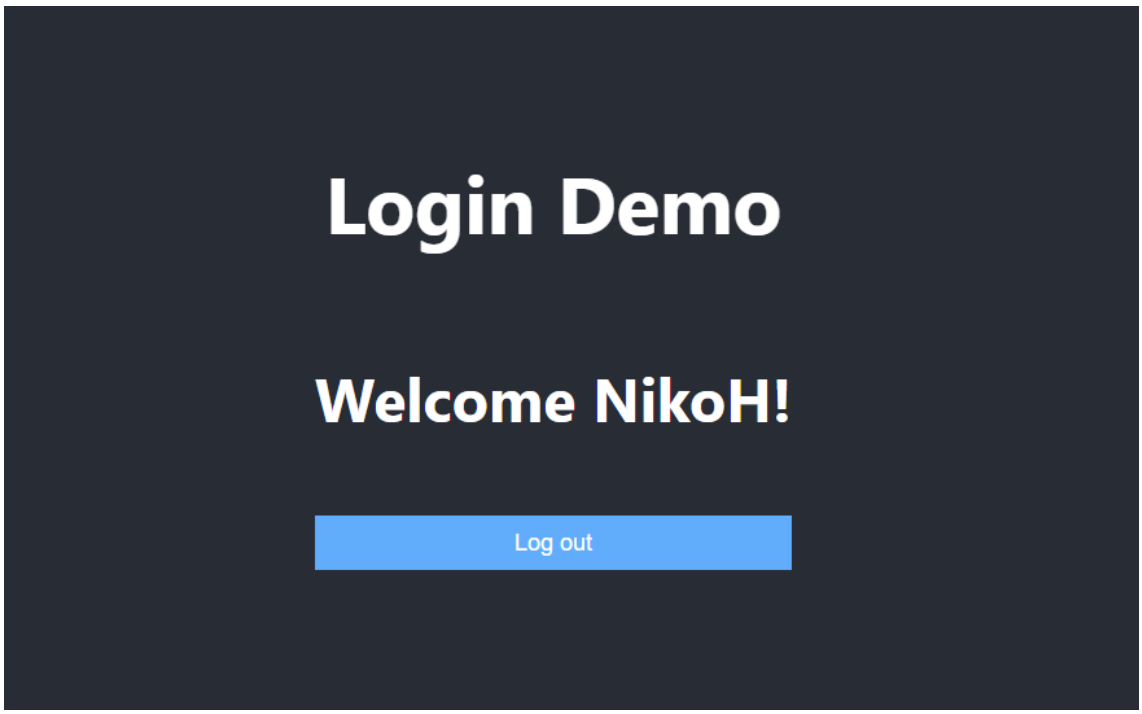
```
type User = {
  username: string;
  password: string;
}

type asyncResponse = {
  data: {
    token: {
      id: string,
      username: string,
    },
    username: string,
  },
  error?: any
}

export async function apiLogin(user: User) {
  const response: asyncResponse = await axios.post(`${config.apiBaseUrl}/login`, user);
  return response.data ? response.data.username : "";
}
```

Kuva 6. HTTP-apufunktiot.

Käyttäjälle näkyvät ruudut sisäänkirjautumisesta ja kotisivusta löytyvät kuvista 7 ja 8.



KUVA 7. Sisäänkirjautumisnäkymä



KUVA 8. Kotisivu, jonne päästään onnistuneen kirjautumisen jälkeen

5.5 Express-taustaohjelma sisäänkirjautumisen perustana

Jotta käyttöliittymä voi vaihtaa dataa tietokannan kanssa, on tärkeää käyttää taustaohjelmaa kommunikoinnin välikappaleena. Tällä tavoin tietokantaan tehtävät kutsut eivät ole näkyvissä tai muokattavissa käyttäjälle. Kuten erityissanastossa mainittiin, käyttöliittymä on osa sovelluksen koodista, jonka käyttäjä näkee ja pahimmassa tapauksessa pystyy vaikuttamaan koodiin selaimen kautta. Taustaohjelma on turvassa, sillä sen koodia ei näe selaimesta. Tämän takia on kriittistä käyttää taustaohjelmaa tietokannan kanssa kommunikointiin.

NewbieMakerin taustaohjelma on toteutettu Express-ohjelmointikehyksellä. Kuten käyttöliittymässä, näytetty koodi on yksinkertaistettu versio sovelluksen oikeasta koodista ja siinä näytetään vain keskeisimmät osat koodista. Kaikessa yksinkertaisuudessaan taustaohjelma ajetaan omassa portissaan ja se jää aktiiviseksi odottamaan pyyntöjä käyttöliittymältä ja vastaa niihin protokollan mukaan.

Tässä esimerkissä taustaohjelma kuuntelee POST-pyyntöä reitillä ”/login”. Pyynnön tullessa taustaohjelma ottaa POST-pyynnölle ominaisen ”kehon” (body) pyynnöstä ja etsii tietokannasta käyttäjää pyynnön käyttäjänimellä. Jos käyttäjänimi löytyy, verrataan salattua salasanaa pyynnön salasanaan. Jos salasanat

täsmäävät, tehdään jwt-tunnus ja lähetetään se käyttöliittymälle onnistuneen kirjautumisen merkiksi. Jos joku edellä mainituista kohdista epäonnistuu (käyttäjää ei löydy, salasanat eivät täsmää, yms.) lähetetään käyttöliittymälle asianmukainen virheilmoitus. Edellä mainittu tapahtumaketju on sisäänkirjautuminen pseudokoodattuna. JavaScriptillä tehty toteutus löytyy kuvasta 9.

```
app.post("/login", (req: Request, res: Response) => {
    const body = req.body;
    User.findOne({ username: body.username }, (user) => {
        bcrypt.compare(body.password, user.password, (err, passwordcorrect) => {
            if (passwordcorrect) {
                const token = jwt.sign({
                    username: user.username,
                    id: user.id
                },
                    process.env.SECRET!,
                )
                res.status(200).send({ token, username: user.username });
            }
        })
    })
        .catch((error) => {
            console.log("Error in login: " + error);
            return res.status(401).json({ error: "error in login" });
        });
})
```

KUVA 9. Taustaohjelman koodi sisäänkirjautumiselle

Esimerkissä on MongoDB-tietokantapalvelun syntaksilla tehty käyttäjän etsiminen tietokannasta ja bcrypt-salauskirjaston syntaksilla suoritettu salatun salasanan tarkistus. Käyttöliittymään lähetettävä jwt-tunnus on tehty jsonwebtoken-kirjastolla ja sisältää ympäristömuuttujan ”SECRET”, jolla varmistetaan tunnuksen autenttisuus. Aidoissa projekteissa lähetetään käyttäjästä todennäköisesti enemmän dataa kuin vain tunnus ja käyttäjänimi.

