Ajanvarausjärjestelmä MERN-pinoa hyödyntäen

Henna Paananen

Haaga-Helia ammattikorkeakoulu

Tradenomin tutkinto

Opinnäytetyö

2023

Tiivistelmä

Tekijä(t)
Henna Paananen

Tutkinto
Tradenomi

Raportin/Opinnäytetyön nimi
Ajanvarausjärjestelmä MERN-pinoa hyödyntäen

Sivu- ja liitesivumäärä
47 + 2

Web-sovelluskehitykseen on tarjolla monia erilaisia teknologioita ja työkaluja, joita voidaan hyödyntää ohjelmistoprojektien kehityksessä. Näitä ovat muun muassa erilaiset ohjelmointikielet, sovelluskehykset ja tietokannat, joista voidaan muodostaa erilaisia teknologiapinoja.

Tämä opinnäytetyö keskittyy tutkimaan MERN-teknologiapinoa ja sen käyttömahdollisuuksia web-sovelluskehityksessä. Työssä käsitellään sen neljää avoimen lähdekoodin teknologiaa: MongoDB-tietokantaa, Express.js-sovelluskehystä, React-käyttöliittymäkirjastoa ja Node.js-ajoympäristöä. Opinnäytetyössä toteutetaan prototyyppisovellus käyttäen näitä teknologioita. Lisäksi käydään läpi sovelluskehityksen eri vaiheita, jotka ovat tässä projektissa määrittely, suunnittelu, toteutus, testaus ja julkaisu. Tässä opinnäytetyössä keskitytään erityisesti toteutusvaiheeseen, mutta myös muihin vaiheisiin tehdään tarkastelua projektin laajuuden mukaisesti.

MERN-pino on kokonaisuudessaan JavaScript-pohjainen teknologiapino. Sen tietokanta poikkeaa perinteisistä SQL-tietokannoista, sillä se on dokumenttiorientoitunut. Käyttöliittymäpuoli on kirjastopohjainen ja sen avulla on mahdollista rakentaa käyttöliittymä pienistä, itsenäisistä komponenteista. Taustapalveluna toimii Node.js ja sen päällä toimiva Express.js, joka hoitaa muun muassa HTTP-pyyntöjen käsittelyn ja virheiden hallinnan. Node.js mahdollistaa asynkronisten pyyntöjen käsittelyn, jolloin koodin ei tarvitse odottaa vastausta, vaan se voi jatkaa suoritustaan samanaikaisesti.

Opinnäytetyö toteutetaan sovelluskehityksen vaiheiden mukaisesti ja määrittelyssä esitetyt vaatimukset otetaan huomioon eri vaiheissa esimerkiksi koodiesimerkkien, testitulosten ja käyttöliittymäkuvien avulla. Toteutusvaiheessa käytiin läpi, kuinka pinon eri teknologioita ja niiden lisäosia voidaan hyödyntää ajanvarausjärjestelmää kehitettäessä.

Tuotoksena syntyi vaatimukset täyttävä, suunnittelun mukainen, toimiva ja testattu prototyyppisovellus, joka julkaistiin Oraclen pilvipalveluun. Opinnäytetyön eri vaiheissa syvennettiin ymmärrystä teknologiapinosta sekä kehitettiin JavaScript-osaamista.

Asiasanat
MERN, MongoDB, Express.js, React, Node.js, JavaScript, Full Stack

Sisällys

1 Johdanto

Moderni web-kehitys on muuttunut nopeasti viime vuosina ja erilaiset teknologiat ja kirjastot ovat kehittyneet tukemaan sovellusten suorituskykyä ja käytettävyyttä. Yksi teknologiapinoista on MERN (MongoDB, Express.js, React, Node.js), joka yhdistää useita teknologioita sekä mahdollistaa kehittäjille monimutkaisten ja laadukkaiden web-sovellusten luomisen nopeasti ja tehokkaasti.

Teknologiapinossa käytetty JavaScript on kymmenettä vuotta peräkkäin maailman yleisimmin käytetty ohjelmointikieli (Stack Overflow 2022), jota käytetään laajasti erilaisissa sovelluskehitysprojekteissa ja en osaaminen on arvokasta tulevaisuuden työmarkkinoilla. JavaScriptin vahva asema teknologiapinossa kuvastaa sen monipuolisuutta ja kykyä soveltua erilaisiin käyttötarkoituksiin.

Opinnäytetyössä luodaan MERN-teknologiapinoa hyödyntäen ajanvarausjärjestelmä kuvitteelliselle mielenterveyspalveluita tuottavalle Mielen Voima -yritykselle. Ajanvarausjärjestelmän keskeinen tehtävä on mahdollistaa asiakkaille aikojen varaaminen erilaisiin mielenterveyspalveluihin. Valitsin aiheen, koska aihe on ajankohtainen ja mielenterveysongelmat ovat yleisiä. Tarve hoidolle ja tuelle on suuri. Suomessa aikuisista 20–25 % kokee vuosittain mielenterveyden häiriöitä ja suomalaisten työkyvyttömyyseläkkeistä yli puolet on mielenterveysperusteisia. Silti vain puolet saa tarvitsemaansa hoitoa mielenterveyden häiriöihin. (MIELI ry 2023.)

Sovelluksen kehittäminen alusta loppuun on hyödyllistä, sillä sen avulla syntyy ymmärrystä sovelluksen kokonaisvaltaisesta kehitysprosessista. Se auttaa myös hahmottamaan paremmin, miten erilaiset teknologiat toimivat yhdessä. Sovelluskehitys etenee tyypillisesti useiden vaiheiden kautta ja tässä projektissa vaiheet on jaettu määrittelyyn, suunnitteluun, toteutukseen, testaukseen ja julkaisuun. Opinnäytetyön kehitysprosessi seuraa näitä vaiheita. Tärkeässä roolissa kehityksessä on myös JavaScriptin monipuolinen hyödyntäminen sovelluksen eri osa-alueilla.

1.1 Tavoitteet ja rajaus

Tämän opinnäytetyön tarkoituksena on tutkia MERN-teknologiapinoa (MongoDB, Express.js, React ja Node.js) sekä kehittää ajanvarausjärjestelmä näitä teknologioita hyödyntäen. Tavoitteena on syventää ymmärrystä käytetyistä teknologioista, tutkia pinon vahvuuksia ja heikkouksia, sekä kehittää valmiita ratkaisuja ajanvarausjärjestelmän toteuttamiseen. Pyrin myös selvittämään, mitä työkaluja tarvitaan pinon avulla kehittämiseen ja mitä toteutetun ajanvarausjärjestelmän käyttöönotto muissa laitteissa vaatii. Tutkimusprosessissa käydään läpi MERN-pinon eri osa-alueita, eli MongoDB-tietokantaa, Express.js-websovelluskehystä, React-käyttöliittymäkirjastoa ja Node.js-ajoympäristöä. Lisäksi toteutusvaiheessa käydään myös läpi projektin kannalta tarpeellisia lisäosia.

Lopullisena tavoitteena on tuottaa toimiva ja testattu ajanvarausjärjestelmä, joka hyödyntää MERN-teknologiapinoa ja sen ominaisuuksia.

Projektissa keskitytään pääasiassa sovelluksen toteutukseen, jättäen muut kehitysvaiheet pintapuolisemmiksi. Tarkastelu rajoittuu valittuihin tekniikoihin, sillä jokainen vaihe on oma laaja aihealueensa ja tarjoaa useita erilaisia tekniikoita hyödynnettäväksi. Määrittely rajataan käyttäjätarinoiden ja vaatimusten luomiseen. Käyttäjätarinoiden perusteella luodaan käyttötapauskaavio havainnollistamaan sovelluksen käyttötapauksia. Suunnitteluvaiheessa keskitytään arkkitehtuuri- ja käyttöliittymäsuunnitteluun. Arkkitehtuurisuunnittelussa tarkastellaan tulevaa sovellusrakennetta, ja se rajataan tietokantarakenteen ja yhden esimerkkipäätepisteen rajapintasuunnitelman läpikäymiseen. Ajanvarausjärjestelmän käyttöliittymän suunnittelu rajataan rautalankamallin ja prototyypin luomiseen käyttöliittymästä. Toteutuksen vaiheessa käydään läpi projektin teknistä toteutusta, ja se rajataan projektissa käytettyjen tekniikoiden ja toteutusta havainnollistavien koodiesimerkkien läpikäymiseen. Testauskin on laaja aihe, joten sen tarkastelu rajataan projektiin valittuihin testaustekniikoihin. Projektin julkaisua käsitellään lyhyesti, esittäen esimerkkinä projektin julkaisun Oracle-pilvipalveluun.

1.2 Sanasto ja lyhenteet

BSON	Binary JSON. Binäärinen tiedonvälitysformaatti, joka sisältää tietotyyppien ja pituuksien koodauksen binäärimuodossa.
Dokumenttitietokanta	Tiedot tallennetaan kantaan dokumentteina sen sijaan, että ne tallennettaisiin riveinä taulukossa.
DOM	Document Object Model. Dokumentin rakenne kuvataan DOM-puuna, joka on jaoteltu eri osiin.
Express.js	Suosittu taustapalvelun puolen sovelluskehys. Käytetään REST-rajapintojen luomiseen Node.js-ajoympäristön kanssa.
Full Stack	Ohjelmistokehitystä laajasti sekä käyttöliittymän – että taustapalvelun puolella.
HTTP	HyperText Transfer Protocol. Protokolla, jota käytetään tietojen siirtämiseen verkkopalvelimen ja selaimen välillä.
JavaScript	Kevyt, tulkattu ohjelmointikieli, jota voi käyttää sekä käyttöliittymän että taustapalvelun kehityksessä (GeeksForGeeks 18.3.2023).

JSON	JavaScript Object Notation. Tiedonvälitysformaatti, joka perustuu avain-arvo-pareihin.
JWT	JSON Web Token. Token, joka koostuu kolmesta eri osasta: otsikko (header), sisältö (payload) ja allekirjoitus (signature) (Jwt s.a).
Käyttöliittymä	Sovelluksen osa, joka sisältää visuaaliset elementit, kuten sivut, napit, kuvakkeet jne. Käyttäjä näkee käyttöliittymän ja voi tämän avulla olla vuorovaikutuksessa sovelluksen kanssa.
MERN-pino	Teknologiapino, jossa käytetään MongoDB-, Express.js-, React- ja Node.js-teknologioita.
MongoDB	Tietokantaohjelmisto, joka säilyttää tietoa JSON-kielen kaltaisissa dokumenteissa (MongoDB s.a).
MVC	Käyttöliittymäsuunnittelussa käytetty arkkitehtuuri, jossa ohjelma jaetaan malliin (model), näkymään (view) ja käsittelijään (controller).
Node.js	Ajoympäristö, joka suorittaa JavaScriptin ajamisen palvelinpuolella.
NoSQL	Tietokanta, joka tallentaa dataa muussa muodossa kuin relaatiotaulukoissa.
React	Käyttöliittymien rakentamista varten luotu JavaScript-kirjasto (React s.a).
Taustapalvelu	Sovelluksen osa, joka käsittelee sovelluksen tietoa ja logiikkaa taustalla. Taustapalvelu ei ole suoraan näkyvissä käyttäjälle.
Teknologiapino	Yhdistelmä teknologioita, jotka on valittu sovelluksen toteuttamiseen.
Virtuaalinen DOM	Kevyt kopio oikeasta DOM-puusta. Sen avulla React päivittää vain tarvittavat muutokset oikeaan DOM-puuhun.

2 MERN-pino

MERN on JavaScript-pohjainen teknologiapino, joka mahdollistaa monipuolisten ja skaalautuvien web-sovellusten rakentamisen. Se koostuu neljästä eri avoimen lähdekoodin teknologiasta: MongoDB-tietokantaohjelmistosta, Express.js-sovelluskehyksestä, React-käyttöliittymäkirjastosta ja Node.js-ajoympäristöstä. Pino on suunniteltu tukemaan modernia kehitystä, kuten pilvipohjaista arkkitehtuuria ja käyttöliittymäpohjaista suunnittelua. Koska MERN-pinossa käytetään samaa ohjelmointikieltä sekä käyttöliittymän että taustapalvelun puolella, kehitystiimien jäsenet voivat käyttää samaa kieltä, mikä helpottaa kommunikointia ja yhteistyötä. Vuonna 2022 sovelluskehittäjille tehdyssä kyselyssä neljän eniten käytetyn web-sovelluskehyksen joukossa oli teknologiapinossa käytetyt Node.js, React ja Express.js (Statista 2022).

Muita vastaavia JavaScript-teknologiapinoja ovat MEVN ja MEAN. MERN-pinossa käytetyn React-kirjaston sijaan MEVN käyttää Vue.js-sovelluskehystä ja MEAN käyttää Angular.js-sovelluskehystä. Kaikki nämä käyttävät samoja taustapalvelun komponentteja, mutta eroavat toisistaan käyttöliittymän työkaluissa. Jokaisella teknologialla on omat vahvuutensa ja heikkoutensa, ja pinon valinta riippuu projektin luonteesta. React soveltuu hyvin moderneihin sovelluksiin, joissa painotus on käyttöliittymässä, Angular.js sopii laajoihin ja monimutkaisiin projekteihin, ja Vue.js on hyvä valinta korkeaa suorituskykyä ja helppoa käyttöliittymän kehitystä painottaviin sovelluksiin. (Stoyko 4.2.2022.) Esimerkiksi Angular.js-sovelluskehykseen verrattuna React-kirjaston väitetään suoriutuvan paremmin käyttöliittymän renderöinnissä ja suorituskyvyssä (Baiskar, Paulzagade, Koradia, Ingole & Shirbhate 2022).

PERN-teknologiapino puolestaan eroaa MERN-pinosta tietokantatyökaluissa, sillä PERN käyttää PostgreSQL-tietokantaa. Stack Overflow:n web-kehittäjäkyselyn mukaan se on toiseksi suosituin tietokanta heti MySQL-tietokannan jälkeen (Stack Overflow 2022). Se valitaan mieluummin tietokannaksi, koska siinä on tiukat säännöt tiedon eheydestä ja se noudattaa ACID-periaatetta. Se mahdollistaa monimutkaisempia kyselyitä ja skaalautuu helpommin kuin NoSQL-tietokannat. (Noble Desktop 13.1.2023.)

Muita suosittuja teknologiapinoja ovat esimerkiksi LAMP- ja Ruby on Rails-teknologiapinot. LAMP-teknologiapino pitää sisällään Linux-käyttöjärjestelmän, Apache-verkkopalvelimen, MySQL-tietokannan ja PHP-, Perl- tai Python-ohjelmointikielen. Ruby on Rails on Full Stack -kehys, joka pohjautuu Ruby-ohjelmointikieleen. Sen sijaan, että vertaisi teknologiapinoja toisiinsa, voi olla parempi keskittyä vertaamaan teknologiapinojen sisältämiä eri teknologioita niitä vastaaviin teknologioihin.

