FireFit-ajanvaraus- ja -hallintajärjestelmä Keski-Suomen pelastuslaitokselle

Janne Arkko

Opinnäytetyö, AMK
Lokakuu 2024
Tieto- ja viestintätekniikan tutkinto-ohjelma (AMK)

jamk | Jyväskylän ammattikorkeakoulu
University of Applied Sciences

jamk

Kuvailulehti

Arkko, Janne

FireFit-ajanvaraus- ja -hallintajärjestelmä Keski-Suomen pelastuslaitokselle

Jyväskylä: Jyväskylän ammattikorkeakoulu. **Lokakuu 2024**, 96 sivua.

Tieto- ja viestintätekniikan tutkinto-ohjelma. Opinnäytetyö AMK.

Julkaisun kieli: suomi

Julkaisulupa avoimessa verkossa: kyllä

Tiivistelmä

Pelastustoiminnassa mukana olevien pelastajien sekä sivutoimisen pelastushenkilöstön vuosittaisesta toimintakykytestauksesta määrätään pelastuslaissa. Vuosina 2006–2010 luotiin Työterveyslaitoksen ja pelastuslaitosten yhteistyössä FireFit-arviointikäytänteet yhtenäistämään kansallinen toimintakykytestaus. Näiden arviointikäytänteiden toteutus on kuitenkin pelastuslaitosten omissa käsissä.

Keski-Suomen pelastuslaitoksella on noin 850 henkilön vahvuinen sivutoiminen pelastushenkilöstö, jonka fyysinen toimintakyky testataan yksittäisen henkilön osalta tavoitetason mukaan vuoden tai kahden vuoden välein. Näiden testien organisoimiseksi pelastuslaitoksen henkilöstö on käyttänyt alueellisesti sekä paloasemakohtaisesti lukuisia erilaisia ajanvaraustapoja ja sovelluksia.

Opinnäytetyössä kehitettiin toimeksiantajan käyttöön ajanvaraus- ja hallintajärjestelmä, joka yhtenäistää ajanvarauksiin liittyvät käytänteet sekä mahdollistaa entistä paremman ja helpomman tavan suunnitella testausprosessissa olevan virkahenkilöstön työaikaa toimintakykytestaukseen.

Tutkimustyö toteutettiin soveltavana kehitystyönä, joka perustui olemassa olevaan tietoon sekä aitoon käytännönläheiseen ongelmaan. Tutkimuksen teoria perustettiin laajalti ymmärtämään verkkosovelluksen arkkitehtuuria ja teknologian vaikutusta verkkosovellusten toimintaan.

Kehitystyön tuloksena sovellus julkaistiin verkkoon toimeksiantajan palvelimelle ja on sieltä kaikkien määriteltyjen kohderyhmien saavutettavissa sekä työpöytä-, että mobiililaitteilla sekä on käytettävyydeltään ja turvallisuudeltaan vaatimusmäärittelyn mukainen.

Avainsanat (asiasanat)

Sovelluskehitys, verkkosovellus, React.js, Node.js, käytettävyys, saavutettavuus, turvallisuus, käyttöliittymä, palvelin, tietoturva, kerrostettu suojaus

Muut tiedot (salassa pidettävät liitteet)

-

jamk

Description

Arkko, Janne

FireFit appointment booking and management system for the Central Finland Rescue Department

Jyväskylä: JAMK University of Applied Sciences, October 2024, 96 pages.

Degree Programme in Information and Communication Technology. Bachelor´s thesis.

Permission for open access publication: Yes

Language of publication: Finnish

Abstract

The annual physical fitness testing of firefighters and part-time rescue personnel involved in rescue operations is mandated by the Rescue Act. Between 2006 and 2010, the FireFit evaluation procedures were developed through a collaboration between the Finnish Institute of Occupational Health and various fire departments to standardize national fitness testing. However, the implementation of these evaluation procedures is left to the discretion of individual fire departments.

The Central Finland Rescue Department has a part-time rescue staff of about 850 people, whose physical fitness is tested according to target levels every one or two years for each individual. To organize these tests, the rescue department's personnel have used a variety of different regional and station-specific scheduling methods and applications.

In this thesis, a scheduling and management system was developed for the client to standardize scheduling practices and to provide a better and easier way to plan the work hours of official staff involved in the fitness testing process.

The research was conducted as applied development work, based on existing knowledge and addressing a practical, real-world problem. The theoretical foundation of the research was broadly aimed at understanding the architecture of web applications and the impact of technology on the functioning of web applications.

As a result of the development work, the application was published on the client's server and is accessible to all defined target groups on both desktop and mobile devices. Its usability and security meet the requirements outlined in the specification.

Keywords/tags (subjects)

Application development, Web application, React.js, Node.js, Usability, Accessibility, Security, User interface, Server, Data security, Layered protection

Miscellaneous (Confidential information)

-

Sisältö

1 Ajanvarauksen hallinnasta sujuvuutta arkeen ... 4
1.1 Tausta ja tavoite ... 4
1.2 Toimeksiantaja ... 5
2 Tutkimusasetelma ... 6
2.1 Tutkimusongelma ... 6
2.2 Tutkimuskysymykset ... 6
2.3 Tutkimuksellinen rajaus ... 6
2.4 Tutkimusmenetelmät ... 7
2.5 Luotettavuus ja eettisyys ... 7
3 Vaatimusmäärittely ... 8
3.1 Palvelukuvaus ... 8
3.2 Kohderyhmät ... 9
3.3 Sidosryhmät ... 10
3.4 Toiminnalliset vaatimukset ... 11
3.5 Laadulliset vaatimukset ... 12
3.6 Käyttöliittymäkuvaus ... 13
4 Verkkosovellusten arkkitehtuuri ... 16
4.1 Ohjelmointikielet ... 16
4.1.1 JavaScript ... 17
4.1.2 TypeScript ... 19
4.1.3 Python ... 20
4.2 Palvelin ja ajoympäristöt ... 20
4.2.1 Node.js ... 20
4.2.2 CPython ... 24
4.3 Sovelluskehykset ... 25
4.3.1 React.js (Node.js) ... 25
4.3.2 Django (Python) ... 29
4.3.3 Flask (Python) ... 29
4.4 Tietoturvallisuus verkkosovelluksissa ... 29
4.4.1 Tietoturva ... 29
4.4.2 Tietosuoja ... 30
4.4.3 Tietoturvan työkalut verkkosovelluksissa ... 31

5 Verkkosovelluksen kehitys ... 36
5.1 Suunnittelu ... 36
5.1.1 Ohjelmointikielet ... 36
5.1.2 Palvelin ja ajoympäristö ... 37
5.1.3 Sovelluskehys ... 38
5.1.4 Tietoturvallisuus ... 39
5.2 Palvelin ... 40
5.2.1 Rakenne ... 40
5.2.2 Server.js ... 41
5.2.3 Rajapinnat ... 45
5.2.4 Middleware ... 59
5.2.5 Tietokanta ... 60
5.2.6 Konfiguraatio ja apuohjelmat ... 62
5.3 Käyttöliittymä ... 67
5.3.1 Rakenne ... 67
5.3.2 Layout ... 70
5.3.3 API-komponentti ... 74
5.3.4 Hallintapaneeli ... 75
5.3.5 Käyttäjänäkymä ... 80
5.3.6 Kirjautumissivu ... 83
5.3.7 Middleware ja apuohjelmat ... 85
6 Pohdinta ... 87
Lähteet ... 90
Liitteet ... 95
Liite 1. Käyttöliittymän kuvankaappauksia ... 95
Liite 2. Implementoitu kerrostettu suojaus ... 96