5.6 Itse tehty kirjautumisratkaisu

Kirjautumisen koodaaminen taustaohjelmalla vaatii osaamista jostain taustaohjelmaan soveltuvasta ohjelmointikielestä ja ymmärrystä siitä, miten taustaohjelma toteutetaan järkevästi ja tietoturvallisesti. Itse tekeminen sallii täyden vapauden toteuttaa sisäänkirjautuminen juuri kuten tekijä itse haluaa. Express-kirjaston käyttäminen perustuu muiden vapaavalinnaisten kirjastojen käyttöön, kuten bcrypt, jsonwebtoken, yms. Eli toteutuksen voi tehdä vapaasti omilla tekniikoilla,

mutta tulisi tietää mitä tekee, sillä riskinä on käyttäjätietojen vuotaminen vääriin käsiin.

Seuraavassa luvussa esitellään Cognito ja sen jälkeen luvussa 7 integroidaan Cognito React-sovellukseen tarkastellen sen hyötyjä ja kompastuskiviä verrattuna express-sovellukseen.

6 COGNITO KÄYTTÄJÄHALLINNAN POHJANA

6.1 Cogniton anatomia

Cognito on AWS:n tarjoama web-palvelu, joka tarjoaa ratkaisuja web- tai mobiiliapplikaatioiden käyttäjähallintaan ja autentikointiin. Palvelu skaalautuu muutamasta käyttäjästä miljooniin ja tukee sisäänkirjautumista sosiaalisilta identiteetin toimittajilta, kuten Facebook ja Google. Cogniton käyttämiseen tarvitsee AWS-tilin, jonka avulla pääsee käsiksi Cogniton kojelautaan, josta voi konfiguroida Cogniton toiminnan applikaation käyttäjähallinnassa.

Cogniton toiminta perustuu sen kahteen pääkomponenttiin, käyttäjäaltaisiin ja identiteettialtaisiin. Käyttäjäaltaat ovat käyttäjän kirjautumista ja käyttäjänluontia varten, identiteettialtailla käyttäjät pääsevät muihin AWS-sovelluksiin. Tässä raportissa keskitytään pelkästään käyttäjäaltaisiin.

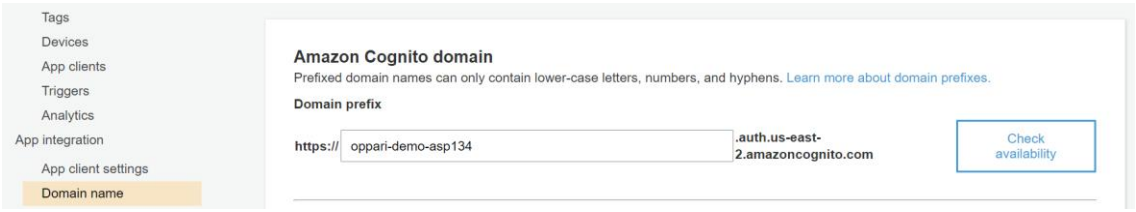
Jotta Cognitoa voi käyttää, tarvitsee AWS-tilin. Tili on ilmainen ja sen voi luoda AWS:n sivuilla. Kun tili on luotu, voi kirjautua sisään AWS-konsoliin ja alkaa luomaan altaita.

6.2 Käyttäjäaltaiden käyttö

Cogniton toiminta käyttäjähallinnassa pohjautuu käyttäjäaltaisiin. Käyttäjäaltaaseen tallennetaan sovelluksen käyttäjät ja Cognito voi sen pohjalta hallita käyttäjiä. Käyttäjäaltaan luominen vaatii altaalle nimen ja päätöksen siitä, tapahtuuko käyttäjänluonti vapaavalintaisen käyttäjänimen vai sähköpostin kautta ja tarvitaanko muita käyttäjäattribuutteja. On huomion arvoista, että näitä asetuksia ei voi enää altaan luonnin jälkeen muuttaa, joten ne kannattaa suunnitella hyvin. NewbieMakerin osalta päädyttiin käyttäjänluontiin sähköpostin kanssa, koska projektissa ei tarvita erillisiä käyttäjänimiä. Myöskään muita käyttäjäattribuutteja ei tarvita, sillä Cognitoon tallennetaan vain kirjautumistiedot käyttäjästä ja loput tiedot tallennetaan sovellukseen omaan tietokantaan. Tämän jälkeen voi luoda altaan ja alkaa käyttämään sitä.

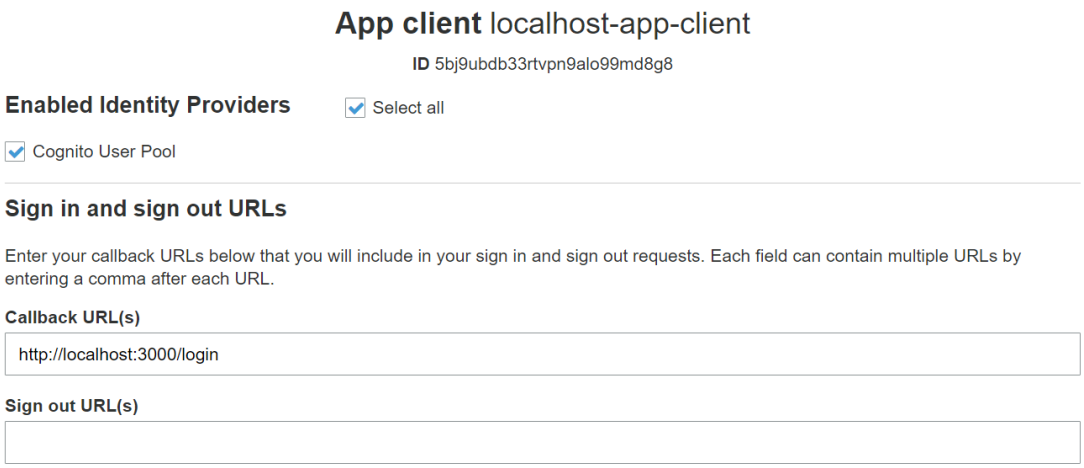
6.3 Sovelluksen valtuutus Cogniton käyttöön

Jotta Cognitoa voi käyttää autentikointiin sovelluksissa, sovellukset pitää hyväksyttää Cognitossa. Cognitossa käyttäjäaltaaseen voi liittää valtuutettuja sovelluksia. ”App Client” viittaa käyttäjäaltaaseen rekisteröityyn sovellukseen, joka saa käyttää käyttäjäaltaan palveluita. Ennen sovelluksen rekisteröintiä, tulee käyttäjäaltaalle lisätä domain, eli verkkotunnus. Tässä verkkotunnuksessa sijaitsee käyttäjäaltaan kirjautumis- ja käyttäjänluontisivut. Verkkotunnuksen voi luoda käyttäjäaltaan ”App Integration” kohdasta kuvan 10 mukaisesti.