MERN-pinossa on myös haittapuolia, joista voi koitua haasteita sovelluskehityksessä. Se on riippuvainen Node Package Managerista, ja ongelmat tai muutokset näissä moduuleissa voivat vaikuttaa projektin toimintaan. Tehokkuuden puolesta käyttöliittymässä käytetty React vaatii enemmän koodia, sillä se on kirjasto eikä sovelluskehys, ja tämä voi vaikuttaa suorituskykyyn. Suuremman skaalan sovelluksissa MERN voi koitua hankalaksi hallita ja siinä voi ilmetä suorituskykyongelmia, jotka mahdollisesti estävät sovelluksen skaalautumisen. (Ramotion 20.12.2023.)

2.1 MongoDB

MongoDB on dokumenttitietokantaohjelmisto, joka eroaa perinteisistä SQL-pohjaisista tietokantajärjestelmistä siten, että se tallentaa tietoja dokumentteina, ei relaatiomalleina. Dokumentit ovat helppokäyttöisiä ja ne voivat sisältää sekä strukturoitua että ei-strukturoitua dataa samassa dokumentissa. Nämä dokumentit tallennetaan sitten kokoelmiin, joille voi suorittaa erilaisia tietokantatoimintoja, kuten lisääminen ja poistaminen. (Phaltankar, Ahsan, Harrison & Nedov 2020, luku 1.) Samalla periaatteella toimivia tietokantaohjelmistoja ovat muun muassa Couchbase, CouchDB, RavenDB ja Cassandra.

Dokumenttiperustainen lähestymistapa tarjoaa joustavuutta tiedon tallentamiseen, esimerkiksi silloin kun olemassa olevaan tietokantakokoelmaan halutaan lisätä uusia tietokenttiä. MongoDB:ssä skeema on joustava, joten uuden kentän lisääminen ei tuota ongelmia toisin kuin relaatiotietokannoissa voisi käydä. Tällainen olematon kenttä katsotaan MongoDB-dokumenteissa aina tyhjäksi. (Phaltankar ym. 2020, luku 2.)

BSON-tietomuoto (Binary JSON) sai alkunsa MongoDB:stä ja on käytössä MongoDB-tietokannoissa. BSON mahdollistaa nopean tiedon tallentamisen ja siirtämisen. Dokumentit ovat JSON-tietomuodon kaltaisia ja sisältävät esimerkiksi metatietoa, kuten kenttien pituudet ja alidokumentit, mikä nopeuttaa dokumentin jäsentämistä ja läpikulkua. (Phaltankar ym. 2020, luku 2.)

Suorituskyvyn optimoimiseksi MongoDB on asettanut joitakin rajoitteita dokumenteille. MongoDB rajoittaa dokumentin koon enintään 16 megatavuun ja tukemansa sisäkkäisyyden maksimisyvyyden 100 tasoon tehokkuus- ja muistinkäyttöongelmien välttämiseksi. Myös dokumenttien nimien suhteen on joitakin sääntöjä. Kentän nimi ei voi sisältää tyhjää merkkiä, vain taulukon tai upotetun dokumentin kentät voivat alkaa dollarimerkillä, kun taas pääkenttien nimet ei saa alkaa dollarimerkillä. Dokumentteja, joissa on duplikaattikenttänimiä, ei tueta. (Phaltankar ym. 2020, luku 2.)

Tietokannalle voi luoda useampia käyttäjiä, jotka ovat valtuutettuja suorittamaan tietokantatoimintoja, joihin käyttäjälle asetetun roolin oikeudet riittävät. MongoDB-tietokannassa on käyttäjille sisäänrakennettuja rooleja, kuten dbAdminAnyDatabase ja readWriteAnyDatabase. MongoDB Atlas-

pilvipalvelussa oletuksena on Read and write to any database-rooli, joka pitää sisällään luku- ja kirjoitusoikeuden. (Phaltankar ym. 2020, luku 3.)

MongoDB:n mukaan sen hyötyjä ovat muun muassa dokumenttitietokantojen helppolukuisuus, monimutkaisten objektien säilöminen, tuki useampiin ohjelmointikieliin, skaalautuvuus ja transaktiokyky sekä suuri ja kukoistava kehittäjäyhteisö (MongoDB).

2.2 Express.js

Express.js on Node.js-pohjainen sovelluskehys verkkosovellusten kehittämiseen. Kehys ratkaisee monia yleisiä ongelmia, kuten HTTP-pyyntöjen käsittelyn, sessioiden hallinnan, virheiden käsittelyn ja URL-parametrien poiminnan. Express.js käyttää middleware-komponentteja, jotka ovat kehyksen kulmakivi ja mahdollistavat HTTP-pyyntöjen ja -vastausten käsittelyn sekä eri toimintojen suorittamisen halutussa järjestyksessä. Tämä mahdollistaa kehittäjille uudelleenkäytettävän koodin ja mahdollisuuden rakentaa sovelluksia MVC-tyyppisellä rakenteella. (Mardan 2022, luku 1.)

2.3 React

React on Facebookin kehittämä käyttöliittymiä varten luotu JavaScript-kirjasto. React-kirjaston avulla kehittäjät voivat luoda yksittäisiä käyttöliittymäkomponentteja, jotka päivittyvät reaaliaikaisesti ilman, että koko verkkosivua tarvitsee päivittää. Tämä tapahtuu sen ansiosta, että React käyttää virtuaalista DOM-puuta optimoidakseen suorituskykyä. Virtuaalinen DOM on Reactin luoma kaksiulotteinen kopio oikeasta DOM-puusta muistissa. Kun muutoksia tehdään DOM-puuhun, React muuttaa vain yhtä kopiota ja vertaa sitä toiseen kopioon selvittääkseen, mitä on muuttunut. React kerää nämä muutokset yhteen paikkaan ja päivittää oikean DOM-puun kerran, mikä vähentää laskemisen ja piirtämisen tarvetta. (Narayn 2022, luku 3.)

React-komponentit mahdollistavat monimutkaisten käyttöliittymien jakamisen pienempiin itsenäisiin palasiin. Komponentteja on kolmenlaisia: funktionaaliset komponentit, luokkakomponentit ja tehdaskomponentit. Näistä tehdaskomponentti on vanhentunut eikä toimi enää Reactin versiossa 17. (Elrom 2022, luku 3.)

Komponentin elinkaari on sarja metodeja, joita voidaan kutsua aina kun komponentti ladataan tai päivitetään. Esimerkiksi render()-metodi on osa komponentin elinkaarta. Kaksi pääasiallista elinkaarta ovat asennusvaihe ja päivitysvaihe. (Banks & Porcello 2017, luku 7.) Sekä funktionaalisissa että luokkakomponenteissa voidaan käyttää React-koukkuja, jotta päästään käsiksi tilan ja komponentin elinkaaren eri toimintoihin (Elrom 2022, luku 3). Funktionaaliset komponentit ja React-koukut ovat uudempi tapa päästä käsiksi komponentin elinkaareen. Reactin vanhemmassa dokumentaatiossa kannustetaan kehittäjiä kokeilemaan React-koukkujen käyttöä uusissa komponen-

teissa. React-koukkujen odotetaan olevan ensisijainen tapa kirjoittaa React-komponentteja pidemmällä aikavälillä. (Legacy React s.a.)

2.4 Node.js

Node.js on ajoympäristö, joka soveltuu erityisesti palvelimen puolella toimivien korkean suorituskyvyn ja suuren käyttökuorman sovellusten kehittämiseen. Node.js hyödyntää asynkronisia I/O-pyyntöjä, jolloin koodin ei tarvitse odottaa vastausta vaan voi jatkaa suoritusta samanaikaisesti. Node.js käyttää Google V8-moottoria ja toimii tapahtumaperustaisesti yhdellä säikeellä. Tämä mahdollistaa muiden pyyntöjen käsittelyn ilman odottamista. (Zammetti 2022.) Node.js vastaa web-kehityksen merkittävimpiin haasteisiin, sillä se on suunniteltu nimenomaan niitä silmällä pitäen (Bawane, Gawande, Joshi, Nikam, & Prof. Bachwani 2022). Node.js-paketinhallinta NPM pitää sisällään yli 2 miljoonaa pakettia ja maailman suurin ohjelmistorekisteri (Npm s.a).

Node.js tarjoaa useita etuja, kuten skaalautuvuus ja JSON-muotoisen datan käyttömahdollisuus. Se kykenee käsittelemään useita yhteyksiä samanaikaisesti ja siirtämään tietoja tehokkaasti, mikä parantaa sovelluksen suorituskykyä. Kuitenkin liiallinen takaisinkutsujen käyttö tai suorituskykyä vaativien tehtävien runsas määrä voivat aiheuttaa haasteita, sillä Node.js on asynkroninen ja priorisoi suorituskykyä vaativat tehtävät ensin. Hankaluuksia voi myös tuottaa parhaan paketin valinta useista eri vaihtoehdoista, joita on saatavilla. (Danielkievich 8.9.2021.)

3 Määrittely

Tämän mielenterveyspalveluiden ajanvarausjärjestelmän tarkoitus on parantaa kuvitteellisen yrityksen nimeltä Mielen Voima mielenterveyspalveluiden saatavuutta, helpottaa käyttäjää varmistamaan itselleen sopivimman ajan varaamista haluamansa ammattilaisen kanssa ja auttaa mielenterveyden ammattilaisia hallinnoimaan aikataulujaan sujuvammin. Järjestelmän käyttöliittymältä vaaditaan selkeyttä ja mahdollisuutta käyttää järjestelmää sekä tietokoneella että mobiilisti.

3.1 Käyttäjätarinat

Käyttäjätarinan (user story) avulla voidaan kuvata jotakin ohjelmiston ominaisuutta loppukäyttäjän näkökulmasta. Käyttäjätarinaan pyritään tallentamaan vaatimuksen olennaiset osat eli kenelle, mitä halutaan ja miksi. (Visual Paradigm s.a.) Kaikki käyttäjätarinat ovat fiktiivisiä ja niiden tarkoitus on havainnollistaa todellisten käyttäjien tarpeita mielenterveyspalveluja tuottavan yrityksen ajanvarausjärjestelmälle.

Ajanvarausjärjestelmä palvelee kolmea eri sidosryhmää: asiakkaita, työntekijöitä ja työnantajapuolta. Asiakas on henkilö, joka käyttää yrityksen mielenterveyspalveluita ja rekisteröityessään järjestelmään voi esimerkiksi varata, tarkastella ja perua aikoja järjestelmässä. Työntekijä on mielenterveyspalveluja tarjoava henkilö, joka voi muun muassa luoda aikoja, tarkastella varausten tilaa ja peruuttaa aikoja tarvittaessa. Työnantajapuolta voisi edustaa esimerkiksi toimitusjohtaja tai palvelupäällikkö, jotka voisivat tarkastella järjestelmästä aikojen toteutumista ja palvelujen sujumista.

3.1.1 Asiakkaan käyttäjätarinat

AS1: Ajan varaaminen

Asiakkaana haluan varata ajan psykologille, jotta saan tarvitsemaani apua.

AS2: Omien aikojen tarkastelu

Asiakkaana haluan nähdä omat aikani, jotta voin esimerkiksi tarkistaa milloin varamaani aika on toteutumassa tai onko se peruttu.

AS3: Ajan peruminen

Asiakkaana haluan mahdollisuuden perua ajan, jos en pysty sitä käyttämään, jotta joku muu voi hyödyntää ajan.

AS4: Vapaiden aikojen tarkastelu

Asiakkaana haluan voida selata vapaita aikoja, jotta voin valita niistä itselleni sopivimman

AS5: Työntekijän kuvauksen tarkastelu

Asiakkaana haluan nähdä tietoja eri työntekijöistä, jotta voin varata ajan itselleni sopivimmalle tekijälle.

AS6: Palautteen antaminen

Asiakkaana haluan antaa palautetta palvelusta, jotta voin kertoa kokemuksestani ja se voidaan ottaa huomioon tulevaisuudessa.

AS7: Rekisteröityminen palveluun

Asiakkaana haluan voida rekisteröityä palveluun, jotta minun ei tarvitse syöttää tietojani varauksien yhteydessä uudestaan.

3.1.2 Työntekijän käyttäjätarinat

TT1: Uuden ajan lisääminen

Työntekijänä haluan luoda aikoja järjestelmään, jotta asiakkaat voivat varata niitä.

TT2: Hinnan määrittäminen ajalle

Työntekijänä haluan määrittää ajan luomisen yhteydessä ajan hinnan, jotta voin itse päättää paljonko veloitan palvelusta.

TT3: Sijainnin määrittäminen ajalle

Työntekijänä haluan määrittää ajan luomisen yhteydessä sijainnin, jotta voin itse päättää missä sijainnissa varattava aika tulee toteutumaan.

TT4: Omien aikojen tarkastelu

Työntekijänä haluan tarkastella omia aikojani ja niiden tietoja reaaliaikaisesti, jotta saan ajoissa tiedon mahdollisista muutoksista aikoihini.

TT5: Ajan peruminen

Työntekijänä haluan mahdollisuuden perua ajan, jos estyn sen tekemisestä, jotta myös asiakkaani saavat tiedon muutoksista aikoihini.

3.1.3 Työnantajan käyttäjätarinat

TA1: Työntekijätunnusten luominen

Työnantajana haluan luoda työntekijöille tunnuksia, jotta he voivat omatoimisesti käyttää järjestelmää.

TA2: Palvelun käyttöoikeuksien rajoittaminen

Työnantajana haluan mahdollisuuden estää ajanvarauksen asiakkaalta, jotta väärin käyttäytyvät asiakkaat eivät voi varata työntekijöille aikoja.