Kuviot

Kuvio 1. Keski-Suomen pelastuslaitoksen organisaatio (Meistä N.d). ... 5
Kuvio 2. Suunniteltu testattava-roolin käyttöliittymä ... 13
Kuvio 3. Suunniteltu testaaja-roolin käyttöliittymä ... 14
Kuvio 4. Suunniteltu esihenkilö- ja pääkäyttäjä-roolien käyttöliittymän Varaukset-välilehti ... 14
Kuvio 5. Esihenkilö- ja pääkäyttäjä-näkymän Statistiikka-välilehden suunnitelma ... 15
Kuvio 6. Esihenkilö- ja pääkäyttäjänäkymän Suunnittelu-välilehden suunnitelma ... 15
Kuvio 7. 10 suosituinta ohjelmointikieltä 2023 (Cass 2023). ... 16

Kuvio 8. DOM-puu (Robie N.d). 17
Kuvio 9. Käytetyimmät sovelluskehykset 2020 (2020 Developer Survey 2020). 19
Kuvio 10. Esimerkki Node.js moduulista ja sen käytöstä 22
Kuvio 11. Palvelinmalli Express-kirjastolla 23
Kuvio 12. Palvelinmalli ilman Express-kirjastoa käyttäen moduuleja 23
Kuvio 13. Hypoteettinen palvelinmalli ilman moduuleja tai kirjastoja 24
Kuvio 14. React.js komponentti ja sen käyttö 27
Kuvio 15. Ominaisuuden välittäminen komponentille 28
Kuvio 16. OWASP Top Ten 2021 (OWASP Top Ten 2011). 30
Kuvio 17. Kerrostettu suojaus (Into 2011) 32
Kuvio 18. Parametrisoimaton ja parametrisoitu SQL-kysely 35
Kuvio 19. Palvelimen hakemistorakenne 40
Kuvio 20. Tietokantataulujen relaatiot 60
Kuvio 21. Tietokantataulut staattisille muuttujille 61
Kuvio 22. Navigaatio- ja sivupalkki 72
Kuvio 23. UserInfo-komponentti 73
Kuvio 24. Statistiikka-välilehti 77
Kuvio 25. Henkilöstö-välilehti 78
Kuvio 26. Hallintapaneelin Varaukset-välilehti 78
Kuvio 27. Suunnittelu-välilehti 79
Kuvio 28. Ilmoitustaulu-välilehti 80
Kuvio 29. Kalenteri-välilehti 81
Kuvio 30. Käyttäjän uusi ajanvaraus 82
Kuvio 31. Käyttäjänäkymän Varaukset-välilehti 82
Kuvio 32. Kirjautumissivu 84
Kuvio 33. Virheenhallinta kirjautumissivulla 84
Kuvio 34. DialogBox-komponentti 86
Kuvio 35. Palaute-modaali 86

Taulukot

Taulukko 1. Toiminnalliset vaatimukset 11
Taulukko 2. Toiminnallisuuksien CRUD-taulukko 12
Taulukko 3. Laadulliset vaatimukset 12

1 Ajanvarauksen hallinnasta sujuvuutta arkeen

1.1 Tausta ja tavoite

Pelastuslaki uudistettiin vuonna 2011, jolloin siihen tuli maininta, että pelastustoimintaan osallistuvan henkilön tulee ylläpitää tehtäviensä edellyttämiä perustaitoja ja fyysistä kuntoa. Tämän lain valvonta ja toteutus kuuluu maakunnan alueella toimivalle pelastusviranomaiselle. (L 379/2011.)

Fyysisen toimintakyvyn testaukseen ja arviointiin liittyvien käytänteiden yhtenäistämiseksi Työterveyslaitos toteutti vuosina 2006–2007 ja 2008–2010 yhteistyössä valittujen pelastuslaitosten, niiden työterveyshuoltojen sekä Aino Health Management Oy:n kanssa kehittämishankkeen, jonka tuloksena syntyi FireFit-arvointikäytäntö pelastajien toimintakyvyn arviointiin. (Lusa, Halonen, Punakallio, Wikström, Lindholm & Luukkonen 2015, luku 1.2.)

Siitä asti, kun FireFit-arviointikäytänteet tuotiin osaksi pelastuslaitosten arkea, on niiden valvonnan ja toteutuksen prosessiin käytetty lukuisia erilaisia keinoja ja sovelluksia. Lusa ym. (2015, luku 1.3) mukaan prosessissa on haluttu korostaa sitä, että toimintakyvyn arviointiin liittyvällä henkilöstöllä on arvioinnin tueksi ja työajan suunnittelun pohjaksi yhteys työterveyshuoltoon sekä henkilöstöhallintoon. Lusa ym. (2015, luku 1.3), mainitseekin, että ”pelastuslaitokset ovat organisoineet testaustoiminnan monella eri tavalla.”

Tämän opinnäytetyön lopputuloksena kehitettiin edellä mainittujen arviointikäytänteiden tueksi ja organisoinnin helpottamiseksi ajanvaraus- ja hallintajärjestelmä, jonka kautta lakisääteisen toimintakyvyn arvioinnin piirissä olevien virka- ja sopimushenkilöiden vuosittainen työkyvyn arviointi ja organisointi helpottuu.

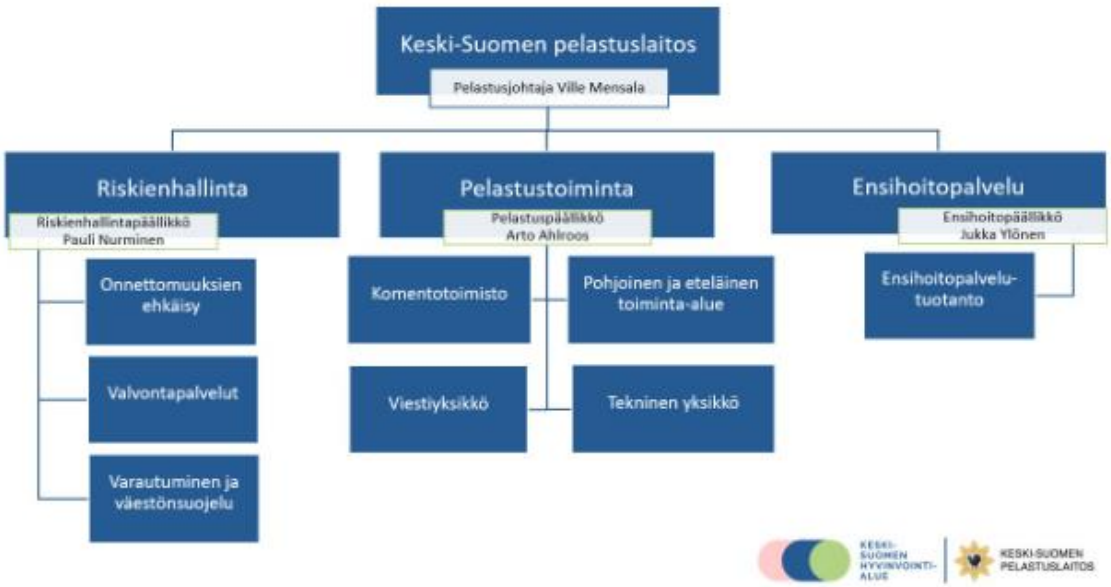
Tavoite oli, että tuloksena syntynyt sovellus on käytettävyydeltään, saavutettavuudeltaan sekä yleisilmeeltään käyttötarkoitukseensa sopiva, noudattaa vaatimusmäärittelyä ja käyttää moderneja ohjelmointiteknologioita sekä on helposti muokattavissa tulevaisuudessa.