KUVA 10. Käyttäjäaltaan verkkotunnuksen valinta

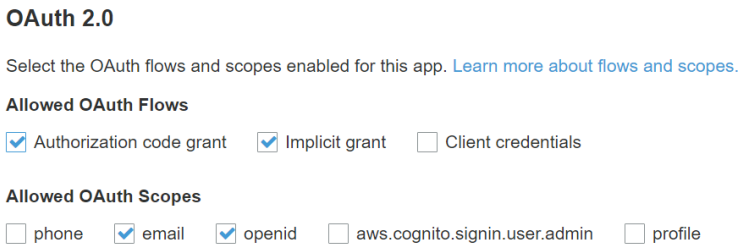
Seuraavaksi tulee käyttäjäaltaalle kertoa URL:it, joihin Cognito lähettää käyttäjän sisään- ja uloskirjautumisen jälkeen. Tämä tieto lähetetään URL-parametrina sisäänkirjautumissivulle samalla kun käyttäjä lähetetään sinne sovelluksesta. Käyttäjän mukana tuleva URL tulee täsmätä täysin käyttäjäaltaaseen lisättyä URL-osoitetta tai kirjautumispalvelut eivät toimi. Demosovellus ajetaan paikallisesti tässä tapauksessa, mutta Cognito tukee paikallista kehitystä ja sallii paikalliset URL:it kehityksessä. Kuvassa 11 on sallittujen sovellusten konfigurointisivu, josta näkyy localhost-pohjainen URL, jota demossa käytetään.



KUVA 11. Käyttäjäaltaan URL:it

6.4 OAuth

Kuvasta 12 näkyy OAuth virtauksien ja soveltamisalojen valinta. OAuth on it-alan standardin mukainen valtuutusprotokolla. Sen tarkoitus on antaa sovelluksille rajoitettu pääsy (soveltamisala) käyttäjän dataan paljastamatta käyttäjän salasanaa. Raible selittää OAuth-valtuutuksen vertaamalla sitä hotellihuoneavaimeen; avainkortilla pääsee huoneeseen, mutta kortin saadakseen, tulee vastaanotossa autentikoida itsensä. Autentikoinnin jälkeen ja kortin saatuaan voi päästä huoneeseen autentikoimatta itseään uudestaan joka kerta huoneeseen tultaessa. (Raible, 2007.)

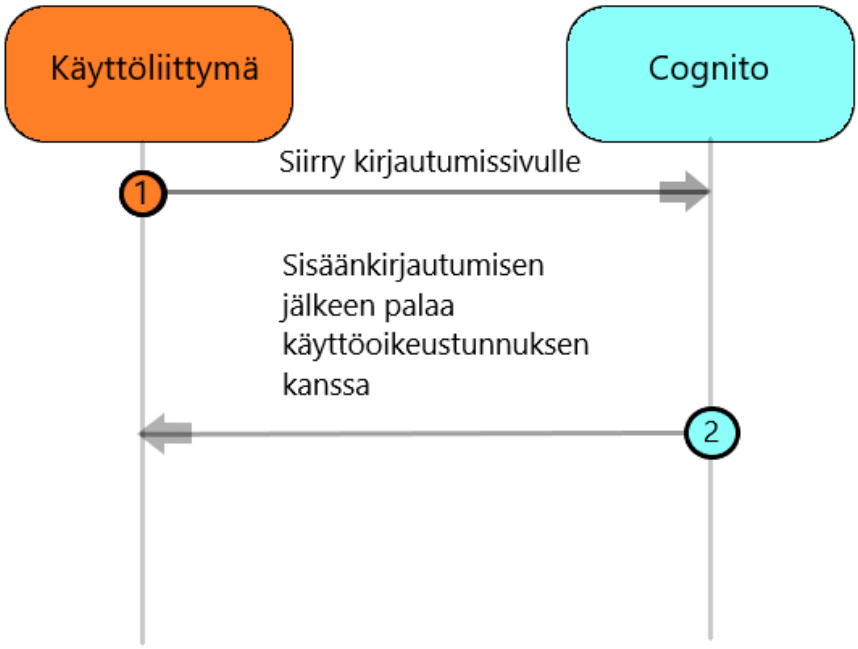


KUVA 12. OAuth-valinnat

OAuth soveltamisalat (scopes) määrittelevät asiat, joita applikaatio saa tehdä käyttäjän luvalla. Näihin voivat kuulua mm. julkaisujen tekeminen käyttäjän puolesta, käyttäjän aikaisempien julkaisujen lukeminen tai käyttäjän sisäänkirjautuminen ilman salasanan käyttöä. Soveltamisalat hyväksytetään käyttäjällä ja tämä voi itse valita sallimansa soveltamisalat. Cogniton tapauksessa esimerkiksi email-soveltamisala sallii sovelluksen nähdä käyttäjän sähköpostin ja onko tämän sähköposti varmistettu. Tästä on hyötyä siten, että käyttäjän ei tarvitse syöttää salasanaansa kirjautuessa ensimmäisen kerran jälkeen.

OAuth virtaukset (flows) määrittelevät protokollan, jonka mukaan autentikointi suoritetaan sovellukseen palatessa käyttäjäaltaan kirjautumissivulta. Implisiittinen virtaus (implicit flow) on nimensä mukaisesti implisiittinen, sillä sen virtauksessa kaikki kommunikaatio autentikoinnissa suoritetaan selaimessa. Aiemmin Express-taustaohjelman kanssa käyttöliittymä lähettää pyynnön taustaohjelmalle