TA3: Kaikkien aikojen tietojen tarkastelu

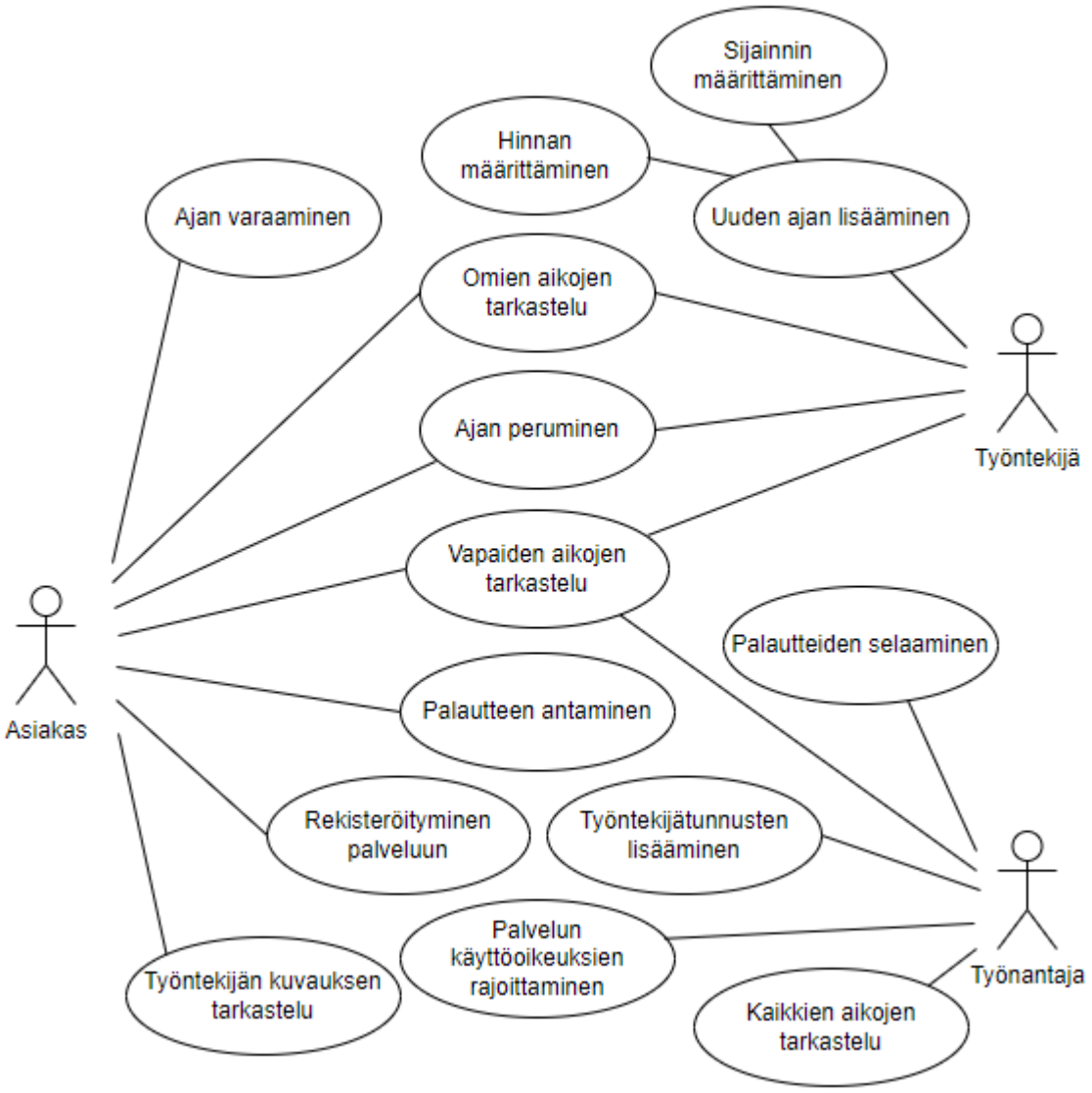
Työnantajana haluan selata näkymää, jossa näkyy kaikki järjestelmän ajat ja niiden tiedot, jotta voin seurata esimerkiksi aikojen toteutumista.

TA4: Palautteiden selaaminen

Työnantajana haluan selata asiakaspalautteita, jotta voin välittää sekä positiivista että rakentavaa palautetta työntekijöilleni.

3.2 Käyttötapauskaavio

Käyttötapauskaaviosta käy ilmi käyttötarinoiden mukaiset toiminnot, joita järjestelmässä kunkin sidosryhmän tulisi voida suorittaa.



Kuva 1. Käyttötarinoiden mukainen käyttötapauskaavio

3.3 Vaatimukset

Tyyppiluokittelun mukaan toiminnallinen vaatimus on vaatimus, joka määrittelee ohjelmiston toimintaa, kun taas ei-toiminnallinen vaatimus määrittää rajoitukset ja reunaehdot näille.

Vaatimusten prioriteetteja on tunnistettu kolme:

1. Välttämätön: Vaatimus on välttämätön järjestelmän toiminnan kannalta, jotta voidaan saavuttaa toiminnalliset tavoitteet.
2. Tarpeellinen: Vaatimus on tarpeellinen järjestelmän toiminnan kannalta, mutta ei välttämätön.
3. Hyödyllinen: Vaatimus on hyödyllinen järjestelmän toiminnan kannalta. Se tuottaa lisäarvoa yhdelle tai useammalle sidosryhmälle, mutta ei ole välttämätön eikä tarpeellinen.

Taulukko 1. Vaatimukset taulukoituna

Tunnus	Tyyppi	Kuvaus	Hyväksymiskriteeri	Prioriteetti
yksilöivä tunnus	*toiminnallinen, ei-toiminnallinen*	*vaatimus lyhyesti kuvailtuna*	*mittari tai tapa, jolla voidaan todeta, että vaatimus on otettu huomioon tai toteutettu*	*1 = välttämätön, 2 = tarpeellinen, 3 = hyödyllinen*
AS1	toiminnallinen	Ajan varaaminen	Ajan status muuttuu varatuksi eikä aikaa enää pysty varata järjestelmästä	1
AS2	toiminnallinen	Omien aikojen tarkastelu	Asiakkaalle on luotu näkymä, jossa näkyy hänen omat aikansa	1
AS3	toiminnallinen	Ajan peruminen	Ajan status muuttuu perutuksi asiakkaan toimesta	1
AS4	toiminnallinen	Vapaiden aikojen tarkastelu	Vapaat ajat ovat näkyvissä omassa näkymässään	2
AS5	toiminnallinen	Työntekijän kuvauksen	Järjestelmässä on nä-	2

		tarkastelu	kymä, jossa on nähtävillä työntekijäkuvaukset	
AS6	toiminnallinen	Palautteen antaminen	Kun ajan status on toteutunut, asiakas voi jättää palautetta saamastaan palvelusta	3
AS7	toiminnallinen	Rekisteröityminen palveluun	Rekisteröityminen onnistuu ja asiakas kirjautuu sisään onnistuneesti	1
TT1	toiminnallinen	Uuden ajan lisääminen	Ajan luomisen jälkeen aika näkyy vapaiden aikojen näkymässä	1
TT2	toiminnallinen	Hinnan määrittäminen ajalle	Ajan luomisen yhteydessä pyydetään tietoa hinnasta	2
TT3	toiminnallinen	Sijainnin määrittäminen ajalle	Ajan luomisen yhteydessä pyydetään tietoa sijainnista	2
TT4	toiminnallinen	Omien aikojen tarkastelu	Omille ajoille on näkymä, jossa näkyy muun muassa aikojen statukset	1
TT5	toiminnallinen	Ajan peruminen	Ajan status muuttuu työntekijän toimesta perutuksi	1
TA1	toiminnallinen	Työntekijätunnusten luominen	Tunnuksen luomisen jälkeen työntekijä kirjautuu sisään onnistuneesti	1
TA2	ei-toiminnallinen	Palvelun käyttöoikeuksien rajoittaminen	Asiakas ei voi varata aikoja, kun palvelun käyttö on estetty	2

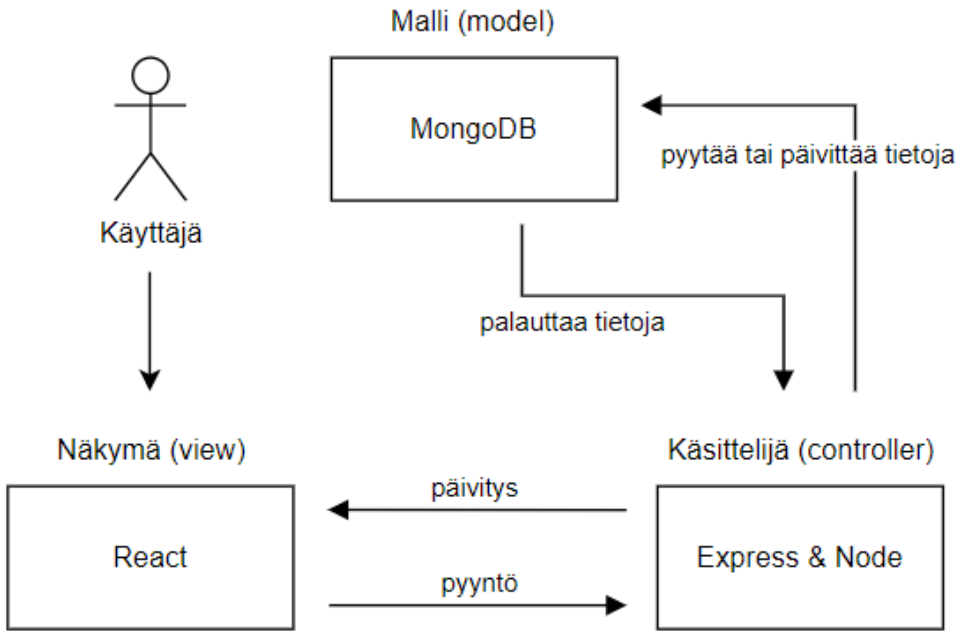
TA3	toiminnallinen	Kaikkien aikojen tietojen tarkastelu	Työnantajalle on luotu näkymä, josta näkee kaikki ajat ja niihin liittyvät tiedot	3
TA4	toiminnallinen	Palautteiden selaaminen	Työnantajalle on luotu näkymä, josta näkyy kaikki annetut palautteet	3
YH1	ei-toiminnallinen	Järjestelmää tulee voida käyttää sekä tietokoneella että mobiilisti	Kokeillaan järjestelmän eri osia ja niiden toimivuutta molemmilla laitteilla	3

4 Suunnittelu

Ajanvarausjärjestelmän suunnittelussa painotettiin kahta eri näkökulmaa: arkkitehtuurisuunnittelua ja käyttöliittymäsuunnittelua. Arkkitehtuurisuunnittelussa keskityttiin tietokanta- ja rajapintasuunnitteluun, kun taas käyttöliittymää suunniteltaessa lähdettiin liikkeelle rakentamalla käyttöliittymälle rautalankamalli, jonka jälkeen siirryttiin prototyypin luomiseen.

4.1 Arkkitehtuurisuunnittelu

Järjestelmän arkkitehtuurin suunnittelu on keskeinen osa kehitysprosessia, jonka huolellinen toteutus mahdollistaa järjestelmän optimaalisen toimivuuden. MVC on arkkitehtuurimalli, jossa sovellus jaetaan kolmeen pääkomponenttiin: malliin, näkymään ja käsittelijään. MERN-sovelluksessa MongoDB toimii datamallina, Express.js ja Node.js toimivat yhdessä käsittelijänä ja React toimii näkymänä.

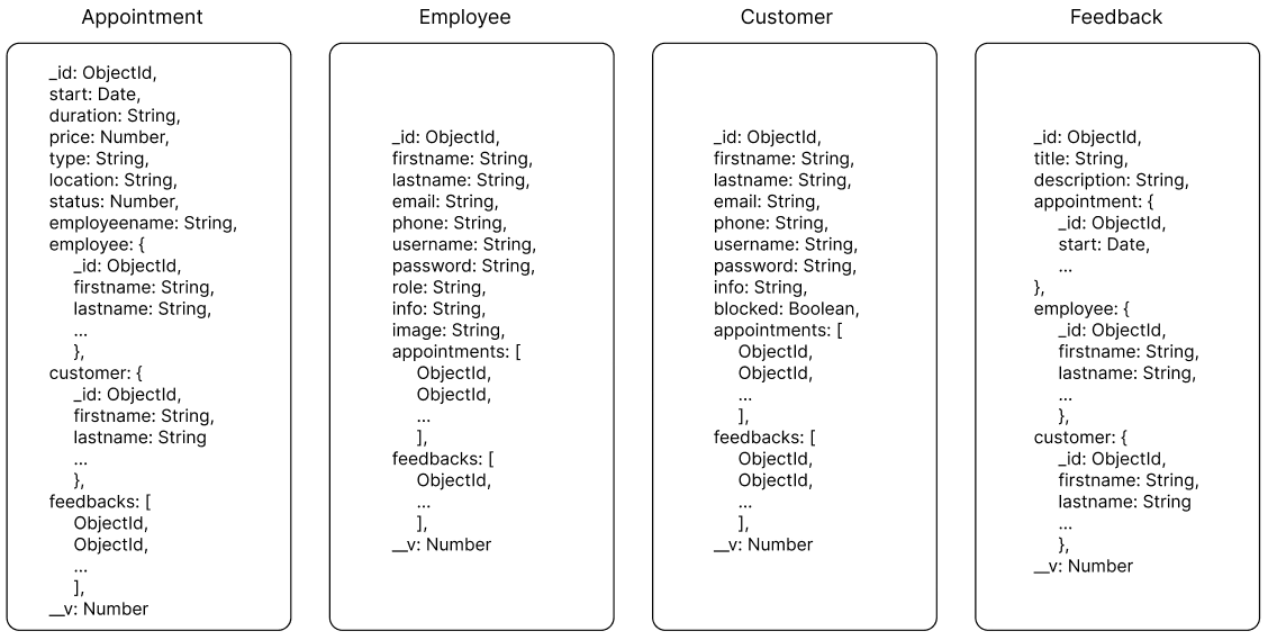


Kuva 2. MERN-sovelluksen MVC-malli

4.1.1 Tietokantasuunnittelu

Tietokantasuunnittelu on prosessi, jossa suunnitellaan tietokantaan tallennettavien tietojen rakenne ja organisaatio. Tähän kuuluu kokoelmien, kenttien ja tietokantasuhteiden määrittely. Tavoitteena on varmistaa tietojen tehokas tallennus ja helpottaa tietojen hakemista. MongoDB on dokumenttiorientoitunut tietokanta, jossa tiedot tallennetaan dokumenttiobjekteina.

Koska projektissa käytetään Node.js-ajoympäristöä, otetaan siinä käyttöön Mongoose-kirjasto, joka mahdollistaa datan rakenteen määrittämisen JSON-muodossa projektin sisällä. Kirjasto pitää sisällään apufunktioita ja -metodeja, se vähentää sisäkkäisten kutsujen monimutkaisuutta ja se palauttaa datan JSON-objektina. Skeemat ja mallit ovat Mongoose:n keskeisiä komponentteja. Skeema määrittelee kokoelman tietokenttien nimet ja tietotyypit. Mallit ovat puolestaan skeeman käännettyjä versioita, jotka käsittelevät dokumentille tehtäviä CRUD-toimintoja. (Holmes 2013, luku 1.)