1.2 Toimeksiantaja

Toimeksiantajana tälle opinnäytetyölle toimii Keski-Suomen pelastuslaitos. Keski-Suomen pelastuslaitos on vuonna 2004 perustettu Keski-Suomen maakunnan alueella toimiva pelastuslaitos, joka on osa Keski-Suomen hyvinvointialuetta. Se on yksi maamme 21 pelastuslaitoksesta ja sen vastuualueeseen kuuluu noin 275 000 asukasta (Kuntien avainluvut 2023). Keski-Suomen pelastuslaitoksella työskentelee päätoimisissa pelastustoimen ja ensihoidon tehtävissä noin 400 henkilöä sekä noin 850 sopimushenkilöä jaettuna 45 paloasemalle ympäri maakunnan aluetta. (Meistä N.d.)

Keski-Suomen pelastuslaitos vastaa palvelutasopäätöksessään määriteltyjen pelastustoimen palveluiden tuottamisesta, joihin lukeutuu muun muassa pelastustoiminta, onnettomuuksien ehkäisy, väestönsuojelu sekä varautuminen häiriö- ja poikkeustilanteisiin (Keski-Suomen pelastuslaitoksen palvelutasopäätös vuosille 2021–2024 2021).

Keski-Suomen pelastuslaitoksen organisaatio (ks. kuvio 1.) koostuu kolmesta tulosalueesta: pelastustoiminta, ensihoitopalvelu sekä riskienhallinta. Pelastuslaitoksen ylin viranhaltija on pelastusjohtaja, joka toimii suorassa alaisuudessa hyvinvointialueeseen. Pelastuslaitoksen toiminnan ylimpänä päätöksentekijänä toimii hyvinvointialueen aluehallitus ja aluevaltuusto. (Meistä N.d.)



Kuvio 1. Keski-Suomen pelastuslaitoksen organisaatio (Meistä N.d).

2 Tutkimusasetelma

2.1 Tutkimusongelma

Vuoden 2011 pelastuslain muutoksen jälkeen FireFit-arviointikäytänteitä ja vuosittaista fyysisen toimintakyvyn testausta on järjestetty alueellisesti hyvin vaihtelevasti. Ongelmaksi muodostuu Keski-Suomen pelastuslaitoksen alueella toimivan sopimushenkilöstön vuotuisen testauksen järjestäminen. Keski-Suomen pelastuslaitoksen sopimushenkilöstön vahvuus on noin 850 henkilöä. (Lusa ym. 2015; Meistä N.d.)

Erilaisia ajanvarausmenetelmiä on vuosien saatossa muodostunut pelastuslaitoksen alueella lukuisia sekä alueiden, että paloasemien välillä on myös eroja ajanvarauskäytäntöjen välillä. Monet käytännöistä ovat edelleen toimivia, mutta käytänteiden ja ajanvarausmenetelmien yhtenäistäminen on koko henkilöstön yhteinen etu. (Monthan 2024.)

2.2 Tutkimuskysymykset

Kehittämistyön tavoitteena oli kehittää verkkosovellus, joka vaatimuksiltaan vastaa laadukkaasti toteutettua ja jokapäiväisessä käytössä toimivaa sekä eri selaintyypeille skaalautuvaa sovellusta. Sovelluksen toteutuksessa käytettiin tutkimusongelman ratkaisua, että vaatimusmäärittelyä tukevia olemassa olevia teknologioita ja kehitysmenetelmiä.

Kehitystyön keskeisimmät, vaatimusmäärittelystä johdetut tutkimuskysymykset olivat:

- Kuinka luoda käytettävyydeltään ja saavutettavuudeltaan laadukas sovellus, jonka ominaisuudet vastaavat kaikkien sidosryhmien tarpeisiin?
- Mikä vaikutus erilaisilla arkkitehtuureilla ja ohjelmointikielillä on lopputulokseen?
- Kuinka verkkosovelluksesta tehdään turvallinen?

2.3 Tutkimuksellinen rajaus

Kehittämistyön lopputuloksena syntynyt verkkosovellus on laajuudeltaan ja ominaisuuksiltaan niin laaja, että tämän opinnäytetyön ulkopuolelle rajataan sovelluksen käyttämät ulkoiset rajapinnat ja

yhteydet muihin järjestelmiin. Tämä tutkimusraportti keskittyy kehitetyn sovelluksen arkkitehtuuriin ja esitettyihin tutkimuskysymyksiin vastaamiseen.

Kehittämistyön sidosryhmiin ei myöskään kuulu Keski-Suomen pelastuslaitoksen operatiivisessa toiminnassa toimivaa virkahenkilöstöä, vaan kehittämistyö on myös rajattu koskemaan ainoastaan pelastuslaitoksen sopimushenkilöstöä sekä tämän henkilöstön vuosittaiseen testausprosessiin osallistuvaa virkahenkilöstöä.

2.4 Tutkimusmenetelmät

Tämä kehittämistyö toteutettiin käyttäen soveltavaa tutkimusmenetelmää, joka soveltuu hyvin olemassa olevan tiedon soveltamiseen ja ratkaisemaan jokin aito työelämälähtöinen ongelma. Soveltava tutkimus on ennen kaikkea kokoelma erilaisia menetelmiä ja se voi sisältää erilaisia lähestymistapoja (Pernaa 2013).

Pernaan (2013) mukaan keskeisintä soveltavassa tutkimuksessa on teoriaan pohjautuva kehittäminen ja tästä syystä tämän tutkimuksen teoria on kasattu verkkosovellusten kehittämiseen vahvasti sidoksissa olevista aiheista muun muassa arkkitehtuurista ja ohjelmointikielistä. Teoria-aineistoa kerätään asiasanojen avulla, tutkimuksen tekijän oman koulutuksen, ammattitaidon sekä kokemusten perusteella, että tekoälymallien suosittelemien asiakohtien perusteella kirjallisista ja luotettavista lähteistä.