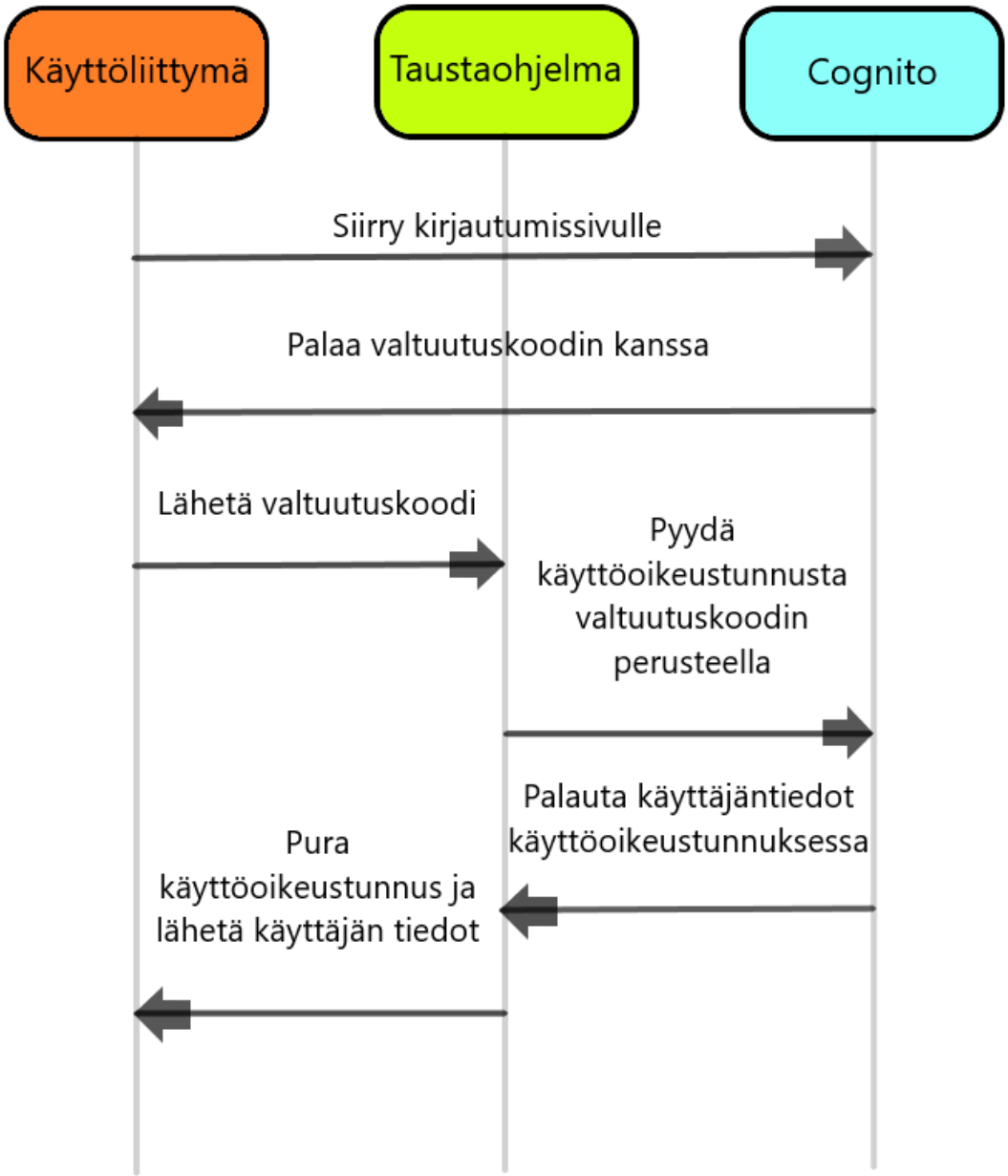
ja taustaohjelma vastaa pyyntöön joko käyttäjän tiedoilla ja tunnuksella tai virheviestillä. Implisiittisessä virtauksessa ei tarvita taustaohjelmaa. Autentikointi tapahtuu pelkästään selaimessa. Kuvan 13 mukaisesti sovelluksesta navigoidaan Cogniton sisäänkirjautumissivulle ja onnistuneen sisäänkirjautumisen jälkeen Cognito navigoi käyttäjän takaisin käyttöliittymään käyttöoikeustunnuksen kanssa. Käyttöoikeustunnus on jwt-tunnus, jonka purkamalla käyttöliittymä saa käyttäjän tiedot. Implisiittinen virtaus on paras sovelluksille, joilla ei ole taustaohjelmaa tukemassa niitä. Taustaohjelman puuttuminen nopeuttaa ja yksinkertaistaa autentikointia mutta kuten luvussa 5 mainittiin, selaimessa oleva koodi on haavoittuvainen tietoturvallisuus-uhille.



KUVA 13. Implisiittinen virtaus

Valtuutuskoodien myöntämisvirtaus (Authorization code grant) on standardisoitu luotettavimpana virtauksena, sillä sen toiminta perustuu käyttöliittymän ja taustaohjelman käyttöön. Tässä virtauksessa käyttöliittymä saa Cognitosta valtuutuskoodin ja lähettää sen taustaohjelmalle. Taustaohjelma vastaanottaa valtuutuskoodin ja vaihtaa sen käyttöoikeustunnukseen ja palauttaa käyttöliittymälle käyttäjän tiedot onnistuneen kirjautumisen merkiksi. Kuva 14 havainnollistaa tätä virtausta. Tätä virtausta suositellaan sovelluksille, joilla on taustaohjelma, joka voi

käyttää tunnus päätepistettä, sillä se on tietoturvallisempi ratkaisu kuin implisiittinen virtaus, koska käyttöoikeustunnus käsitellään taustaohjelman välityksellä.



KUVA 14. Valtuutuskoodin myöntämisvirtaus

Käyttäjäaltaan OAuth-virtausten valinnassa voi valita kummatkin edellä mainituista virtauksista ja kehittää sovelluksen käyttämään kumpaa vain. NewbieMakerissa käytetään valtuutuskoodien myöntämisvirtausta ja tämän takia myös raportin demossa käytetään kyseistä virtausta.

7 COGNITON INTEGRAATIO REACT WEB-SOVELLUKSEEN

7.1 Lähtökohdat integraatioon

Integraatio ohjelmistotuotantoympäristössä kuulostaa monimutkaiselta, mutta se tarkoittaa vain kahden tai useamman asian yhteen liittämistä yhdeksi kokonaisuudeksi. Luvussa 5 luotiin React-sovellus ja sille express-pohjainen taustaohjelma. Luvussa 6 esiteltiin Cognito, sen toiminta ja kuinka se otetaan käyttöön. Tässä luvussa esitellään NewbieMakerin Cognito-toteutus, joka perustuu edellisissä luvuissa käsiteltyihin konsepteihin. NewbieMaker on kuitenkin ”oikean maailman” sovellus ja paljon laajempi kokonaisuutena kuin luvun 5 demosovellus.

Tässä luvussa esiintyvä koodi on EUPL-lisenssin mukaisesti Cybercom Finland Oy:n omaisuutta.