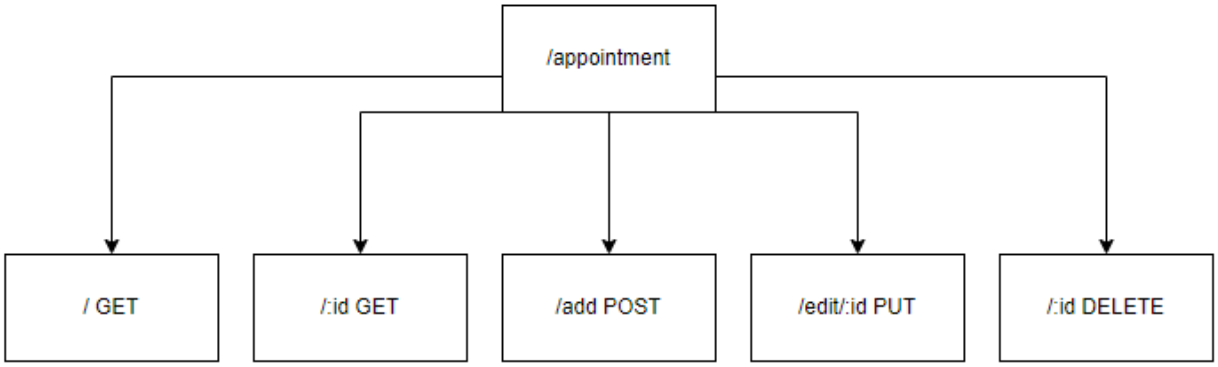


Kuva 3. Kuvaus tietokantakokoelmista ja kokoelmien tietokenttien tyypeistä

Yllä olevasta kuvasta käy ilmi tietokannalle suunniteltu rakenne ja tietokenttien tyypit. Kokoelmissa näkyvät _id-kenttä (dokumentin id) ja __v-kenttä (dokumentin versio), generoituvat dokumentteihin automaattisesti. Pakollisiksi tiedoiksi katsottiin kaikki paitsi lisätietoihin liittyvät kentät ja taulukkoja sisältävät kentät. Tällaisessa järjestelmässä tulisi säilyttää vain toiminnan kannalta pakollisia tietoja. Sen lisäksi katsottiin, että kentät sähköposti, puhelinnumero ja käyttäjätunnus tulisivat olla uniikkeja sekä käyttäjiä että työntekijöitä koskevissa tauluissa. Jotta ajanvaraushistoria säilyisi eheänä myös työntekijän tunnuksen poistamisen jälkeen, lisättiin aikojen tietoihin työntekijän nimi.

4.1.2 Rajapintasuunnittelu

Taustapalvelun rajapinta tarjoaa mahdollisuuden toteuttaa erilaisia toiminnallisuuksia tietokannan ja käyttöliittymän välillä. Tavallisimpia toiminnallisuuksia ovat CRUD-toiminnallisuudet, jotka mahdollistavat tiedon luomisen, lukemisen, päivittämisen ja poistamisen.



Kuva 4. Aikoihin liittyvien rajapintakutsujen rajapintasuunnitelma

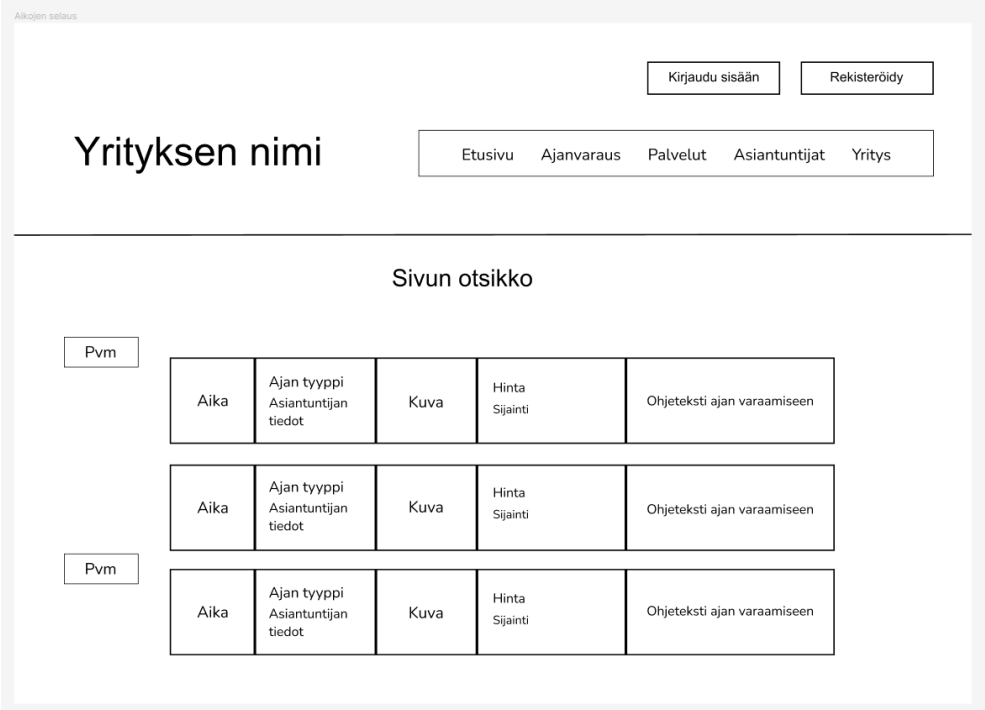
Tiedon luomiseen käytetään HTTP POST -kutsua, jossa lähetetään JSON-objekti, joka sisältää lisättävän ajan tiedot kutsun rungossa (body). Lukemiseen käytetään HTTP GET -kutsua, joka pyytää halutun tiedon palvelimelta. Kaikki ajat voidaan hakea GET-kutsulla, ja tietyn ajan hakemiseen voidaan käyttää ajan id:tä parametrina. Tiedon muokkaamiseen käytetään HTTP PUT -kutsua, ja tämän kutsun yhteydessä tarvitaan ajan id, jotta voidaan muokata tiettyä aikaa. Poistaminen tapahtuu HTTP DELETE -kutsulla, joka myös tarvitsee ajan id:n parametrina.

4.2 Käyttöliittymäsuunnittelu

Palvelun suunnittelussa korostettiin erityisesti käyttäjäystävällisyyttä ja selkeyttä. Suunnitteluprosessi alkoi rautalankamallien luomisella palvelun eri verkkosivuista, joista sitten tehtiin prototyyppi. Sekä rautalankamallit että prototyyppi suunniteltiin ensin tietokoneen näytölle ja sen jälkeen mobiililaitteelle sopiviksi. Suunnittelussa käytettiin Figma-suunnittelutyökalua.

4.2.1 Rautalankamalli

Rautalankamalli on yksinkertaistettu malli, joka auttaa hahmottamaan palvelun rakennetta. Siinä elementit kuvataan esimerkiksi yksinkertaisilla laatikoilla tai muilla helposti tulkittavilla symboleilla. Prototyypin suunnittelussa on hyvä aloittaa rautalankamallista, sillä sen luominen on nopeaa ja sen toimintaa voidaan kokeilla ennen kuin sitoudutaan enemmän aikaa vieviin kehitysvaiheisiin.



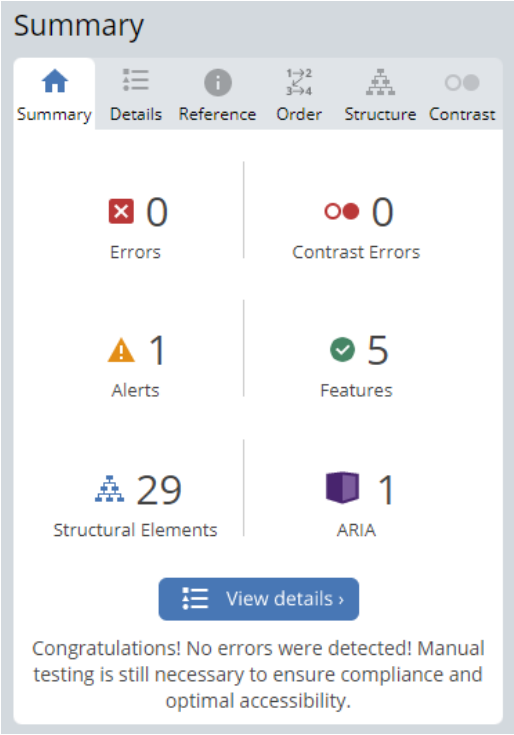
Kuva 5. Rautalankamalli ajanvaraussivusta

Kaikissa ajanvarausjärjestelmälle tehdyissä rautalankamalleissa sivun yläosa noudattaa samaa kaavaa, eli se pitää sisällään otsikon, jossa näkyy yrityksen nimi, sekä navigaation sivuston eri osiin. Lisäksi oikeassa ylänurkassa sijaitsee painikkeet, joiden kautta voi kirjautua sisään tai rekisteröityä palveluun.

Aikojen selausnäkymä on yleisesti käytetty ratkaisu ajanvaraussivustoilla, koska useimmat näistä noudattavat samankaltaista asettelua. Tyypillisesti ajanvaraussivulla vapaat ajat esitetään listauksena, jossa ajankohta on keskitetty vasemmalle, ajan tiedot sijoittuvat keskelle ja ajanvaraukseen eteneminen tapahtuu oikealta.

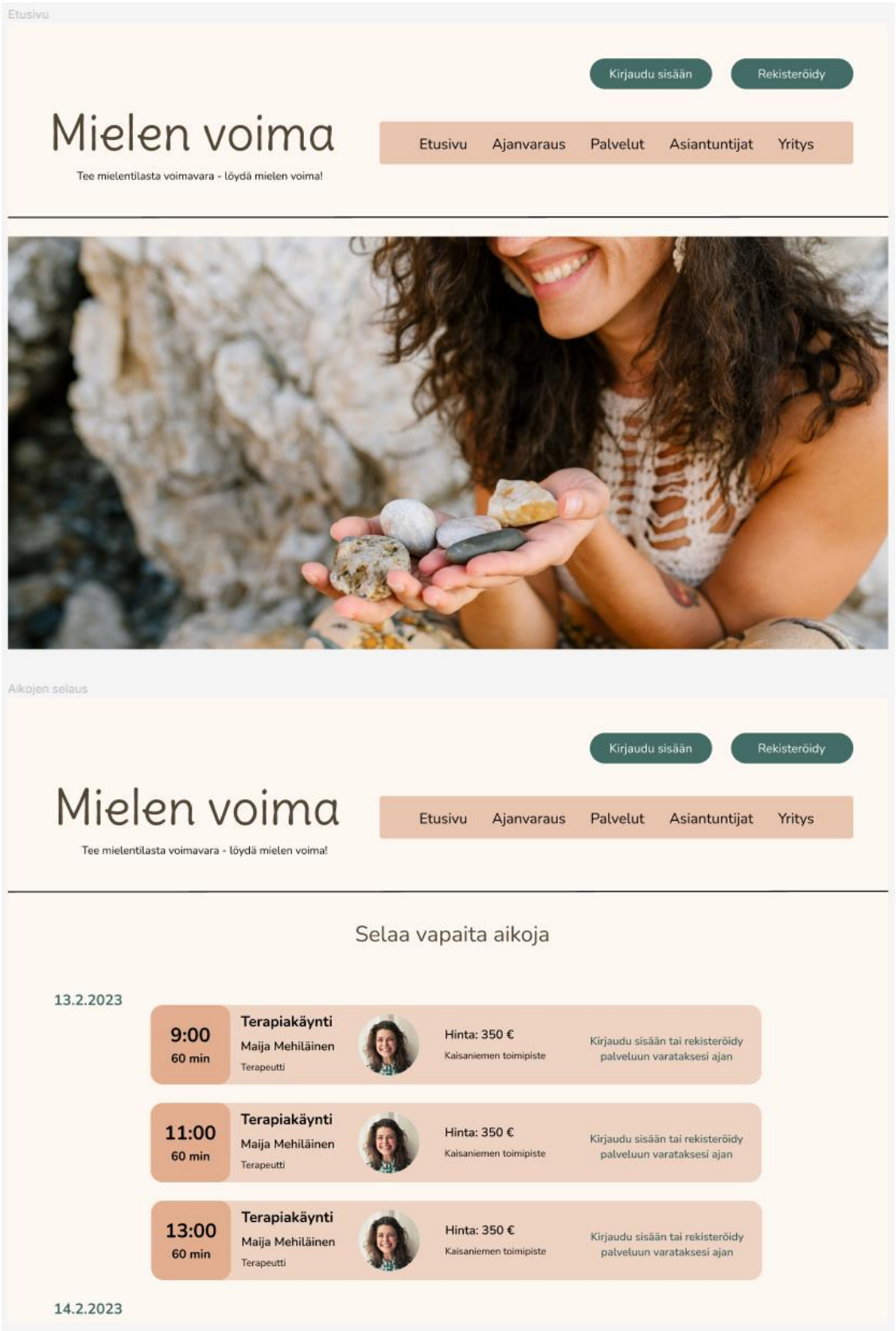
4.2.2 Prototyyppi

Projektissa prototyyppi toimii alustavana käyttöliittymän mallina, jonka avulla voidaan havaita mahdollisia puutteita, virheitä tai muita ongelmia tuotteen suunnittelussa tai toiminnassa jo ennen varsinaisen tuotteen kehittämistä. Käyttöliittymän keskeinen tavoite on tarjota käyttäjälle helppo ja ymmärrettävä käyttökokemus, jotta käyttäjän ei tarvitsisi käyttää vaivaa haluttujen toimintojen etsimiseen. Tämä edellyttää, että käyttöliittymä on selkeä, ja että sen rakenne ja toiminnallisuudet ovat helposti hahmotettavissa.



Kuva 6. Kuva Wave-työkalun yhteenvedosta ajanvaraussivulla

Värimaailman kontrastit tarkistettiin käyttämällä Chrome-selaimen Wave-lisäosaa, joka auttaa kehittäjiä tarkistamaan verkkosivustojen saavutettavuutta. Tämän avulla voitiin tarkistaa esimerkiksi kontrastit, alt-tekstit ja otsikoiden järjestys verkkosivulla. Kuvassa 7 näkyvän prototyypin visuaalisesta ilmeestä oli tarkoitus tehdä käyttäjälle mahdollisimman helposti lähestyttävä ja rauhoittava.



Kuva 7. Prototyypin etusivu ja ajanvaraussivu

5 Toteutus

Ajanvarausjärjestelmä toteutettiin MERN-pinon teknologioita hyödyntäen, käyttäen React-kirjastoa käyttöliittymäpuolella, MongoDB-tietokantaa tiedon tallentamiseen sekä Node.js-ajoympäristöä ja Express.js-sovelluskehystä taustapalvelun puolella. Projektissa on monoliittinen rakenne, eli kaikki projektin tiedostot ja resurssit sijaitsevat yhden kansion sisällä. Kansion sisällä eriytettiin käyttöliittymäpuoli ja taustapalvelun puoli omiin kansioihinsa ja niitä ajetaan omina palveliminaan.

Projekti kehitettiin Windows 11 -käyttöjärjestelmällä, johon oli asennettu Node.js versio 19.9.0, ja ohjelmoinnissa käytettiin Visual Studio Code-editoria.

5.1 Taustapalvelu

Taustapalvelun tehtävänä on hoitaa pyyntöjä käyttöliittymän puolelta, tietokantayhteyksien hallinta, käyttäjien todennus ja huolehtia sovelluksen suoritusympäristöstä. Taustapalvelua varten luotiin projektin juureen sille oma hakemisto nimeltä backend.

Node.js-ympäristön alustus tapahtuu hakemistossa komennolla npm init, ja tämä generoi projektille package.json-tiedoston. Tiedosto on JSON-muotoinen ja pitää sisällään esimerkiksi Node.js-projektin metatiedot ja riippuvuudet sekä skriptit, joilla sovellusta voidaan ajaa. Lisäosien asentaminen Node.js-ympäristöön tapahtuu Noden komennoilla, esimerkiksi nodemon-lisäosan asennus tapahtuu ajamalla terminaalissa komennon npm i nodemon. Asennuskomennot voidaan myös kirjoittaa muodossa npm install ja paketin nimi. Nodemon on työkalu, joka käynnistää sovelluksen uudelleen automaattisesti havaitessaan muutoksia koodissa (Nodemon s.a).