Soveltavaan tutkimukseen kuuluu vahvasti suunnitelmallisuus sekä yhteistyö sidosryhmien välillä (Pernaa 2013), joten kappaleessa kolme (3) on käsitelty tutkimustyön lopputulosta ohjaava vaatimusmäärittely, joka on luotu yhteistyössä sidosryhmien kanssa.

2.5 Luotettavuus ja eettisyys

Yleisesti tieteellisen tutkimuksen luotettavuutta arvioidaan sen pätevyyden ja toistettavuuden avulla, mutta soveltavan tutkimuksen luotettavuuden arviointi on haasteellista lopputuloksen mahdollisen laajuuden tai tulosten salassapidollisten asioiden vuoksi. Kehittämistutkimuksen luotettavuutta voidaan kuitenkin arvioida kuvaamalla kehittämisen vaiheita sekä pyrkimällä käyttämään teorioita, jotka ohjaavat ja kuvailevat kehittämistyötä. Kehittämistyön prosessiin tulee myös

kuulua tosiasiallista testaamista kehittämistyön lopputulokselle suunnitellussa ympäristössä. (Pernaa 2013.)

Hyvä tieteellinen käytäntö ohjaa soveltavan kehittämistyön tekijöitä pohtimaan kehittämistyötä myös eettisten kysymysten osalta. Kehittämistyön toteutuksessa tulee käyttää avoimia ja läpinäkyviä menetelmiä sekä dokumentoida kehittämistyö kattavasti. Sidosryhmät ja osallistujat tulee pitää tiiviisti kehitystyössä mukana sekä pitää heidät tietoisina kehityksen vaiheista. Tallennettujen henkilötietojen tai muiden tietojen lakien ja asetusten mukainen käsittely sekä luottamuksellisten tietojen salassa pysyminen ovat myös hyvän tieteellisen käytännön mukaisia. (Liimatainen, Hautamäki, Kirjalainen, Kokko, Korhonen, Laitinen-Väänänen, Norvapalo, Törn-Laapio & Hyvätti 2024, luku 3.1.)

Tämän kehittämistyön kokonaisvaltaista luotettavuutta pystytään arvioimaan käytettyjen teknologioiden ja arkkitehtuuristen valintojen, osalle sidosryhmistä avoimen koodipohjan sekä autenttisen testauksen kautta. Testaus järjestetään testaukseen vapaaehtoisesti ilmoittautuneista sidosryhmien henkilöistä ja testaukseen liittyy vahvasti vapaan palautteen antamisen ilmapiiri.

Tämän kehittämistyön aikana on hyödynnetty OpenAI:n ChatGPT 4o-mallia ohjaamaan ja neuvomaan erityisesti ohjelmointivaiheen aikana ilmenneistä ongelmista sekä uusien tekniikoiden ja kirjastojen käyttöönottoon liittyvistä yksityiskohdista. Tekoälyä ei ole käytetty teorialähteenä tutkimuksen missään vaiheessa, eikä tekoäly ole yksin vastuussa mistään kehitystyön yksittäisestä osasta tai ohjelmakoodista, vaan tekoälyn ehdotukset on käyty kriittisesti läpi ja niiden hyödyntämistä tutkimustyössä on tarkasti pohdittu. Tekoälymalli ChatGPT 4o ei lähtökohtaisesti kykene luomaan suoraan toimivaa ohjelmakoodia, joka väkisinkin pakottaa tutkimuksen tekijän enemmänkin referoimaan tekoälymallin tuottamaa tietoa omaan tutkimukseensa kuin suoraan kopioimaan sitä.

3 Vaatimusmäärittely

3.1 Palvelukuvaus

Tämän kehittämistyön lopputuloksena syntyvän verkkosovelluksen tavoitteena on tarjota kokonaisvaltainen ajanvaraus- ja hallintajärjestelmä, joka vastaa kohde- ja sidosryhmien tarpeeseen

keskittää vuosittaiseen kuntotestaukseen liittyvän prosessin ajanvaraustoiminta. Sovellus suunnitellaan toimimaan eri päätelaitteilla, mukaan lukien mobiililaitteet ja työpöytäselaimet, jotta käyttäjäkokemus pysyy johdonmukaisena riippumatta käytetystä laitteesta.

Sovelluksen keskeinen tarkoitus on toimia alustana pelastuslaitoksen sopimushenkilöstön kuntotestaukseen liittyvään ajanvaraukseen, jonka kautta sopimushenkilöstö pystyy helposti ja luotettavasti varaamaan aikoja kuntotestaukseen. Sovellus yhdistää reaaliaikaiset päivitykset ja intuitiivisen käyttöliittymän tarjotakseen sujuvan ja helppokäyttöisen kokemuksen.

Sovelluksen ydin on turvallinen ja käyttäjäystävällinen käyttöliittymä, jossa käyttäjät voivat muun muassa varata testausaikoja, hallita ja muokata omia tietojaan sekä lukea ja lähettää viestejä ja ilmoituksia. Sovellukseen tehdään kaksi eri roolipohjaista käyttöliittymää: testattava- ja testaajaroolien käyttöliittymä sekä pääkäyttäjä- ja esihenkilöroolien käyttöliittymä.

Testattava- ja testaaja -roolien käyttöliittymä suunnitellaan tavallisille käyttäjille ja testaajan roolissa toimiville käyttäjille. Tämän käyttöliittymän kautta käyttäjät pystyvät muun muassa varata testausaikoja, tarkastella ja muokata omia tietojaan sekä lukea sovelluksen ilmoitustaulua. Pääkäyttäjille sekä esihenkilöille suunniteltavan käyttöliittymän tulee näyttää sovelluksen ydintoiminnan statistiikkaa päätöksenteon tueksi, mahdollistaa testaukseen käytettävän työajan suunnittelu sekä kyky lisätä ja hallita käyttäjäkuntaa.

Sovelluksesta rakennetaan turvallinen käyttöympäristö, jossa käyttäjien tiedot suojataan. Sovelluksessa käytetään moderneja teknologioita mikä mahdollistaa nopean kehityksen ja tehokkaan toiminnan eri sovellusalustoilla. Tietoturva on keskeinen osa sovellusta, ja kaikki käyttäjätiedot käsitellään ja säilytetään GDPR:n mukaisesti. Käyttäjätiedot suojataan vahvalla autentikoinnilla ja salatuilla tiedonsiirtomenetelmillä, mikä varmistaa sekä tietoturvan, että tietosuojan korkean tason.

3.2 Kohderyhmät

Sovelluksen kohderyhmänä ovat erityisesti seuraavat käyttäjäryhmät, joilla on tarpeita, tavoitteita ja velvollisuuksia vuosittaisen kuntotestausprosessin hoitamiseen:

- **Testattavat:** Tämä ryhmä koostuu käyttäjistä, jotka toimivat Keski-Suomen pelastuslaitoksen sivutoimisena pelastushenkilöstönä. Nämä käyttäjät käyttävät pääasiassa sovelluksen varausominaisuuksia. Heidän pääasialliset tarpeensa liittyvät sovelluksen helppokäyttöisyyteen, turvallisuuteen ja saavutettavuuteen. Testattavilla käyttäjille tulee olla mahdollisuus tehdä, tarkastella ja poistaa omia varauksia, päivittää omia tietojaan sekä vastaanottaa ja saada tietoa pääkäyttäjien tai esihenkilöiden lähettämistä ilmoituksista.