7.2 OAuth-virtauksen valinta

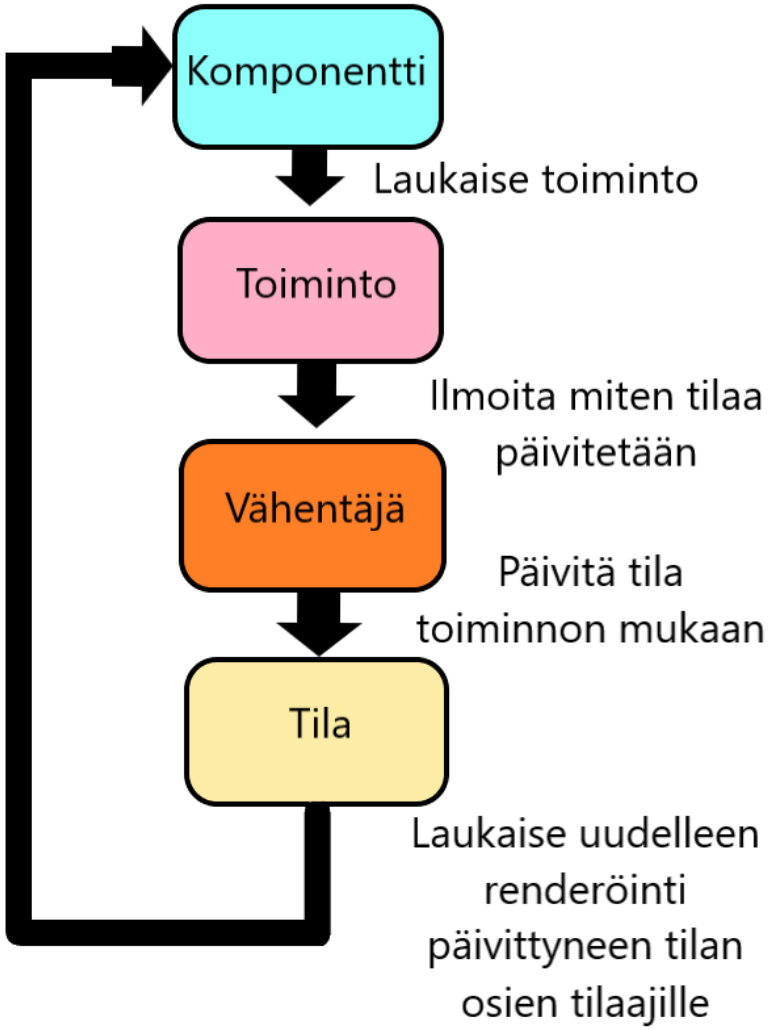
Kuten luvussa 6 mainittiin, Cognito toimii muun muassa implisiittisellä virtauksella sekä valtuutuskoodien myöntämisvirtauksella. Implisiittisessä virtauksessa Cogniton kirjautumissivu palauttaa käyttäjän salatun tunnusmerkin suoraan käyttöliittymälle selaimessa. Valtuutuskoodien myöntämisvirtauksessa taas kirjautumissivulta palataan käyttöliittymään valtuutuskoodin kanssa, joka lähetetään sovelluksen taustaohjelmalle, joka noutaa valtuutuskoodin avulla tunnistuspisteeltä käyttäjän tunnusmerkin ja palauttaa jwt-tunnuksen ja käyttäjän tiedot käyttöliittymälle. NewbieMakerissa hyödynnetään jälkimmäistä virtausta, sillä se on tietoturvallisempi ja NewbieMakerin tapauksessa sovelluksella on oma tietokanta, josta käyttäjän tiedot haetaan. Tämän takia autentikointi taustaohjelmassa menee luonnollisena osana kirjautumisen tapahtumaketjua.

7.3 Redux tilanhallinnan perustana

Sisäänkirjautuminen NewbieMakerissa perustuu samoihin elementteihin kuin demosovelluksen toteutus. Kummassakin on React-käyttöliittymä ja express-taustaohjelma. Vaikka NewbieMakerin toteutus on suuremmassa skaalassa, samat perusperiaatteet kuin demosovelluksessa pätevät. Kirjautumisessa siirrytään

Cogniton kirjautumissivulle, palataan valtuutuskoodin kanssa, haetaan taustaohjelmalla käyttöoikeustunnus ja saadaan käyttäjän tiedot käyttöliittymään, jonka jälkeen käyttäjä voi alkaa käyttämään sovellusta. NewbieMakerissa sovelluksen tilanhallinnan pohjana on redux-kirjasto.

Redux on tilanhallintakirjasto, jonka toiminta perustuu flux-arkkitehtuuriin. Kuvan 15 mukaisesti flux-arkkitehtuurissa komponentti laukaisee toiminnon (action), jota kuuntelee vähentäjä (reducer), joka toiminnon saadessaan päivittää sovelluksen tilaa (store) toiminnon mukaisesti. Komponentti ”tilaa” osan tilasta ja näyttää siitä riippuen erilaista sisältöä. Tilan päivittyessä komponentti renderöi itsensä uudestaan, jos sen tilaama osa tilaa päivittyy.



KUVA 15. Flux-arkkitehtuuri

Tilan päivitys toimii siis toimintojen ja vähentäjien kautta. Tässä ketjussa ei saa olla asynkronisia toimintoja. Asynkroninen ohjelmointimaailmassa tarkoittaa toimintaa, jonka kestoa ei voi tietää etukäteen. Redux ei salli asynkronisia vähentäjiä koska reduxin periaate on luoda ennalta-arvattavaa tilanhallintaa. Sisäänkirjautumisessa kuitenkin tarvitaan asynkronisia toimintoja kuten valtuutuskoodin lähettäminen taustaohjelmalle ja käyttäjän tietojen vastaanottaminen. Tämän toiminnon kestoa ei voi tietää ennalta ja siksi sitä ei voi suorittaa puhtaalla vähentäjällä reduxissa.

Reduxin toimintaan kuuluu väliohjelmistojen (middleware) käyttäminen asynkronisten toimintojen ja sivuvaikutusten suorittamiseksi. Väliohjelmistot reduxissa ovat lisäosia koodiin, jotka muodostuvat tilan lähettäjäfunktion (dispatch) ympärille. Tilan lähettäjä on metodi, jonka kutsuminen on ainoa tapa päivittää tilaa. Lähettäjä ottaa argumenttina toiminnon, jonka mukaan tilaa päivitetään. Lähettäjää voi ajatella toiminnon ”laukaisijana”. Väliohjelmisto tarjoaa väliosan toiminnon lähettämisen ja sen vähentäjään saapumisen välille, jossa voi suorittaa sivuvaikutuksia. Tällä tavoin asynkroniset toiminnot väliohjelmistossa tapahtuvat ennen tilan päivityksen kutsumista.

NewbieMakerin tapauksessa väliohjelmistona on redux-saga. Redux-saga on kirjasto, jonka tarkoituksena on tehdä sivuvaikutusten j0061 asynkronisten toimintojen hallitseminen helpoksi ja tehokkaaksi. Redux-sagan toiminta perustuu JavaScriptin generaattorifunktioihin. Generaattorit ovat funktioita, jotka eivät toteuta funktiokehoaan heti niitä kutsuessa. Generaattorit toimivat hieman samalla protokollalla kuin asynkroniset funktiot JavaScriptissä siinä mielessä, että kummassakin funktiossa funktion suorituksen voi pysäyttää jonkin asian odottamiseksi. Kuvasta 16 näkee generaattorien ja asynkronisten funktioiden yksinkertaisen syntaksin. Asynkronisten funktioiden ja generaattoreiden syntaksi varsinkin vastausten suhteen on yksinkertaista ja ymmärrettävää.