Express.js-kehys asennettiin myös samankaltaisesti, eli npm i express-komennon avulla. Kehyksen avulla voidaan rakentaa palvelinpuolen sovellusrajapinta (API), joka toimii tietokannan ja käyttöliittymän välissä. Express.js vastaa käyttäjän pyyntöjen käsittelystä, tietokantakyselyistä ja vastauksien lähettämisestä takaisin käyttäjälle (Mardan 2022, luku 1). Express.js voidaan ottaa käyttöön määrittelemällä se taustapalvelusovelluksen app.js-tiedostossa Express-sovellukseksi.

```
import express from "express";
const app = express();
app.use(express.json());
```

Koodi 1. Luodaan uusi Express-sovellus ja otetaan käyttöön middleware-komponentti express.json()

App.js-tiedostossa tapahtuu koko taustapalvelua koskevia konfiguraatioita, kuten tietokantaan yhdistäminen ja eri reitittimien käyttöönotto. Jotta taustapalvelun koodista tulisi mahdollisimman selkeää, se jaettiin eri kansioihin: controllers, models ja routes.

Toteutuksessa controllers-hakemistoon lisättiin jokaista tietokantakokoelmaa vastaavat kontrollerit. Näissä kontrollereissa määritellään erilaisia funktiota, jotka tukevat tarvittavia toimintoja, kuten tietojen hakemista, lisäämistä, muokkaamista ja poistamista. Alla olevan funktion avulla voidaan päivittää ajan statusta, eli esimerkiksi perua aika sekä asiakkaan että työntekijän taholta.

```
export const update = async (req, res) => {
  try {
    const id = req.params.id;
    const { status } = req.body;

    const appointment = await Appointment.findByIdAndUpdate(id, { status });

    if (!appointment) {
      return res.status(500).json({ message: "Aikaa ei voitu päivittää" });
    }

    return res.status(200).json({ appointment });
  } catch (err) {
    console.error(err);
    return res.status(500).json({ message: "Tapahtui virhe" });
  }
};
```

Koodi 2. Funktio, jonka avulla voidaan päivittää ajan status

Models-hakemistoon luotiin tietokantakokoelmia vastaavat skeemat ja routes-hakemistoon lisättiin jokaiselle tietokantakokoelmalle omat reitittimet, joille määritetään reittejä ja niihin liittyviä käsittelyfunktioita, jotka haetaan kontrollerista.

Yllä oleva update-funktio tuodaan appointmentRouter-reitittimeen ja asetetaan ajanvarauksen HTTP PUT-pyynnön reitille. Tämän reitin poluksi on asetettu ”/update/:id”.

```
appointmentRouter.put("/update/:id", update);
```

Koodi 3. Reitti ajanvarauksen päivittämistä varten

5.1.1 MongoDB Atlas

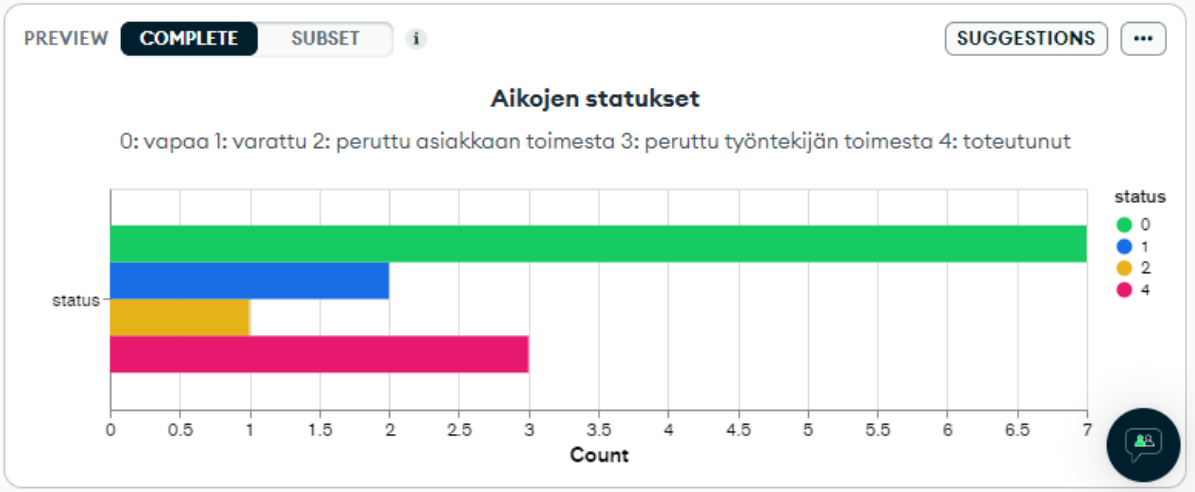
MongoDB-tietokanta voidaan määrittää eri tavoin ja tässä projektissa päädyttiin käyttämään MongoDB:n Atlas pilvipalvelua.

MongoDB Atlaksen hyötyjä (Phaltankar ym. 2020, luku 1):

- Helppo, vain muutaman askeleen käyttöönotto
- Taattu saatavuus, nodet sijaitsee eri saatavuusvyöhykkeillä
- Maailmanlaajuinen saatavuus
- Optimaalinen suorituskyky
- Erittäin turvattu, oletuksena erillinen VPC (virtuaalinen yksityinen pilvi), verkon salaus, pääsynhallinta ja palomuurit
- Automaattiset varmuuskopiot

MongoDB Atlaksen käyttö vaatii tilin luomisen MongoDB Atlas -sivustolla. Tilin luomisen jälkeen kirjaudutaan tunnuksilla sisään ja luodaan uusi klusteri, joka muodostuu ryhmästä MongoDB-palvelimia. Sivustolla luodaan uusi projekti ja projektille tietokanta. Tämän jälkeen käyttäjän tulee valita palveluntarjoaja sekä alue, johon haluaa tallentaa tietokannan tiedot. Tietokannalle luodaan myös käyttäjänimi ja salasana, joita tarvitaan yhteyden konfiguroimiseen taustapalvelun puolella. Tietoturvan puolesta on hyvä määrittää IP-osoitteet, joista pääsy verkkoon sallitaan.

Atlas tarjoaa käyttäjilleen MongoDB:n Charts-palvelun, jonka avulla käyttäjä voi luoda erilaisia visuaalisia esityksiä tietokannan tiedoista, kuten taulukoita ja kaavioita.



Kuva 8. MongoDB Chartsin avulla luotu taulukko, joka esittää kaikkien aikojen tilastot statuksen mukaan

5.1.2 Mongoose

Mongoose on npm paketti, joka tarjoaa useita ominaisuuksia skeeman määrittämiseen, tietokannan validointiin ja kyselyjen suorittamiseen MongoDB-tietokannassa. Yhteyden muodostamiseksi MongoDB Atlaksen avulla määritettyyn tietokantaan käytetään Mongoosen connect()-metodia, joka saa parametrina sivustolta haetun yhteysmerkkijonon, jossa ovat tietokannat tiedot sekä sen käyttäjätunnus ja salasana.

```
mongoose
  .connect(
    "mongodb+srv://<username>:<password>@<cluster>?retryWrites=true&w=majority"
  )
```

Koodi 4. Yhdistäminen tietokantaan Mongoosen connect()-metodilla

Mongoose-skeema määrittää millaisia tietueita tietokantaan voidaan tallentaa ja mitä ominaisuuksia niillä on. Siinä määritellään kunkin tietueen kentät, niiden tyyppi ja validointi. Skeemassa voi määrittää erilaisia kenttätyyppejä, kuten String-merkkijono, Boolean-totuusarvo tai Array-taulukko. Kenttätyyppi voi olla myös Mixed-sekoitettu, joka mahdollistaa erilaisten tietotyyppien tallentamisen kyseiseen kenttään.

Jokaiselle dokumentille generoidaan automaattisesti yksilöllinen tunniste, eli id. Tämä tarkoittaa sitä, että id:tä ei tarvitse erikseen määritellä skeemassa. Kentästä voi tehdä pakollisen asettamalla required-ominaisuuden arvoksi ”true”. Samalla periaatteella voi asettaa kentän arvon uniikiksi asettamalla unique-ominaisuuden arvoksi ”true”.

```
import mongoose from "mongoose";
const Schema = mongoose.Schema;
const feedbackSchema = new Schema({
  title: {
    type: String,
    required: true,
  },
  description: {
    type: String,
    required: true,
  },
  appointment: {
    type: mongoose.Types.ObjectId,
```

```
    ref: "Appointment",
  },
  employee: {
    type: mongoose.Types.ObjectId,
    ref: "Employee",
    required: true,
  },
  customer: {
    type: mongoose.Types.ObjectId,
    ref: "Customer",
  },
});
export default mongoose.model("Feedback", feedbackSchema);
```

Koodi 5. Palautteiden tallentamista varten luotu skeema

Mongoose-skeema määrittää millaisia tietueita tietokantaan voidaan tallentaa ja mitä ominaisuuksia niillä on. Siinä määritellään kunkin tietueen kentät, niiden tyyppi ja validointi. Skeemassa voi määrittää erilaisia kenttätyyppejä, kuten String-merkkijono, Boolean-totuusarvo tai Array-taulukko. Kenttätyyppi voi olla myös Mixed-sekoitettu, joka mahdollistaa erilaisten tietotyyppien tallentamisen kyseiseen kenttään.

Jokaiselle dokumentille generoidaan automaattisesti yksilöllinen tunniste, eli id. Tämä tarkoittaa sitä, että id:tä ei tarvitse erikseen määritellä skeemassa. Kentästä voi tehdä pakollisen asettamalla required-ominaisuuden arvoksi ”true”. Samalla periaatteella voi asettaa kentän arvon uniikiksi asettamalla unique-ominaisuuden arvoksi ”true”.

```
 employee: {
     type: mongoose.Types.ObjectId,
     ref: "Employee",
     required: true,
   },
```

Koodi 6. Työntekijätaulun yhdistäminen Appointments-tauluun

Mongoose tarjoaa erilaisia metodikutsuja, joiden avulla voi suorittaa kyselyitä tietokantaan. Esimerkiksi find()-metodi hakee tietokannasta tietoja, ja populate()-metodi mahdollistaa tietojen hakemisen ja täyttämisen viitekenttien perusteella.

```
const appointments = await Appointment.find()
      .populate("employee")
      .populate("customer");
```

Koodi 7. Haetaan kaikki ajat tietokannasta, ja kytketään niihin liittyvät työntekijä- ja asiakasobjektit

5.1.3 JSON Web Token

Projektissa käytettiin JSON Web Tokenia (JWT) käyttäjän tunnistautumiseen. JWT on avoin standardi (RFC 7519), joka tarjoaa tavan välittää tietoa turvallisesti osapuolten välillä digitaalisesti allekirjoitettujen tokenien avulla. JSON Web Tokeneita käytetään yleensä tunnistautumiseen, mutta sitä voidaan myös käyttää myös muuhun turvalliseen tiedonvaihtoon osapuolten välillä. Tokenit koostuvat kolmesta osasta, jotka erotetaan toisistaan pisteellä. Osat ovat otsikko (header), sisältö (payload) ja allekirjoitus (signature). (Jwt s.a.)

Taustapalvelun puolella asennettiin jsonwebtoken-kirjasto, jonka avulla saadaan tarvittavat toiminnot käyttöön. Funktio jwt.sign() generoi ja palauttaa JWT-tokenin, jota voi sitten hyödyntää haluamallaan tavalla. Funktiolle annetaan parametreina sisältö (payload), salainen avain (secret key) ja lisäksi voidaan syöttää parametrina asetuksia (options) ja takaisinkutsufunktio (callback). (Maison 30.7.2018.) Seuraavassa koodiesimerkissä syötetään sisältönä objekti, joka sisältää käyttäjän _id-arvon. Salainen avain haetaan taustapalvelun ympäristömuuttujista ja syötetään toisena parametrina. Kolmantena parametrina syötetään asetus expiresIn, joka määrittää tokenin vanhentumisajan. Tässä tapauksessa token vanhenee tunnin kuluttua sen luomisesta. Kun token on luotu, se asetetaan otsikon (header) Authorization-kenttään.

```
const token = jwt.sign({ _id: existingCustomer._id }, process.env.JWT_SECRET, {
    expiresIn: "1h",
  });
  return res
    .status(200)
    .set("Authorization", "Bearer " + token)
    .json({ customer: existingCustomer });
```

Koodi 8. Tokenin luominen taustapalvelun kirjautumiskutsussa

Käyttöliittymän puolella voidaan käyttää apuna res.headers.authorization-ominaisuutta, jolla haetaan kyseisen HTTP vastauksen otsikkotiedot. Tämän jälkeen poistetaan Bearer-etuliite tokenista. Lopuksi varmistetaan, että token ei palaudu arvolla ”undefined”, vaan sillä on merkkijonotyyppinen arvo.

```
const token = res.headers.authorization?.replace("Bearer ", "") || "";
```

Koodi 9. Haetaan token otsikkotiedoista

5.1.4 Bcrypt

Salasanojen kryptaamiseen käytettiin Node.js:n bcrypt-pakettia, joka voidaan asentaa komennolla npm i bcrypt. Paketti hyödyntää bcrypt-algoritmia, joka muuttaa salasanan hash-muotoon. Algoritmi generoi ensin suolan, eli satunnaisen merkkijonon. Sen jälkeen se yhdistää suolan ja salasanan ja muuttaa tämän yhdistelmän hash-muotoon. (Patel. 13.3.2023.) Bcrypt ei kuitenkaan muuta samaa arvoa joka kerta samaan hash-muotoon, vaan arvot vaihtuvat.

```
const passwordHash = bcrypt.hashSync(password);
 const customer = new Customer({
   firstname,
   lastname,
   email,
   phone,
   username,
   password: passwordHash,
   info,
   appointments: [],
 });
```

Koodi 10. Salasana tiivistetään bcypt.hashSync()-funktiolla käyttäjän luomisen yhteydessä ja tallennetaan hajautetun salasanan arvo, joka on tallennettu ”passwordHash”-muuttujaan

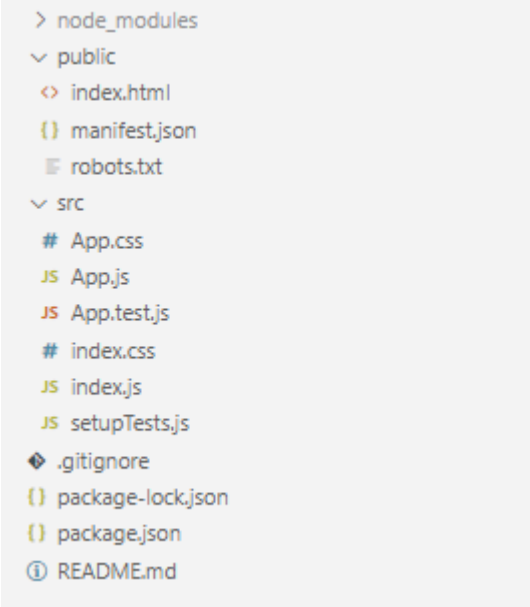
Sisään kirjautuessa bcrypt vertaa compareSync-funktion avulla annetun salasanan hash-arvoa tietokannasta löytyvän käyttäjän salasanan hash-arvoon. Jos hash-arvot vastaavat toisiaan, checkPassword-muuttuja saa arvon ”true”, muussa tapauksessa se saa arvon ”false”.

```
const checkPassword = bcrypt.compareSync(password, existingCustomer.password);
  if (!checkPassword) {
    return res.status(400).json({ message: "Password incorrect" });
  }
  return res
    .status(200)
    .json({ message: "Login successful", customer: existingCustomer });
```

Koodi 11. Salasanan tarkistuksen konfiguraatiot sisään kirjautuessa

5.2 Käyttöliittymä

Käyttöliittymän tehtävänä projektissa on tarjota käyttäjälle selkeä käyttöliittymä, jonka avulla hän voi kommunikoida helposti sovelluksen kanssa ja käyttää sen toimintoja. Single Page Application (SPA) on web-sovellus, joka toimii yhdellä sivulla, ilman että jokaista sivua täytyisi ladata erikseen. React-sovelluksessa tämä tarkoittaa sitä, että kun käyttäjä tekee toimintoja sivustolla, vain tarvittavat elementit päivittyvät ja muut pysyvät ennallaan. React-sovellus voidaan luoda käyttämällä npx create-react-app-komentoa, joka luo valmiin projektirakenteen.