- **Testaajat:** Testaajan roolissa olevat käyttäjät toimivat osittain samoin kuin testattavat käyttäjät, mutta heillä tulee olemaan lisäoikeuksia. He voivat lisätä uusia käyttäjiä, seurata ja vastaanottaa kuntotestejä sekä merkitä omia poissaoloja. Tämä ryhmä koostuu pääasiassa pelastuslaitoksen virkahenkilöstöstä ja he käyttävät pääasiassa sovelluksen testien vastaanottoon ja hallintaan liittyviä ominaisuuksia.

- **Esihenkilöt:** Esihenkilö-roolin käyttäjät ovat pääkäyttäjien ohella vastuussa sovelluksen hallinnasta ja käyttäjäkunnan seurannasta. Tämän ryhmän tarpeet ovat monipuolisemmat ja liittyvät sovelluksen hallinnollisiin toimintoihin, kuten testausaikojen suunnitteluun testaajille, tilastojen tarkasteluun, ilmoitusten julkaisuun sekä käyttäjien lisäämiseen ja hallintaan. Tämä ryhmä tarvitsee tehokkaan ja turvallisen käyttöliittymän, joka tukee sovelluksen toimintojen hallintaa ja raportointia. Esihenkilö-käyttäjät ovat sidoksissa omiin pelastustoimen alueisiin ja hallitsevat oman alueensa paloasemaverkoston henkilöstön testausprosessia.

- **Pääkäyttäjät:** Pääkäyttäjät ovat pääasiallisessa vastuussa sovelluksen hallinnasta ja testausprosessin läpiviennistä. Toisin kuin esihenkilö-roolissa, pääkäyttäjien tulee nähdä ja kyetä hallitsemaan koko pelastuslaitoksen toimialueen henkilöstön testausprosessia. Tähän liittyy muun muassa mahdollisuus koko testaushenkilöstöön liittyvä testaustoiminnan työajan suunnittelusta. Pääkäyttäjällä tulee myös olla saatavilla koko toimialueen tilastot ja statistiikka.

3.3 Sidosryhmät

Sovelluksen sidosryhmät koostuvat eri tahoista, jotka vaikuttavat sen kehitykseen, käyttöönottoon ja käyttöön. Näiden sidosryhmien tarpeet ja odotukset tulee huomioida sovelluksen toteutuksessa, jotta sovelluksesta tulee toimiva ja tehokas ratkaisu kaikille osapuolille.

- **Käyttäjät**: Sovelluksen ensisijaisia sidosryhmiä ovat sen loppukäyttäjät, joihin kuuluvat kaikki sovellukseen rekisteröityneet käyttäjät eri rooleissa. Kukin käyttäjäryhmä hyödyntää sovellusta omalla tavallaan, ja heidän tarpeensa ohjaavat sovelluksen toiminnallisten ja laadullisten vaatimusten suunnittelua.

- **Pelastuslaitoksen johto**: Pelastuslaitoksen johto on vastuussa sovelluksen strategisesta ohjauksesta ja päätöksenteosta. Johdon tehtävänä on varmistaa, että sovellus tukee organisaation toiminnallisia tavoitteita, kuten tehokkuuden parantamista, kustannusten vähentämistä ja testauskokemuksen parantamista.

- **Pelastuslaitoksen viestiyksikkö**: Viestiyksikkö koostuu tieto- ja viestintäjärjestelmien asiantuntijoista ja asentajista. Viestiyksikön vastuulla on sovelluksen tekninen tuki ja ylläpito sekä laadunvarmistus sen jälkeen, kun sovellus siirtyy julkaisuvaiheeseen.

- **Kolmannen osapuolen palveluntarjoajat**: Sovelluksen julkaisualustana tulee toimimaan kolmannen osapuolen tarjoama palvelinhotelli, joka osaltaan vastaa sovelluksen tietoturvapalveluista. Nämä palveluntarjoajat ovat vastuussa tiettyjen teknisten osien toimivuudesta ja ylläpidosta, ja heidän luotettavuutensa on kriittistä sovelluksen toiminnan kannalta.

3.4 Toiminnalliset vaatimukset

Sovelluksen suunnitellut toiminnalliset vaatimukset on esiteltynä taulukossa 1 ja toiminnallisten vaatimusten CRUD-matriisi on esitelty taulukossa 2.

Taulukko 1. Toiminnalliset vaatimukset

Toiminnallisen vaatimuksen kuvaus	Toiminnallinen ominaisuus	Käyttäjäryhmät
Sovellukseen kirjautuminen käyttäjätunnuksella ja salasanalla	Kirjautumis-ominaisuus	Kaikki
Testiaikojen varaus	Ajanvaraus-ominaisuus	Testattavat
Omien tietojen katselu ja muokkaus	Tietojen hallinta-ominaisuus	Kaikki
Ilmoitusten kirjoittaminen	Ilmoitustaulu-ominaisuus	Esihenkilöt, Pääkäyttäjät
Ilmoitusten lukeminen	Ilmoitustaulu-ominaisuus	Kaikki
Käyttäjien lisääminen	Käyttäjä-ominaisuus	Pääkäyttäjät, Esihenkilöt, Testaajat
Testien vastaanotto	Testaus-ominaisuus	Testaajat
Poissaolon merkitseminen	Käyttäjä-ominaisuus	Pääkäyttäjät, Esihenkilöt, Testaajat
Testausaikojen suunnittelu	Työajan suunnittelu-ominaisuus	Pääkäyttäjät, Esihenkilöt
Tilastot ja statistiikka	Statistiikka-ominaisuus	Pääkäyttäjät, Esihenkilöt
Käyttäjähallinta	Käyttäjä-ominaisuus	Pääkäyttäjät, Esihenkilöt
Istunnonhallinta	Tietoturva-ominaisuus	Yleinen
Loki- ja tapahtumienhallinta	Tietoturva-ominaisuus	Yleinen

Taulukko 2. Toiminnallisuuksien CRUD-taulukko

Toiminnallinen ominaisuus	Testattavat	Testaajat	Esihenkilöt	Pääkäyttäjät
Testiaikojen varaus/poisto	CRUD	CRUD	CRUD	CRUD
Omien tietojen katselu ja muokkaus	RU	RU	RU	RU
Ilmoitukset kirjoittaminen/lukeminen	R	R	CRUD	CRUD
Käyttäjien lisääminen	-	C	C	C
Testien vastaanotto	-	CRUD	CRUD	CRUD
Poissaolon merkitseminen	-	CR	CRUD	CRUD
Testausaikojen suunnittelu	-	-	CRUD	CRUD
Tilastot ja statistiikka	-	-	R	R
Käyttäjähallinta	-	-	R	R