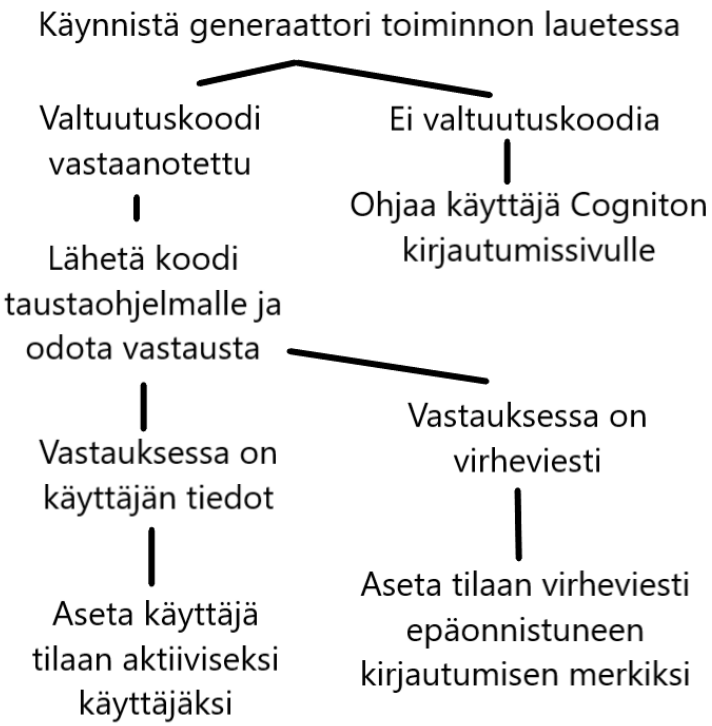
```
async function asyncLogin() {
  const result = await checkAndRedirect();
  return result;
}

function* generatorLogin() {
  const result = yield(checkAndRedirect());
  return result;
}
```

KUVA 16. Asynkroninen funktio ja generaattorifunktio

Kuvasta 17 näkee NewbieMakerin sisäänkirjautumissaagan, joka vastaanottaa toiminnon kuorman (joka tässä tapauksessa on Cogniton valtuutuskoodi) parametrina. Jos kuormaa ei ole, ohjataan käyttäjä Cogniton kirjautumissivulle. Jos kuorma löytyy, lähetetään koodi taustaohjelmalle ja odotetaan käyttäjän tietoja. Jos tiedot löytyvät, asetetaan käyttäjä tilaan aktiiviseksi käyttäjäksi. Jos tietoja ei löydy, asetetaan kirjautumisvirhe tilaan ja ilmoitetaan käyttäjälle virheestä. Kuvasta 18 löytyy kirjautumissaagan protokolla teoreettisesta näkökulmasta.

```
function* loginSaga(action: PayloadAction<string>) {
  // action.payload is the cognito auth code

  yield put(mainActions.setLoading(true));
  try {
    if (action.payload.length === 0) {
      yield redirectToCognito();
    }
    const response = yield call(login, action.payload);
    const user: Api.User = yield call(getOne, response.userId);
    yield put(mainActions.setActiveUser(user));
  } catch (error) {
    const errorMessage = "Error logging in";
    console.error(error);
    yield put(mainActions.loginFailure(errorMessage));
  } finally {
    yield put(mainActions.setLoading(false));
  }
}
```

KUVA 17. NewbieMakerin kirjautumissaaga



KUVA 18. NewbieMakerin kirjautumissaagan protokolla

Redux on tunnetusti hankala ymmärtää aluksi, mutta se on koherentti ja luotettava ratkaisu tilanhallintaan web-sovelluksissa. NewbieMakerin redux-toteutukseen sisältyy paljon muitakin osia ja toiminnallisuutta mutta kirjautumissaaga on tärkein ymmärtää sisäänkirjautumisen kannalta.

7.4 NewbieMakerin sisäänkirjautuminen käyttöliittymässä

NewbieMakerin sisäänkirjautuminen alkaa sisäänkirjautumiskomponentista. Käyttäjän navigoidessa sovellukseen, hänet ohjataan tähän komponenttiin. Ennen kuin komponentti palauttaa sisältönsä, se tarkastaa onko URL:issa parametria ”code”. Koska tälle sivulle palataan Cogniton kirjautumissivulta, koodin läsnäolo URL:issa tarkoittaa, että Cognitosta on autentikoitu onnistuneesti, kun taas koodin poissaolo indikoi, että komponenttiin saavuttiin jollain muulla keinolla kuin onnistuneella autentikoinnilla Cognitosta. Koodin tarkastuksen jälkeen laukaistaan sisäänkirjautumistoiminto, joka aloittaa edellä mainitun sisäänkirjautumissaagan. Kuvassa 19 sisäänkirjautumiskomponentin koodi.

```
const Login: React.FC<RouteComponentProps> = ({ location }: RouteComponentProps) => {
  const classes = useStyles();
  const dispatch = useDispatch();
  const errorMessage = useSelector((state: stateType) => state.main.errorMessage);

  const code = new URLSearchParams(location.search).get("code");

  useEffect(() => {
    dispatch(mainActions.login(!code ? "" : code));
  }, [code, dispatch]);

  return (
    <>
      {isAuthenticated() ? (
        <Redirect to={isAdmin() ? "/admin/users" : isSupervisor() ? "/supervisor" : "/"} />
      ) : errorMessage.length !== 0 ? (
        <Redirect to="/login-error" />
      ) : (
        <Grid container className={classes.root}>
          <Grid item xs={12}>
            <Paper className={classes.paperContainer}>
              <Loader loading={true} />
            </Paper>
          </Grid>
        </Grid>
      )}
    </>
  );
};
```