Kuva 9. Yksinkertaistettu näkymä React-projektin rakenteesta käyttöliittymäpuolen alkuvaiheessa

Kansio node_modules pitää sisällään projektin riippuvuudet, kuten React-, ReactDOM- ja webpack-riippuvuudet. React-paketti tuottaa selaimessa suoritettavat HTML-, CSS- ja JS-koodit. ReactDOM toimii välittäjänä, joka renderöi React-elementit selaimessa käyttämällä root.render() -metodia. Webpack on työkalu, joka yhdistää JavaScript-tiedostot ja luo riippuvuuskaavion, jonka perusteella se luo bundle.js-tiedoston (Narayn 2022., luku 3).

Julkiset tiedostot sijaitsevat public-kansion sisällä, ja näitä ovat esimerkiksi index.html, joka on HTML-tiedosto, joka toimii projektin pääsivuna. Tiedosto manifest.json sisältää metatietoa sovelluksesta, kuten nimen, kuvakkeet ja aloitussivun URL-osoitteen. Tiedoston robots.txt avulla voi määritellä kuinka hakukoneet kommunikoivat sivuston kanssa. Sen avulla voidaan esimerkiksi määritellä mitkä hakurobotit saavat indeksoida sivua ja mitkä tiedostot tai hakemistot tulisi jättää huomiotta.

React-sovelluksen lähdetiedostot sijaitsevat src-kansion sisällä. App.js muodostaa sovelluksen rungon ja index.js on tiedosto, joka luo ja renderöi React-sovelluksen DOM-elementtiin. App.js pitää sisällään sovelluksen peruslogiikan sekä react-router-dom reititykset sovelluksen eri komponentteihin. App.css pitää sisällään tyyliohjeita App.js-komponentin ulkoasulle, ja index.css sisältää tyyliohjeet koko sovelluksen ulkoasulle. Tiedostossa setupTests.js tuodaan projektiin Jestille ominaisia funktioita DOM-elementtien testaamiseen.

Tiedosto package.json pitää sisällään projektin konfiguraation, kuten nimen, version, riippuvuudet ja scriptit. Samankaltainen tiedosto package-lock.json pitää sisällään tarkat versiot asennetuista paketeista ja niiden riippuvuuksista. Sen avulla npm voi esimerkiksi jonkun paketin puuttuessa ladata sen uudelleen tiedoston tietojen perusteella.

React-komponentteja voi kirjoittaa joko luokkakomponentteina tai funktionaalisina komponentteina. Tässä projektissa päädyttiin käyttämään funktionaalisia komponentteja, koska ne ovat yksinkertaisempia toteuttaa ja helpommin luettavia kuin luokkakomponentit.

Myös käyttöliittymälle on tarjolla useita Node Package Managerin avulla asennettavia lisäosia, jotka helpottavat kehitystyötä. Esimerkiksi react-modal lisäosan avulla voidaan luoda palautteen antamiselle modaalinen ikkuna, joka näytetään käyttäjälle keskitettynä sisältöikkunana (liite 1).

```
<Modal isOpen={isOpen} onRequestClose={onClose} className="feedback-modal">
      <h2>Anna meille palautetta</h2>
      {feedbackSent ? (
        <h3 className="centered">Kiitos palautteestasi!</h3>
      ) : (
        <form onSubmit={handleSubmit}>
          <label>
            Otsikko:
            <input type="text" value={title}
              onChange={(e) => setTitle(e.target.value)}/>
          </label>
          <label>
            Kuvaus:
            <textarea value={description}
              onChange={(e) => setDescription(e.target.value)}/>
          </label>
          <button type="submit">Lähetä</button>
        </form>
      )}
    </Modal>
```

Koodi 12. Palautemodaali.

Käyttöliittymässä on Reactin avulla helppo toteuttaa totuusarvojen avulla ehtoja, joilla tietyt elementit näkyvät käyttäjälle. Esimerkiksi seuraavassa koodinpätkässä näytetään ajanvarauspainike, mikäli käyttäjä löytyy eli isCustomer arvo on ”true” ja käyttäjää ei ole estetty eli isBlocked arvo on ”false”:

```
{isCustomer && !isBlocked && (
   <button onClick={handleEdit}>Varaa aika tästä</button>)}
```

Koodi 13. Ajanvarauspainikkeen näkymisen logiikka.

5.2.1 React Hooks

Tärkeä osa modernia React ohjelmointia ovat React-koukut. Ne mahdollistavat tilanhallinnan ilman luokkakomponentteja. Yleisimpiä koukkuja ovat useEffect()- ja useState()-koukut.

UseEffect()-koukkua käytetään usein, kun komponentissa halutaan suorittaa jotain koodia loputtomasti tai tietyssä tilanteessa. Tietty tilanne voidaan määritellä toisena parametrina annetussa taulukossa. Jos taulukko on tyhjä, koodi suoritetaan aina, kun komponentti renderöidään ensimmäisen kerran. (Narayn 2022, luku 8.)

```
useEffect(() => {
    getRole().then((data) => setEmployeeRole(data));
}, [getRole]);
```

Koodi 14. UseEffect()-koukku kuuntelee getRole()-funktion muutoksia ja kun funktio muuttuu, useEffect()-koukku suorittaa sen ja asettaa työntekijän roolin tilaan

Jos yllä olevaan koukkuun ei olisi lisätty getRole-parametria, koukku tekisi sen sisällä olevia toimintoja loputtomasti, mikä voisi johtaa suorituskykyongelmiin. Koukun avulla saadaan selville työntekijän rooli ja roolin perusteella voidaan näyttää käyttäjälle tarkoitettua sisältöä. Esimerkiksi, jos työntekijän rooli on ”admin”, hänelle voidaan näyttää työnantajapuolelle tarkoitettu navigointi.

```
{employeeId && employeeRole === "admin" && (
   <>
     <li>
       <Link to="/appointments/all">
         Selaa kaikkia työntekijöiden aikoja
       </Link>
     </li>
     <li>
       <Link to="/employees/create">Luo uusi työntekijäprofiili</Link>
     </li>
     <li>
       <Link to="/customers/list">Selaa asiakasprofiileja</Link>
     </li>
     <li>
       <Link to="/feedbacks">Selaa asiakaspalautteita</Link>
     </li>
    </>
 )}
```

Koodi 15. Työnantajapuolen navigoinnin näkymisen logiikka

Funktiokomponenteissa tilamuuttujien käyttö on mahdollista useState()-tilakoukun avulla. Koukku ottaa vastaan tilan alkuarvon argumenttina ja palauttaa taulukon. (Narayn 2022, luku 8.) Tilamuuttujaan voi asettaa monipuolista tietoa, esimerkiksi JSON-objektin. Lomakkeiden käsittelyssä käytettiin inputs-tilamuuttujaa, jolle annettiin alkuarvona JSON-objekti. Se piti sisällään avaimet käyttäjänimi ja salasana, jotka molemmat saivat alkuarvona tyhjän merkkijonon.

```
const [inputs, setInputs] = useState({
    username: "",
    password: "",
  });
  const handleChange = (e) => {
    setInputs((prevState) => ({
      ...prevState,
      [e.target.name]: e.target.value,
    }));
  };
  const sendRequest = async () => {
    const res = await axios
      .post(`/api/customer/login`, {
        username: inputs.username,
        password: inputs.password,
      })
```

```
      .catch((err) => console.log(err));

    const data = await res.data;
    return data;
  };
```

Koodi 16. UseState()-koukku luo inputs-tilamuuttujan, joka pitää sisällään kirjautumiseen liittyvät kentät, handleChange()-funktio päivittää tilaa ja sendRequest()-funktio lähettää pyynnön palvelimelle

5.2.2 Local Storage

Tässä projektissa selaimen välimuistiin (Local Storage) tallennetaan kirjautumisen yhteydessä otsikkotiedoista haettu JSON Web Token. Tokenista voi hakea purkamalla käyttäjän id:n, jota käytetään käyttöliittymässä ehtona elementtien näyttämiseen selaimessa.

```
const getCustomerIdFromToken = () => {
    if (authToken) {
      try {
        const decodedToken = jwt_decode(authToken);
        const customerId = decodedToken._id;
        return customerId;
      } catch (error) {
        console.error("Virhe JWT:n purkamisessa:", error);
      }
    }
    return null;
  };

  const customerId = getCustomerIdFromToken();
```

Koodi 17. Haetaan käyttäjän tunniste JWT-tokenista

On kuitenkin huomattava, että tämä lähestymistapa ei noudata parhaita turvallisuuskäytäntöjä. Tallentamalla tietoa selaimen välimuistiin, tehdään tieto haavoittuvaksi skriptihyökkäyksille, kuten XSS-hyökkäyksille. Turvallisempi vaihtoehto olisi säilyttää tokenit esimerkiksi HttpOnly-evästeissä, mikä rajoittaisi skriptin pääsyä niihin. (Copes 17.6.2021.)

5.2.3 Axios

Projektissa käytettiin Axios-kirjastoa, joka on promise-pohjainen HTTP-client. Se tarjoaa itsessään esimerkiksi virheiden käsittelyn sekä automaattisen HTTP-pyynnön rungon (body) serialisoimisen asianmukaiseen muotoon. Se tukee kolmea eri muotoa, joita ovat JSON, Multipart/FormData ja URL-enkoodattu muoto. (Axios s.a.) Axios palauttaa ensin lupauksen, joka mahdollistaa asynkronisen koodin suorituksen ja vastauksen saapumisen odottamisen taustapalvelulta. Pyynnöt suoritetaan asynkronisesti käyttämällä async- ja await-muuttujia.

```
const useFetch = (url) => {
  const [data, setData] = useState();

  useEffect(() => {
    const fetchData = async () => {
      try {
        const res = await axios.get(url);
        setData(res.data);
      } catch (err) {
        console.error(err);
      }
    };

    fetchData();
  }, [url]);

  return data;
};
```

Koodi 18. Haetaan aikojen tiedot hyödyntämällä Reactin useState()- ja useEffect()-funktioita

5.2.4 Router

Käyttöliittymän reitityksen käytettiin React Router-kirjastoa, joka on React-sovellusten yleisin reitityskirjasto ja suunniteltu helpottamaan sivujen navigointia. BrowserRouter sijaitsee yleensä index.js-tiedostossa ja sen tehtävänä on seurata URL-osoitteita ja välittää niitä sen alikomponenteille. Komponentissa Routes määritellään sovelluksen reitit erillisinä Route-komponentteina, joista jokainen määrittelee yhden reitin sovelluksessa. Routes-komponentti tarkistaa vastaako URL-osoite Route-komponentin polkua ja tämän perusteella renderöi oikean komponentin (Narayn 2022, luku 9).

Käyttöliittymän reititys määritellään App.js-komponentissa. Sivu pysyy Header- ja Footer-elementtien puolesta läpi käyttöliittymän samanlaisina, kun taas main-elementin sisällä oleva komponentti vaihtuu aina reitin mukaan. Etusivun jälkeisinä polkuina nähdään esimerkiksi ajanvaraussivulle vievä ”/appointments”-polku, palvelukuvauksien sivulle vievä ”/services”-polku ja asiantuntijatietoja sisällään pitävä ”/experts”-polku.

```
<header>
  <Header />
</header>
<main>
  <Routes>
    <Route path="/" element={<Home />} />
    <Route path="/appointments" element={<Appointments />} />
    <Route path="/services" element={<Services />} />
    <Route path="/experts" element={<Experts />} />
    <Route path="/company" element={<Company />} />
    <Route path="/about" element={<About />} />
    <Route path="/login" element={<Login />} />
    <Route path="/signup" element={<Signup />} />
  </Routes>
</main>
<footer>
  <Footer />
</footer>
```

Koodi 19. Käyttöliittymän reititystä App.js-komponentissa sisällä

6 Testaus

Testaaminen auttaa varmistamaan, että sovellus toimii odotetulla tavalla ja hyvin testattu sovellus on vähemmän altis virheille ja vikatilainteille, mikä tekee siitä luotettavamman ja helpommin ylläpidettävän. MERN-teknologiapinolla toteutetun sovelluksen toimintaa voi testata useilla eri tavoilla ja teknologioilla. Epäonnistuneet testitulokset paljastavat potentiaaliset ongelmat tai puutteet, joita voidaan korjata ennen ohjelmiston julkaisua. Projektin taustapalvelua testattiin yksikkötesteillä Mocha-, Chai- ja Supertest-kirjastoja käyttäen ja käyttöliittymälle suoritettiin yksikkötestejä hyödyntäen React Testing Librarya. Robot Framework- ja Selenium-työkalujen avulla löydettiin projektista kehityskohtia, joista esimerkkinä kirjautumislomakkeen virhetekstin testaaminen.

6.1 Mocha, Chai ja Supertest

Taustapalvelun ja tietokannan yksikkötesteihin käytettiin npm-paketteja Mocha, Chai ja Supertest. Mocha on sovelluskehys, joka mahdollistaa testien ajamisen sekä testiraporttien generoinnin. Chai on assertointikirjasto, joka tarjoaa erilaisia assert-tyyppejä. Supertestin avulla voidaan tarkastaa HTTP-pyynnön vastauksen statuskoodi, sisältö ja muita ominaisuuksia. (Martinez 20.9.2020.)