3.5 Laadulliset vaatimukset

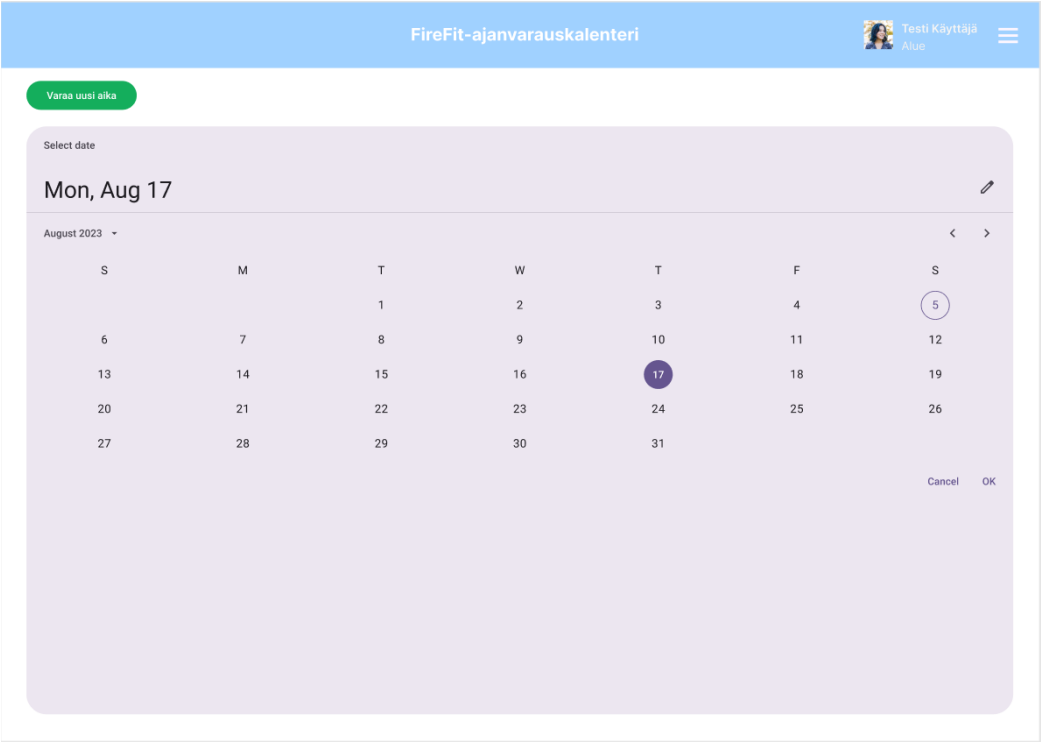
Sovelluksen laadulliset vaatimukset on esitelty taulukossa 3.

Taulukko 3. Laadulliset vaatimukset

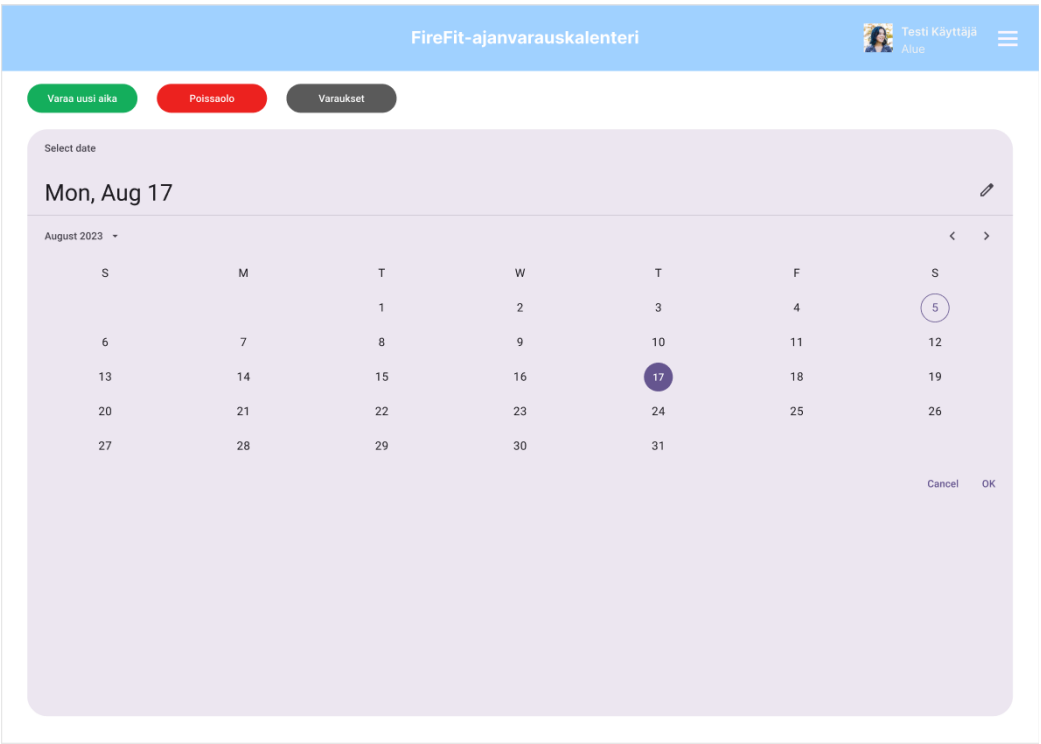
Laadullisen vaatimuksen kuvaus	Laadullinen ominaisuus	Käyttäjäryhmät
Sovelluksen sujuva toiminta eri päätelaitteilla	Skaalautuvuus	Kaikki
Helppokäyttöinen ja intuitiivinen käyttöliittymä	Käytettävyys	Kaikki
Nopea varausjärjestelmä	Suorituskyky	Testattavat, Testaajat
Sovelluksen tietoturva ja tietosuoja	Tietoturva	Kaikki
Käyttäjätietojen turvallinen käsittely ja säilytys	Tietosuoja	Kaikki
Vikasietoisuus ja jatkuva saatavuus	Luotettavuus	Kaikki
Järjestelmän ylläpidon helppous	Ylläpidettävyys	Pääkäyttäjät, Esihenkilöt
Tiedon ajantasaisuus ja synkronointi	Reaaliaikaisuus	Kaikki
Joustava tietojen päivitys	Joustavuus	Kaikki
GDPR:n ja muiden lainsäädäntöjen noudattaminen	Lakien noudattaminen	Kaikki