KUVA 19. Sisäänkirjautumiskomponentti

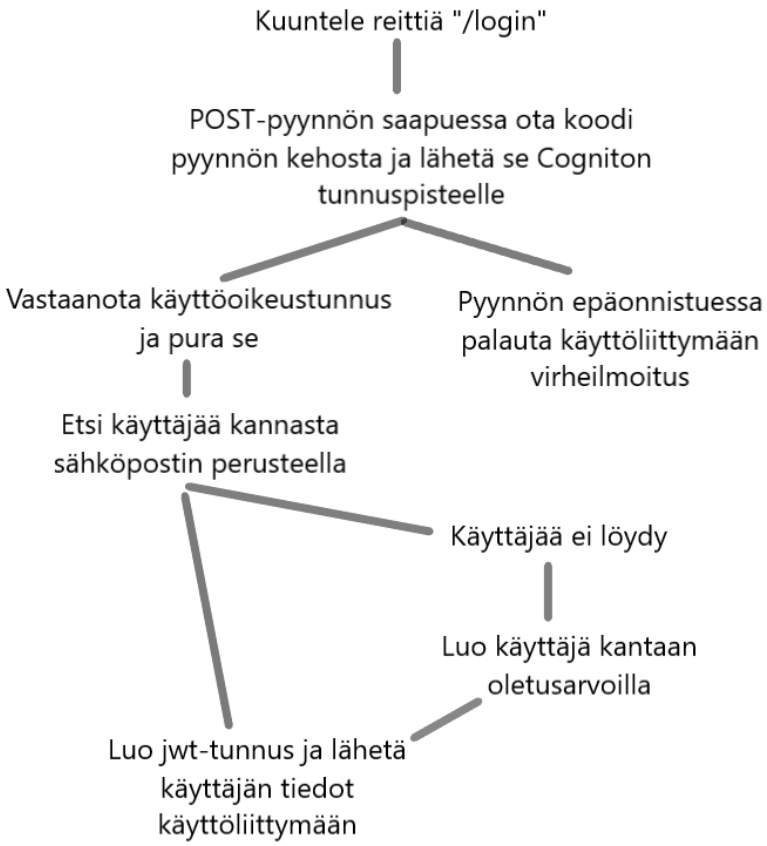
Sisäänkirjautumiskomponentin sisältö on yksinkertaisesti vain latausilmaisin eli pyörivä rulla. Tämä johtuu siitä, että sisäänkirjautumiskomponentti on vain sovelluksen ja Cogniton välikappale, josta joko navigoidaan Cognitoon tai jossa odotetaan sisäänkirjautumissaagan päättymistä. Kirjautumissaagan päätyttyä sovellus ohjaa käyttäjän joko sovelluksen etusivulle tai komponenttiin, jossa ilmoitetaan käyttäjälle kirjautumisen epäonnistuneen. Jälkimmäisen ei tulisi tapahtua, ellei sovelluksen taustaohjelmassa ole ongelmia.

Sisäänkirjautumiskomponentista näkee, että virheilmoituksen ollessa olemassa, käyttäjä ohjataan epäonnistuneen kirjautumisen mukaiselle sivulle. Tästä huolimatta aiemmin raportissa mainittiin, että onnistuneen kirjautumisen jälkeen käyttäjä ohjataan sovelluksen etusivulle. Mikä sovelluksessa ohjaa käyttäjän etusivulle sitten? Sovelluksen juuressa App.tsx:ssä tilataan redux-tilasta aktiivinen käyttäjä ja sen ollessa olemassa käyttäjä ohjataan etusivulle. Edellisen luvun mukaan komponentit, jotka ovat tilanneet sovelluksen tilan päivittyneen osan renderöidään uudestaan vastaamaan uutta dataa tilassa.

7.5 NewbieMakerin sisäänkirjautuminen taustaohjelmassa

NewbieMakerin taustaohjelma on express-sovellus. Taustaohjelman tehtävä on olla välikappale käyttöliittymän ja tietokannan välillä eli tulkita käyttöliittymän pyynnöt ja vastata niihin protokollan mukaisesti. Kuten edellisissä luvuissa on mainittu, taustaohjelman koodi ei näy selaimessa ja siksi siellä on hyvä pitää koodi, jossa kommunikoidaan tietokannan kanssa. NewbieMakerin tietokanta on MongoDB-kanta. MongoDB on dokumenttipohjainen skaalautuva tietokantapalvelu. Dokumenttipohjainen tarkoittaa, että data tallennetaan kantaan JSON-muodossa. Tämä tekee kyselyiden tekemisen kantaan todella tehokkaaksi JavaScriptissä.

Kuten luvun 6 kuvassa 14 ilmenee, taustaohjelman rooli valtuutuskoodi virtauksessa on vastaanottaa valtuutuskoodi, lähettää se Cogniton tunnuspisteelle, vastaanottaa käyttöoikeustunnus, purkaa se ja lähettää käyttäjän tiedot käyttöliittymälle. NewbieMakerin tapauksessa taustaohjelman rooli on hieman isompi, sillä NewbieMakerissa on oma tietokanta, jossa käyttäjästä on tarkempaa tietoa. Eli käyttöoikeustunnuksen purkamisen jälkeen ei vielä vastata käyttöliittymän pyyntöön. Tämän sijaan etsitään sovelluksen tietokannasta käyttäjää sähköpostin perusteella (Cognitossa valittiin sähköposti identifioivaksi tekijäksi). Jos käyttäjä löytyy tietokannasta, luodaan jwt-tunnus ja palautetaan se ja tarpeelliset tiedot käyttäjästä käyttöliittymään. Jos sovelluksen tietokannasta ei löydy käyttäjää Cognitosta tulleen sähköpostiosoitteen perusteella, käyttäjä lisätään kantaan oletusarvoilla ja sen jälkeen luodaan jwt-tunnus ja palautetaan käyttäjän tiedot käyttöliittymälle. Kuvassa 20 on taustaohjelman protokolla visualisoituna.



KUVA 20. NewbieMakerin taustaohjelman sisäänkirjautumisen protokolla

Virheviestejä kirjautumisessa ei tulisi tapahtua taustaohjelmassa, saati käyttöliittymässä sillä käyttäjän syöte otetaan Cognitossa vastaan ja sovelluksen tehtävä on vain käsitellä Cogniton antamat tiedot. Tämä helpottaa koodin testausta ja ylläpidettävyyttä. Jos virheviestejä tulee, ne tulevat joko koodin virheellisestä päivityksestä, sovelluksen palvelimen virheestä tai Cogniton virheestä. Mikään edellä mainituista ei ole normaalisti odotettavissa mutta on silti hyväksi rakentaa koodi olemaan valmis niiden tapahtumiselle.

8 POHDINTA

Työn lähtökohtana oli selvittää eroja käsin tehdyn ja kolmannen osapuolen tarjoaman käyttäjähallinnan välillä, pohtia käyttäjähallinnan merkityksellisyyttä ja tarkastella NewbieMakerin Cognitolla toteutettua käyttäjähallintaa. Työssä tultiin siihen tulokseen, että käsin tehty ratkaisu käyttäjähallinnassa on varteen otettava ratkaisu, jos kehitystiimillä on osaamista taustaohjelmaohjelmoinnista ja toteutuksen sovelluksessa on mahdollisuus käyttää taustaohjelmaa. Kuitenkin jos kehityksessä on resurssien suhteen mahdollista käyttää kolmannen osapuolen työkalua käyttäjähallintaan, se olisi suositeltavaa.

Kolmannen osapuolen käyttäjähallintatyökalut ovat tällä hetkellä luotettavia käyttää sovelluksissa. Niiden integroiminen sovellukseen sujuu helposti linkittämällä sovellus ja työkalu yhteen, useimmiten lisäämällä tunnistusmuuttujat kumpaankin. Monet käyttäjähallintatyökalut tukevat erilaisia käyttötarkoituksia, esimerkiksi Cognitoa on mahdollista käyttää pelkästään autentikointiin implisiittisellä virtauksella tai koko sovelluksen käyttäjätietojen pohjana valtuutuskoodien myöntämisvirtauksella. Tietoturvastandardit nousevat sitä mukaa mitä vaativampia käyttötarkoituksia käyttäjähallintatyökaluille tulee. Jotta isot yritykset ja julkinen sektori voivat käyttää näitä työkaluja sovelluksissaan, tietoturvan on pakko olla kehittyneellä tasolla tai palvelun asiakas mieluummin vaihtaa kilpailevaan tuotteeseen.