Esimerkiksi ajanvarauskontrollerille tehtiin yksinkertainen yksikkötesti, joka testaa GET-pyynnön polkua ”/appoinment”. Chai-kirjaston expect()-funktion avulla voidaan määrittää mitä odotetaan pyynnön palauttavan. Seuraavassa koodinpätkässä odotetaan vastausstatuksen olevan 200 OK ja vastauksen appointments-kentän olevan taulukko.

```
import { expect } from "chai";
import request from "supertest";
import app from "../src/app.js";
describe("appointment-controller", () => {
  describe("GET /appointment", () => {
    it("should get all appointments", (done) => {
      request(app)
        .get("/appointment")
        .end((err, res) => {
          expect(res.status).to.equal(200);
          expect(res.body.appointments).to.be.an("array");
          done();
        });
    });
  });
});
```

Koodi 20. Yksikkötesti, jossa haetaan kaikki ajat niiden statuksesta riippumatta

Mocha-sovelluskehys suorittaa testit ja tulostaa testituloksen terminaaliin. Testitulos sisältää tiedon testien lukumäärästä sekä eriteltynä onnistuneiden testien määrän ”passing” ja epäonnistuneiden testien määrän ”failing”. Epäonnistuneen testin tulosteesta käy myös ilmi mitä vastauksena saatiin odotetun vastauksen sijaan.

```
> backend@1.0.0 test
> mocha


  appointment-controller
    GET /appointment
Connected to database
Listening to port 5000
      ✓ should get all appointments (649ms)


  1 passing (655ms)
```

Kuva 10. Testitulos appointment-controller-testistä, joka on onnistunut

6.2 React Testing Library

Käyttöliittymän yksikkötesteissä käytettiin React Testing Library-kirjastoa sekä Jestin DOM-toiminnallisuuksia laajentavaa @testing-library/jest-dom-kirjastoa (Bennett 30.7.2020). Seuraavassa yksikkötestissä tarkasteltiin käyttöliittymän Header-komponentin renderöintiä ja asetettiin expect()-funktioille odotusarvoina tekstejä, jotka tulee löytyä komponentista, jotta testi onnistuu.

```
describe("<Header />", () => {
  it("renders the Header component", () => {
    const { getByText } = render(
      <BrowserRouter>
        <Header />
      </BrowserRouter>
    );
    expect(getByText("Etusivu")).toBeInTheDocument();
    expect(getByText("Ajanvaraus")).toBeInTheDocument();
    expect(getByText("Palvelut")).toBeInTheDocument();
    expect(getByText("Asiantuntijat")).toBeInTheDocument();
    expect(getByText("Yritys")).toBeInTheDocument();
  });
});
```

Koodi 21. Yksikkötesti, jolla testataan Header-komponentin renderöintiä

6.3 Robot Framework ja Selenium

Robot Framework on monipuolinen testiautomaatiokehys, joka kattaa erilaisia testauksen tasoja, kuten yksikkötestaus, integraatiotestaus ja järjestelmätestaus. Tässä projektissa hyödynnettiin Robot Frameworkia yhdessä Selenium-kirjaston kanssa. Robot Framework käyttää Selenium-kirjaston toimintoja suorittaakseen erilaisia toimintoja web-sivustolla, kuten sivujen avaamista ja linkkien napsauttamista (Ihnatsyeu 21.5.2021).

Ennen testien suorittamista tarvitaan muutamia asennuksia, jotta saadaan ajettua testejä. Robot Framework perustuu Python-kieleen, joten se tulee olla asennettuna järjestelmässä. Pythonin asennuksen mukana tulevan Pip-paketinhallintatyökalun avulla voidaan asentaa Robot Framework ja Selenium-kirjasto. Lisäksi testien suorittaminen edellyttää selainajurin asentamista, joka hallitsee selainta ohjelmallisesti testien aikana.

```
*** Settings ***
Library     SeleniumLibrary

*** Variables ***
${BROWSER}      Chrome
${URL}          http://localhost:3000/

*** Test Cases ***
Click Ajanvaraus Link
    Open Browser    ${URL}    ${BROWSER}
    Maximize Browser Window
    Click Link    Ajanvaraus
    Page Should Contain    Selaa vapaita aikoja
    Close Browser
```

Koodi 22. Yksinkertainen Robot Framework-testi, joka klikkaa linkkiä ja tarkistaa, että sivulta löytyy haluttu tieto

Ajettuaan testit Robot Framework tulostaa terminaaliin testitapausten tuloksen, joko onnistuneen tuloksen ”pass” tai epäonnistuneen tuloksen ”fail”. Lisäksi terminaaliin tulostuu testit ja kohdat missä testit menivät pieleen. Robot Framework tarjoaa myös testin tulosteen, raportin ja lokitiedot HTML-muodossa.

On hyödyllistä kirjoittaa monipuolisia testejä, sillä epäonnistuneiden testien avulla voidaan havaita vikoja, joita ei välttämättä muuten tulisi huomattua. Seuraavassa esimerkissä testataan kirjautumislomaketta antamalla virheelliset tunnistetiedot ja odotetaan, että sivulla näkyy virheteksti ”Kirjautuminen epäonnistui”. Aluksi testi epäonnistui, koska virhetekstiä ei ollut määritelty lomakkeelle. Kun virheteksti lisättiin lomakkeelle, testi epäonnistui uudelleen. Tämä johtui siitä, että Robot Framework yritti välittömästi etsiä virhetekstiä, kun se ilmestyi sivulle pienellä viiveellä. Siksi testiin lisättiin Sleep-komento, jonka arvoksi asetettiin 0.5, jotta Robot Framework odottaa 0.5 sekuntia ennen tekstin ilmestymisen tarkistamista sivulla.

```
*** Settings ***
Library     SeleniumLibrary

*** Variables ***
${BROWSER}      Chrome
${URL}          http://localhost:3000/login

*** Test Cases ***
Invalid Login Test
    Open Browser    ${URL}    ${BROWSER}
    Maximize Browser Window
    Input Text    name=username    invalid_username
    Input Text    name=password    invalid_password
    Submit Form
    Sleep    0.5
    Page Should Contain    Kirjautuminen epäonnistui
    Close Browser

*** Keywords ***
Submit Form
    Click Button    //button[@type='submit']
```

Koodi 23. Robot Framework -testi, jossa syötetään virheelliset tunnistautumistiedot

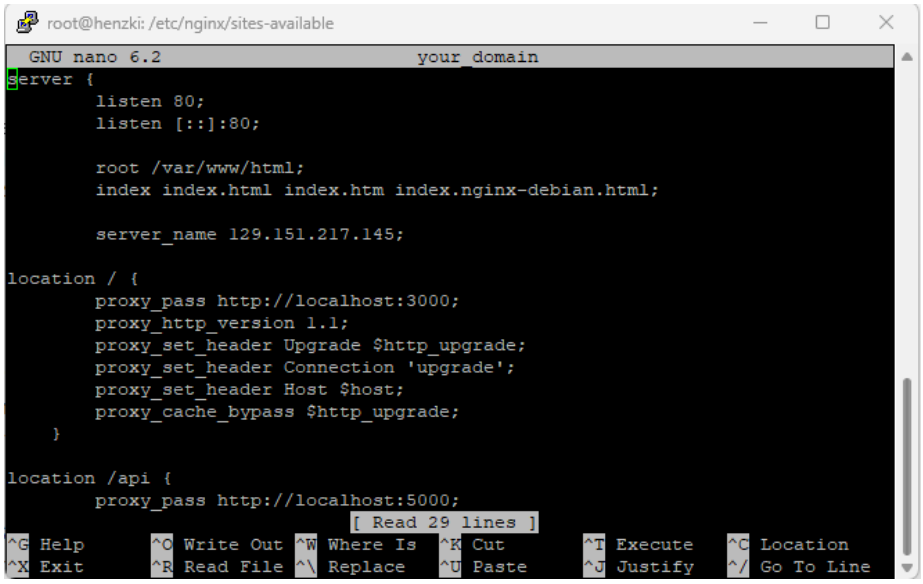
7 Julkaisu

Sovelluksen julkaisu on tärkeä vaihe sovelluksen elinkaaren aikana ja siihen on syytä valita projektiin sopivin julkaisuympäristö, kuten pilvipalvelu tai oma palvelin. MERN-sovellus on mahdollista julkaista monissa eri palveluntuottajien palveluissa ja verkosta löytyy paljon erilaisia ohjeita MERN-sovellusten julkaisuun. Projekti lisättiin Gitin avulla omaan GitHub-hakemistoonsa, josta se haettiin git clone-komennolla palvelimelle.

7.1 Oracle

Oma valintani tämän projektin julkaisuun oli Oracle-pilvipalvelu. Oraclelta löytyy paljon Always Free -resursseja, jotka yhdessä tarjoavat ilmaisen pilviympäristön (Oracle s.a). Palvelun käyttöönotto oli suhteellisen helppoa ja siihen löytyi runsaasti ohjeita verkosta. Yksinkertaistettuna Oraclen puolella käyttöönotto vaatii tilin luomisen ja instanssin luomisen. Instanssia luodessa voi valita palvelimelle erilaisia ominaisuuksia, kuten käyttöjärjestelmän ja mallipohjan, joka määrittelee prosessoriydinten -, muistin – ja muiden resurssien määrän.

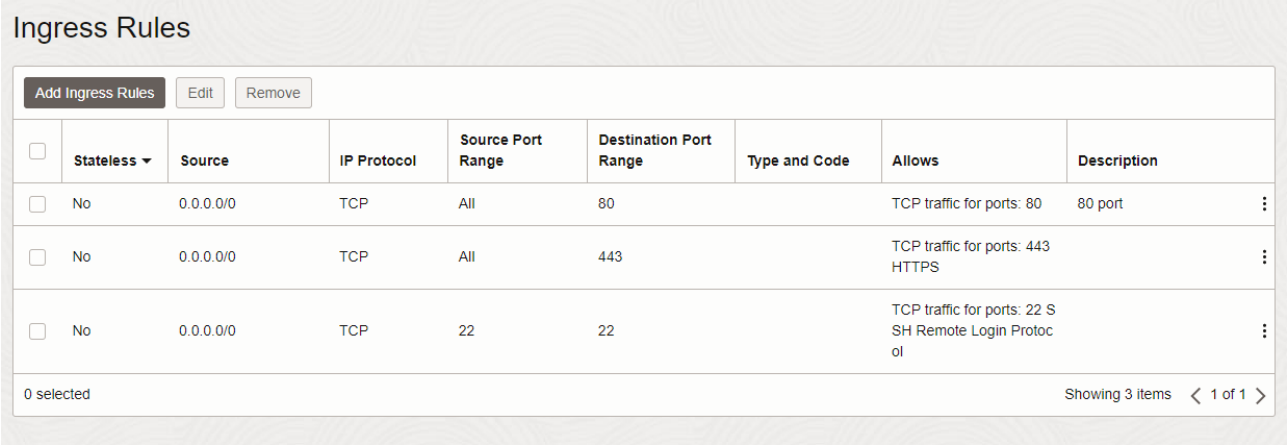
Käyttöjärjestelmäksi valikoitui Ubuntu, johon asennettiin tarvittavat työkalut ja sovellukset, muun muassa Nginx-verkkopalvelimen, Git-versionhallinnan ja Node.js-ajoympäristön, jotta sain sovellukseni käynnistettyä palvelimella. Nginx on tehokas avoimen lähdekoodin verkkopalvelin, joka päihittää muun muassa Apachen suorituskykymittauksissa (NGINX s.a). Palvelimella käytetään sen käänteistä välityspalvelua (reverse proxying), eli se toimii välikätenä selaimen ja palvelimen välillä. Käytin Ubuntu-palvelimen hallinnoimiseen ja tarvittavien pakettien asentamiseen SSH-yhteydellä PuTTy-ohjelmaa.



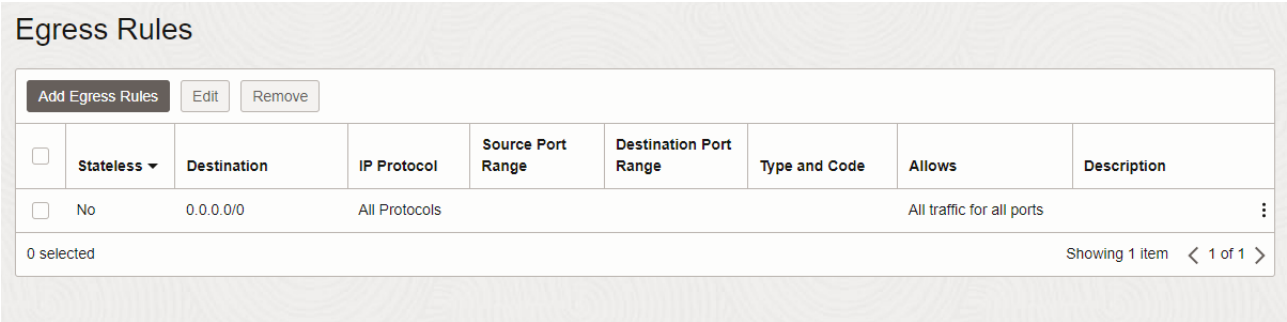
Kuva 11. Näkymä Nginx-konfiguraatioista PuTTy-konsolissa.

Oraclen ja Ubuntun palvelimen palomuuriasetuksia on syytä käydä läpi, jotta sovellukseen pääsee käsiksi ulkoverkosta. Palomuuriasetukset koostuvat saapuvan – ja lähtevän liikenteen säännöistä, jotka hallitsevat ja valvovat palvelimen verkkoliikennettä. Ingressi tarkoittaa liikennettä, joka lähetetään julkisen internetin osoitteesta yksityiseen verkkoon. Pilvessä egressi tarkoittaa liikennettä, joka poistuu yksityisestä verkosta julkiseen verkkoon. (Aviatrix s.a.) Lähtökohtaisesti palvelimeen ei saa muodostettua yhteyttä, ellei palomuuriasetuksissa ole määritetty tarvittavia lähde- ja kohdeportteja. Poikkeuksena oletusasetuksissa on avattu lähde- ja kohdeportti 22 palvelemaan SSH-yhteyttä, jolla palvelinta pääsee hallinnoimaan.

Saapuvan liikenteen säännöissä avattiin palomuuriasetukset kaikista lähdeporteista kohdeportteihin 80 ja 443 palvelimella. Lähtevän liikenteen säännöissä oli oletuksena avattu palomuuriasetukset kaikista lähdeporteista kaikkiin kohdeportteihin.



Kuva 12. Palvelimen saapuvan liikenteen säännöt.



Kuva 13. Palvelimen lähtevän liikenteen säännöt.

8 Pohdinta

Opinnäytetyön tavoitteena oli ensisijaisesti tutkia MERN-teknologiapinoa ja kehittää toimiva ja testattu ajanvarausjärjestelmä sen teknologioita hyödyntäen. Tutkimusosassa saatiin selville sekä MERN-teknologiapinon vahvuuksia ja heikkouksia esimerkiksi vertaamalla sitä muihin saatavilla oleviin teknologiapinoihin. MERN-pino soveltuu moniin eri projekteihin, mutta projektin luonteen pohjalta voi olla syytä harkita myös muita teknologiapinoja. Valinnassa tulisi ottaa huomioon projektin erityisvaatimukset ja tavoitteet. Esimerkiksi, jos projekti vaatii monimutkaisempia kyselyitä, PostgreSQL-tietokanta voi soveltua siihen paremmin kuin MongoDB-tietokanta. Pinon vahvuutena on maailman suosituimman kielen eli JavaScript-kielen käyttäminen laajasti sovelluksen eri osissa, ja suositun React-käyttöliittymäkirjaston käyttö. React erottaa MERN-pinon muista vastaavista teknologioista, ja sillä voidaan rakentaa tehokkaita, reaaliaikaisesti päivittyviä käyttöliittymiä.