3.6 Käyttöliittymäkuvaus

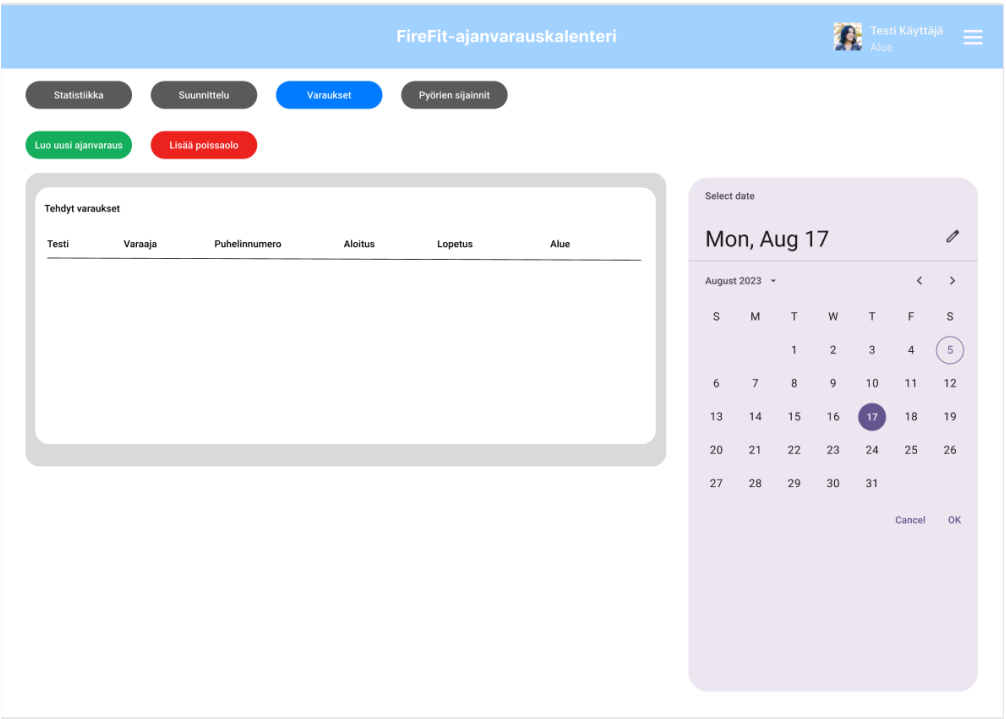
Käyttöliittymän suunnitellut mockup-kuvat on esitelty alla kuvioissa 2, 3, 4, 5 ja 6.



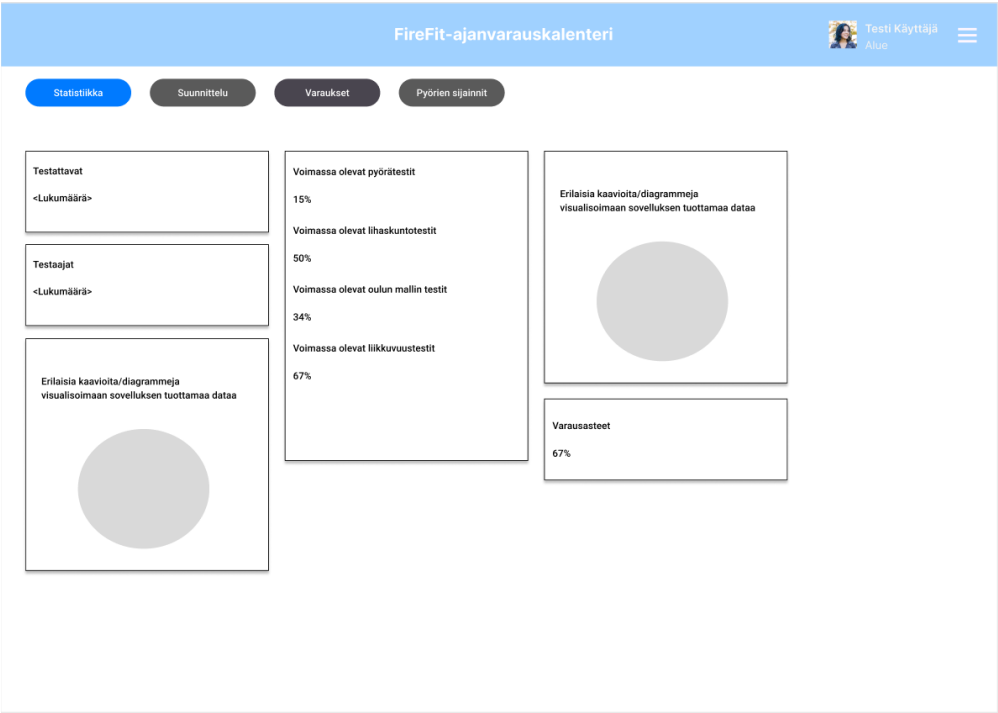
Kuvio 2. Suunniteltu testattava-roolin käyttöliittymä



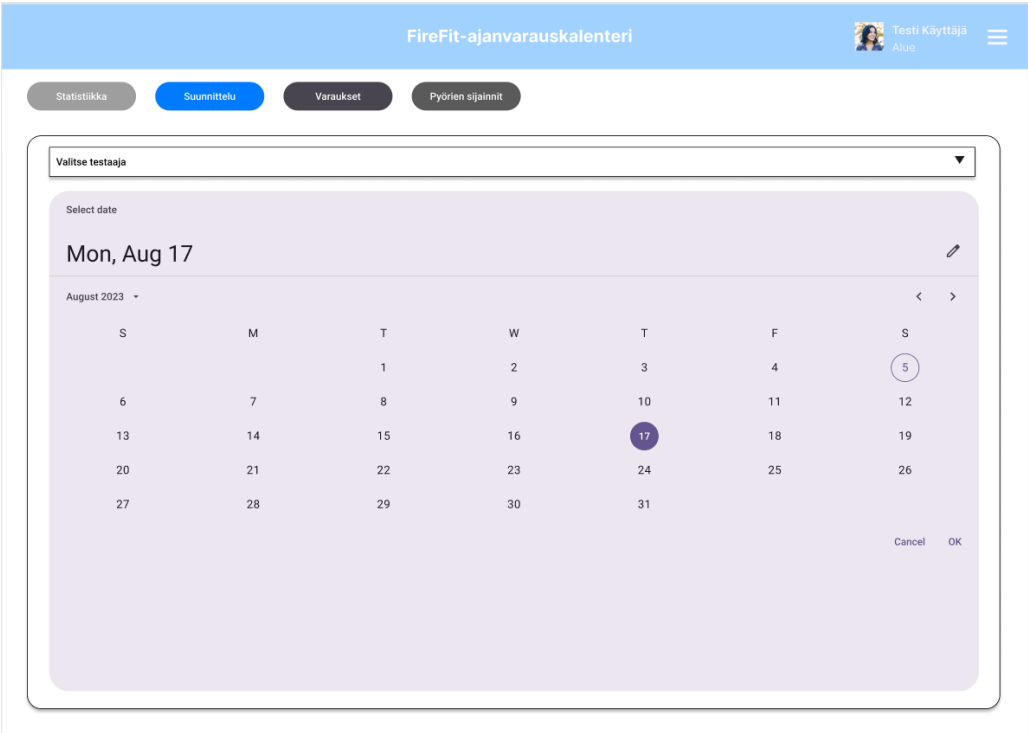
Kuvio 3. Suunniteltu testaaja-roolin käyttöliittymä



Kuvio 4. Suunniteltu esihenkilö- ja pääkäyttäjä-roolien käyttöliittymän Varaukset-välilehti