Työn tuotos oli NewbieMakerin Cognitolla toimiva käyttäjähallinta. Ratkaisu päätyi muiden kehittäjien katselmoinnin jälkeen tuotantoon. Toteutus on koherentti, tehokas, tietoturvallinen ja siinä käytetään pilvipalvelua (Cognito) eli toteutukselle asetetut kriteerit täytettiin. Ratkaisua vietiin eteenpäin implementoimalla mahdollisuudet hallinnoida käyttäjiä ylläpitäjille ja sovelluksen tietokanta siirrettiin pilvipalveluihin. Työ onnistui tavoitteessaan aikataulun mukaisesti ja oli erittäin hyvä oppimiskokemus Cogniton ja muiden kolmannen osapuolen käyttäjähallinnan työkaluihin.

Käyttäjähallinnan tietoturvallisuus oli toinen johdannossa esiin nostettu seikka. Erittäin määrittävä tekijä tietoturvallisuudessa on taustaohjelman käyttö. Käyttäjätietojen noutaminen Cognitosta suoraan selaimessa ei ole tietoturvallisin ratkaisu ja sitä ei tulisi käyttää muuhun käyttöön kuin valtuutukseen, eli käyttäjän

sovellukseen päästämiseen. Taustaohjelman käyttö avaa mahdollisuuden käyttää valtuutuskoodien myöntämisvirtauksen käytön, joka on tietoturvallisempi ratkaisu kuin implisiittinen virtaus. Muita tietoturvallisuutta edistäviä tekijöitä ovat esimerkiksi koodin katselmointi ennen sen julkaisua, tietoturvakoulutus kehittäjille ympäristömuuttujien käyttö ja huolellinen testaus.

Vaikka tällä hetkellä käyttäjähallinta on hyvällä mallilla maailmassa, on silti huomion arvoista pitää katse horisontissa ja pohtia mitä tulevan pitää. Tällä hetkellä huolestuttava trendi on sovelluksien ”näkymättömät valtuutukset”, eli asiat joihin käyttäjä antaa sovellukselle luvan tietämättään pelkästään käyttämällä sovellusta. Google, Apple ja Facebook ovat usein tarkastelun alla hallituksilta liittyen siihen mitä heidän sovelluksensa tekevät käyttäjien datalla. Jokainen kolmesta edellä mainitusta yrityksestä ovat jääneet kiinni toimiensa salailusta liittyen käyttäjien dataan. Huolestuttavimpia tapauksia ovat ne, joissa yritykset ovat vakuuttaneet käyttäjälle esimerkiksi ääniviestien olevan täysin suojattuja ja salattuja ja myöhemmin paljastuukin, että yritykset ovat palkanneet ihmisiä kuuntelemaan käyttäjien ääniviestejä. The Guardianin artikkelin mukaan Applen älykellot tekevät vahinkolaukauksia huolestuttavan usein ja tallentavat jopa 30 sekuntia käyttäjän puhetta (Hern, 2019). Sosiaalisessa mediassa vitsaillaan Googlen olevan Terminator-elokuvissa nähtävä Skynet, elävä AI-järjestelmä, jonka tarkoituksena on hävittää ihmiset maapallolta. Jotta maailmanloppuskenaariot voidaan välttää ja käyttäjien yksityisyys säilyttää, tulee lainsäätäjien kiinnittää asiaan huomiota kansainvälisesti, asettaa rajoituksia ja vaatia yrityksiä ottamaan vastuuta tekemisistään ja miettimään suuntaa, johon he johtavat maailmaa.

Mediassa ajankohtainen huoli vuonna 2020 on Suomalaisen Vastaamo-palvelun tietomurto. Vastaamo on suomalainen psykoterapiakeskus, joka erikoistuu verkkovastaanottoihin. Yrityksen potilastiedot hakkeroitiin vuoden 2020 loppupuolella ja uhattiin vuotaa julkisuuteen. Helsingin sanomien artikkelissa mainittiin, että hyökkäyksen tekijän mukaan potilastietoihin oli päästy käsiksi oletuskäyttäjätunnuksella ja sanasanalla (Halminen, 2020). Tapaus herättää lainsäädäntöön liittyviä kysymyksiä: millaisia tietoturvavaatimuksia yrityksille ja varsinkin potilastietoja käsitteleville yrityksille tulee asettaa, jotta tiedot ovat turvassa ja salattuina, sekä miten katselmoidaan yritysten tietoturvaa säännöllisesti ja lisäksi nostetaan tie-

toisuuttaa huonon tietoturvan riskeistä. Vastaamon tapauksen kaltaiset varoittavat esimerkit toivottavasti nostavat keskustelua päättäjien kesken tietoturvallisuudesta.

AWS Cognito ei suinkaan ole ainoa mahdollinen työkalu käyttäjähallinnan ulkoistamiseksi. Autentikointipilvipalveluita nousee lisää koko ajan tekniikoiden kehittyessä ja tälläkin hetkellä Cognitolla on monia kilpailijoita esimerkiksi Microsoftilta, Googlelta ja monilta muilta isoilta ja pieniltä it-yrityksiltä. Käyttäjähallintapalveluiden tulevaisuus tulee varmasti sisältämään kovaa kilpailua yritysten välillä ja tämän tulisi hyödyttää palveluiden käyttäjiä eli sovellusten kehittäjiä.

LÄHTEET

Dasgupta, D., Roy, A., Nag, A. 2017. Advances in User Authentication. Cham: Springer International Publishing.

Krug, S. 2013. Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability. 3. painos. San Francisco, CA: New Riders.

Amoroso, E. 2007. Cyber Security. Summit, NJ: Silicon Press.

Raible, M. What the Heck is OAuth? Julkaistu 21.6.2017. Luettu 27.10.2020 https://developer.okta.com/blog/2017/06/21/what-the-heck-is-oauth#:~:text=OAuth%20is%20a%20delegated%20authorization,cases%20addressing%20different%20device%20capabilities.

Hern, A. Apple contractors 'regularly hear confidential details' on Siri recordings. The Guardian. Julkaistu 26.7.2019. Luettu 11.11.2020. https://www.theguardian.com/technology/2019/jul/26/apple-contractors-regularly-hear-confidential-details-on-siri-recordings

Halminen, L. Vastaamon potilasrekisteri on ollut erittäin helposti saatavilla, arvioivat HS:n haastattelemat asiantuntijat. Helsingin sanomat. Julkaistu 28.10. Luettu 11.11.2020. https://www.hs.fi/kotimaa/art-2000006702821.html