Pinon avulla kehittämiseen tarvitaan Node.js-asennus sekä MongoDB:n pystyttäminen haluamallaan tavalla. Tarvittavat paketit saadaan asennettua Node.js asennuksen mukana tulevan Node Package Managerin eli npm:n kautta. Kehitysympäristöksi voi valita esimerkiksi Visual Studio Coden tai vaikka komentorivin. Projektin käyttöönotto vaatii sen kloonaamista GitHub-hakemistosta, ja pakettien asentamista npm install-komennolla projektin taustapalvelun - ja käyttöliittymän hakemistoissa sekä tietokannan yhdistämistä projektiin taustapalvelun app.js-tiedostossa.

Mielenterveyspalveluihin tehty ajanvarausjärjestelmä onnistuttiin luomaan fiktionaalisten määrittelyjen ja toteutettujen suunnittelujen mukaisesti. Valmista järjestelmää voidaan käyttää sekä tietokoneella että mobiilisti (liite 2). Liitteestä 2 käy ilmi myös vaatimusten mukainen työntekijöiden aikojen luomiseen liittyvän sivun toteutus, sekä työnantajan näkymä asiakasprofiileista, jossa on myös mahdollisuus estää asiakkailta käyttö. Projektin eri vaiheista käy ilmi ratkaisuja, joilla saatiin projektin vaatimuksia toteutettua. Toteutuksessa hyödynnettiin monipuolisesti MERN-pinon teknologioita ja tarvittavia lisäosia. Ajanvarausjärjestelmää testattiin useampaa eri testausteknologiaa hyödyntäen, ja sen avulla onnistuttiin löytämään puutteita järjestelmästä, jotka saatiin korjattua havaintojen vuoksi. Projekti julkaistiin lopuksi onnistuneesti Oraclen pilvipalveluun, kun siihen oli saatu asennettua tarvittavat työkalut, tehty tarvittavia konfiguraatioita ja palomuuriasetuksia.

Johtopäätöksenä voidaan todeta, että MERN-teknologiapino soveltuu hyvin ainakin prototyyppimäisen ajanvarausjärjestelmän kehittämiseen. Koska toteutus on vielä yksinkertaisella tasolla, ei siihen tarvita teknologioita, jotka tukevat tätä monimutkaisempia ratkaisuja. Käyttöliittymässä hyödynnettiin rautalankamalleja ja prototyyppejä ja saatiin toteutus lähelle prototyypin ulkoasua (liite 3).

Ajanvarausjärjestelmä voi olla hyödyllinen yrityksille, kuten kampaamoille tai lääkäriasemille. Järjestelmä mahdollistaa asiakkaille aikojen varaamisen verkossa, mikä vähentää manuaalista työtä ja parantaa asiakaskokemusta. Opinnäytetyön tuloksia voi soveltaa kehittämällä ja räätälöimällä ajanvarausjärjestelmää vastaamaan tietyn yrityksen tarpeita. Ohjelmistokehityksen puolesta ajanvarausjärjestelmää voi käyttää esimerkiksi pohjana tai inspiraation lähteenä muiden ohjelmistokehitysprojektien toteuttamisessa pinon avulla.

Onnistumisina pidän toimivien toiminnallisuuksien toteutusta ajanvarausjärjestelmään ja onnistunutta integraatiota kaikkien MERN-teknologiapinoon kuuluvien osien välillä. Tämä tarkoittaa, että palvelun eri sidosryhmät voivat sujuvasti käyttää ajanvarausjärjestelmää tarpeisiinsa sopivalla tavalla. Esimerkiksi asiakas voi varata aikoja järjestelmästä, työntekijä voi luoda aikoja järjestelmään ja työnantaja voi tarkastella aikojen toteutumista omassa näkymässään. Teknologioiden integraatio keskenään ei tuottanut vaikeuksia, ja koin kehitysprosessin muutenkin todella mielekkääksi alusta lähtien.

Opinnäytetyön aikana kohtasin haasteita aikataulun mukaisen toteutuksen suhteen. Vaikka olin pyrkinyt suunnittelemaan aikataulun huolellisesti, matkan varrella tuli esteitä, jotka vaikuttivat tavoitteiden saavuttamiseen suunnitellussa aikataulussa. MERN-teknologiapino on laaja kokonaisuus ja vaatii syvempää tutustumista sen osatekijöihin. Tietoperustassani kävin läpi MERN-pinon eri osia ja niiden tärkeimpiä ominaisuuksia, mutta olisin voinut käydä niitä läpi vielä perusteellisemmin. Olisin esimerkiksi voinut tehdä parempaa tutkimusta vertailemalla pinon teknologioita muihin vastaaviin tarkoituksiin tehtyihin teknologioihin.

Ajanvarausjärjestelmää olisi mahdollista jatkokehittää monin eri tavin. Yksi jatkokehitysidea olisi TypeScript-tyypityksen käyttöönotto, joka mahdollistaa muuttujien, funktioiden ja luokkien tyypin määrittämisen, mikä vähentää virheitä koodissa ja parantaa koodin luettavuutta selkeyttämällä sitä. Virheiden havaitsemisen kannalta testaaminen on tärkeää, ja sitä olisi voitu tehdä vielä enemmän kehityksen aikana, koska kattava testaus kehitysvaiheessa auttaa havaitsemaan ja korjaamaan bugeja ennen niiden pääsyä tuotantokäyttöön asti.

Henkilötietojen käsittely liittyy olennaisesti ajanvarausjärjestelmien käyttöön, joten niiden suhteen tulisi noudattaa GDPR-asetusta. Jatkokehityksessä olisi tarpeen luoda tietosuojaseloste, jossa kerrottaisiin käyttäjälle esimerkiksi mihin ja kuinka kauan heidän tietojaan säilytetään. Autentikoinnin puolesta olisi syytä miettiä tokenin säilömispaikkaa uudestaan, koska sen tulisi sijaita ennemmin esimerkiksi HttpOnly-evästeessä kuin selaimen välimuistissa, jotta siihen ei päästä käsiksi skripteillä. Ominaisuutena olisi hyvä myös lisätä hakuominaisuus, jonka avulla voisi hakea aikaa tietylle päivälle. Tällä toteutuksella vapaat ajat näkyvät kyseisestä hetkestä eteenpäin.

Opinnäytetyön aikana kehitin monipuolisesti ohjelmistokehitykseen liittyvää osaamistani. Erityisesti web-sovelluksena toteutetun ajanvarausjärjestelmän kehittäminen vahvisti Full Stack -osaamistani entisestään. Sain paremman käsityksen siitä, miten eri osat, kuten tietokanta, taustapalvelu ja käyttöliittymä, kytkeytyvät toisiinsa ja miten tällaisten sovellusten yleinen logiikka toimii. Myös ohjelmistokehityksen eri vaiheet ja niiden merkitys tulivat selkeämmin esille. Opin esimerkiksi, kuinka tärkeää on huomioida ja priorisoida eri toiminnallisuuksia web-sovelluksen määrittelyssä ja suunnittelussa.

Opin hyödyntämään MERN-teknologiapinoon kuuluvia teknologioita sekä niiden lisäosia ja apukirjastoja paremmin projektin myötä. Node Package Manager tarjoaa runsaasti erilaisia paketteja eri tarkoituksiin. JavaScript-ohjelmointi on keskeinen osa MERN-teknologiapinoa ja sain kehitettyä osaamistani sen parissa.

Lähteet

Aviatrix s.a. What do Egress and Ingress Mean in the Cloud? Luettavissa: https://aviatrix.com/learn-center/cloud-security/egress-and-ingress/. Luettu 10.5.2023.

Axios s.a. Getting Started. Luettavissa: https://axios-http.com/docs/intro. Luettu: 3.4.2023

Baiskar, Y., Paulzagade, P., Koradia, K., Ingole, P. & Shirbhate, D. 18.01.2022. MERN: A Full-Stack Development. International Journal for Research in Applied Science & Engineering Technology (IJRASET). Elektroninen tietoaineisto. Luettavissa: https://www.ijraset.com/best-journal/mern-a-full-stack-development. Luettu: 23.4.2023.

Banks, A. & Porcello, E. 2017. Learning React. O'Reilly Media, Inc. E-kirja. Luettu 23.4.2023.

Bawane, M., Gawande, I., Joshi, V., Nikam, R., & Prof. Bachwani, S A 2022. A Review on Technologies used in MERN stack. International Journal for Research in Applied Science & Engineering Technology (IJRASET). Elektroninen tietoaineisto. Luettavissa: https://www.academia.edu/68509443/A_Review_on_Technologies_used_in_MERN_stack. Luettu: 23.4.2023.

Bennett, B. 30.7.2020. Testing in React, Part 3: Jest & Jest-Dom. Luettavissa: https://javascript.plainenglish.io/testhesting-in-react-part-3-jest-jest-dom-7a8a03ae60b. Luettu: 2.5.2023.

Copes, F. 17.6.2021. JWT authentication: Best practices and when to use it. LogRocket. Blogi. Luettavissa: https://blog.logrocket.com/jwt-authentication-best-practices/. Luettu: 7.5.2023.

Danielkievich, A. 8.9.2021. A Dive into Node.js: Pros and Cons, Examples of Use. Forbytes. Blogi. Luettavissa: https://forbytes.com/blog/nodejs-pros-and-cons/. Luettu: 4.5.2023.

Elrom 2021. React and Libraries. Apress. E-kirja: Luettu: 23.4.2023.

GeeksForGeeks 18.3.2023. JavaScript Tutorial. Luettavissa: https://www.geeksforgeeks.org/javascript/. Luettu: 23.4.2023.

Holmes, S. 2013. Mongoose for Application Development. Packt Publishing. Luettu: 4.5.2023.

Ihnatsyeu, D. 21.5.2021. Robot Framework: The Ultimate Guide to Running Your Tests. BlazeMeter. Blogi. Luettavissa: https://www.blazemeter.com/blog/robot-framework. Luettu: 2.5.2023.

Jwt s.a. Introduction to JSON Web Tokens. Luettavissa: https://jwt.io/introduction. Luettu: 7.5.2023.

Legacy React s.a. Hooks FAQ. Luettavissa: https://legacy.reactjs.org/docs/hooks-faq.html#should-i-use-hooks-classes-or-a-mix-of-both. Luettu: 4.5.2023.

Maison, M. 30.7.2018. Using JWT (JSON Web Tokens) to authorize users and protect API routes. Medium. Blogi. Luettavissa: https://medium.com/@maison.moa/using-jwt-json-web-tokens-to-authorize-users-and-protect-api-routes-3e04a1453c3e. Luettu: 7.5.2023.

Mardan, A. 2022. Pro Express.js. Apress. E-kirja. Luettu: 13.2.2023.

Martinez, D. 22.9.2020. Dead-Simple API Test With SuperTest, Mocha, and Chai. Dev Tester. Blogi. Luettavissa: https://dev-tester.com/dead-simple-api-tests-with-supertest-mocha-and-chai/. Luettu: 2.5.2023.

MIELI ry. 2023. Tilastotietoa mielenterveydestä. Luettavissa: https://mieli.fi/yhteiskunta/mielenterveys-suomessa/tilastotietoa-mielenterveydesta/. Luettu: 1.5.2023

MongoDB s.a. Why Use MongoDB and When to Use It? Luettavissa: https://www.mongodb.com/why-use-mongodb. Luettu: 7.5.2023.

Murtaza, A. 19.12.2022. 8 Top Advantages of Using React for Development. Creative Tim. Blogi. Luettavissa: https://www.creative-tim.com/blog/educational-tech/top-advantages-of-using-react/. Luettu: 24.4.2023.

Narayn, H. 2022. Just React! : Learn React the React Way. Apress. E-kirja: Luettu: 11.3.2023.

NGINX s.a. What is NGINX? Luettavissa: https://www.nginx.com/resources/glossary/nginx/. Luettu: 10.5.2023.

Noble Desktop. 13.1.2023. Best Development Stacks to Use in 2023. Noble Desktop. Blogi. Luettavissa: https://www.nobledesktop.com/classes-near-me/blog/best-web-development-stacks. Luettu: 23.4.2022.

Nodemon s.a. nodemon reload, automatically. Luettavissa: https://nodemon.io/. Luettu: 4.5.2023.

Npm s.a. Build amazing things. Luettavissa: https://www.npmjs.com/. Luettu: 4.5.2023.

Oracle s.a. Always Free Resources. Luettavissa: https://docs.oracle.com/en-us/iaas/Content/FreeTier/freetier_topic-Always_Free_Resources.htm. Luettu: 10.5.2023.

Panchal, J. 14.5.2022. What Are the Benefits of Using Express.js for Developing Enterprise Applications? Rlogical Techsoft Pvt. Ltd. Luettavissa: https://www.rlogical.com/blog/what-are-the-benefits-of-using-express-js-for-developing-enterprise-applications/. Luettu: 23.4.2023.

Patel, H. 31.3.2023. Password hashing in Node.js with bcrypt. Luettavissa: https://blog.logrocket.com/password-hashing-node-js-bcrypt/#what-bcrypt. Luettu: 12.5.2023

Phaltankar, A., Ahsan, J., Harrison, M. & Nedov, L. 2020. MongoDB Fundamentals. Packt Publishing. E-kirja. Luettu 13.4.2023.

Ramotion. 20.12.2022. What Should You Know About MERN Stack. Ramotion. Blogi. Luettavissa: https://www.ramotion.com/blog/what-is-mern-stack/. Luettu: 4.5.2023.

React s.a. React. Luettavissa: https://react.dev/. Luettu: 12.5.2023.

Stack Overflow 2022. 2022 Developer Survey. Luettavissa: https://survey.stackoverflow.co/2022/. Luettu: 23.4.2023.

Statista 2022. Most used web frameworks among developers worldwide, as of 2022. Luettavissa: https://www.statista.com/statistics/1124699/worldwide-developer-survey-most-used-frameworks-web/. Luettu: 23.4.2023.

Stoyko, T. 4.2.2022. Angular Vs React Vs Vue: The Main Differences And Use Cases. Incora. Luettavissa: https://incora.software/insights/react-vs-angular-vs-vue. Luettu: 13.2.2023.

Visual Paradigm s.a. What is User Story? Visual Paradigm. Luettavissa: https://www.visual-paradigm.com/guide/agile-software-development/what-is-user-story/. Luettu: 6.5.2023.

Zammetti, F. 2022. Modern Full-Stack Development : Using TypeScript, React, Node.js, Webpack, Python, Django, and Docker. Apress L. P. Berkeley. E-kirja. Luettu: 22.2.2023

Liitteet

Liite 1. Modulaarinen ikkuna palautteen antamista varten

Liite 2. Ajanvarausjärjestelmä mobiilinäkymässä

Liite 3. Vapaiden aikojen selausnäkymä valmiissa ajanvarausjärjestelmässä