Kuvio 5. Esihenkilö- ja pääkäyttäjä-näkymän Statistiikka-välilehden suunnitelma



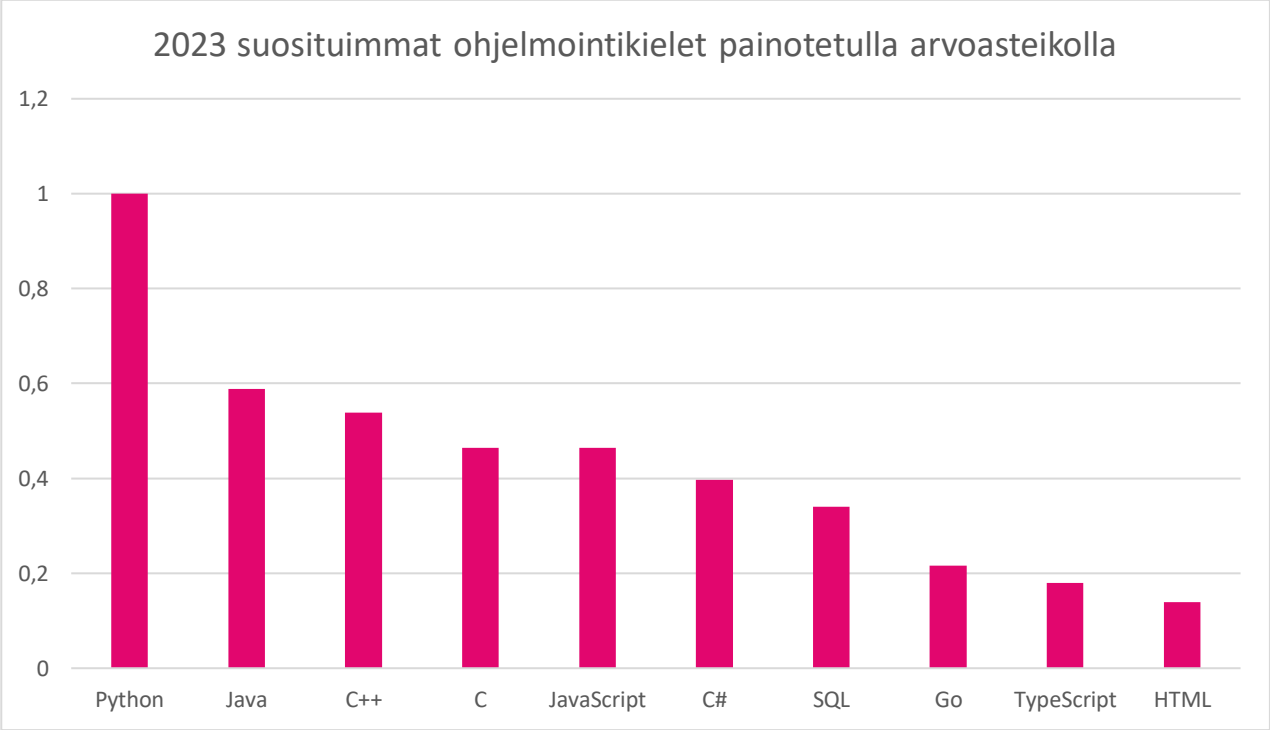
Kuvio 6. Esihenkilö- ja pääkäyttäjänäkymän Suunnittelu-välilehden suunnitelma

4 Verkkosovellusten arkkitehtuuri

4.1 Ohjelmointikielet

Erilaisia verkkokehitykseen soveltuvia ohjelmointikieliä on lukuisia ja lähes poikkeuksetta maailmalla käytetyimmät ohjelmointikielet soveltuvat niin ikään web-kehitykseen. Kansainvälinen tekniikan alan järjestö Institute of Electrical and Electronics Engineers (myöhemmin IEEE), julkaisee vuosittain listauksen sen hetkisistä suosituimmista ohjelmointikielistä. (Lindström 2020.)

Vaikka listaus- ja arviointimenetelmät kehittyvät vuosien saatossa, valtavirran käyttämät kielet pitävät pääasiassa sijoituksistaan tiukasti kiinni (Cass 2023). Kuviossa 7 on esitelty kymmenen suosituinta ohjelmointikieltä vuodelta 2023.



Kuvio 7. 10 suosituinta ohjelmointikieltä 2023 (Cass 2023).

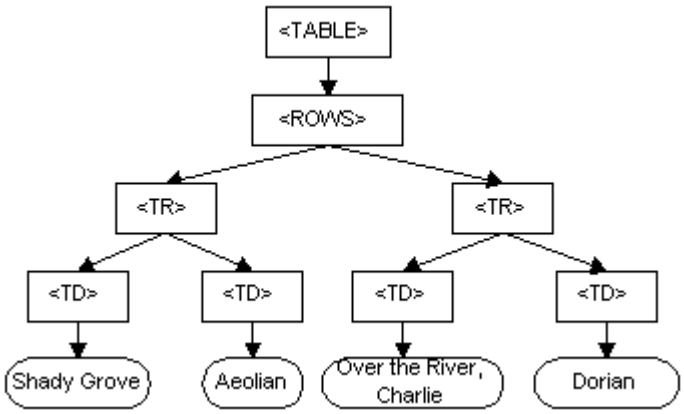
Listauksen kärkisijaa pitää sinnikkäästi ja kasvavalla marginaalilla ohjelmointikieli Python. Suoraan verkkosovellusten kehitykseen suunniteltu JavaScript ja sen staattisia muuttujatyyppejä valvova laajennus, TypeScript, löytyvät myös kymmenen suosituimman ohjelmointikielen listaukselta. (Austin 2023; Cass 2023.)

4.1.1 JavaScript

JavaScript on laajalti verkkosovellusten kehitykseen käytetty moniparadigmainen ohjelmointikieli, jonka tarkoitus on luoda responsiivisuutta sovelluksen ja käyttäjän välille. JavaScript ei ole käyttöjärjestelmäsidonnainen, vaan se pystyy toimimaan useilla käyttöjärjestelmillä ja alustoilla. JavaScript julkaistiin vuonna 1995 ja Marichevan (2023) mukaan 65 % sovelluskehittäjistä on maailmanlaajuisesti käyttänyt JavaScript-ohjelmointikieltä vuonna 2023. (Maricheva 2023.)

JavaScriptin pääasiallinen tarkoitus on luoda ja muokata verkkosovellusten sisältöä dynaamisesti ilman, että käyttäjän tarvitsee päivittää verkkosivustoa. Tällaisia ominaisuuksia ovat esimerkiksi animaatiot, interaktiiviset lomakkeet tai valokuvaesitykset. Jos vierailemasi verkkosivusto päivittää jotain ilman käyttäjän toimia, kyse on mitä todennäköisimmin JavaScriptistä. (Morris N.d.)

Tapa, jolla JavaScript muokkaa verkkosovellusten sisältöä perustuu DOM-puun (Document Object Model) muokkaamiseen sovelluksen käytön aikana. DOM-puu on yksinkertaisimmillaan puumainen rakenne, jonka oksille jokainen sovelluksessa käytetty komponentti asettuu loogiseen järjestykseen. JavaScript kykenee muokkaamaan näitä puun haaroja ja oksia, poistamaan osia tai päivittämään niiden sisältöä tai sijaintia. (Robie N.d.) Kuviossa 8 on esiteltynä yksinkertainen DOM-puu.



Kuvio 8. DOM-puu (Robie N.d).

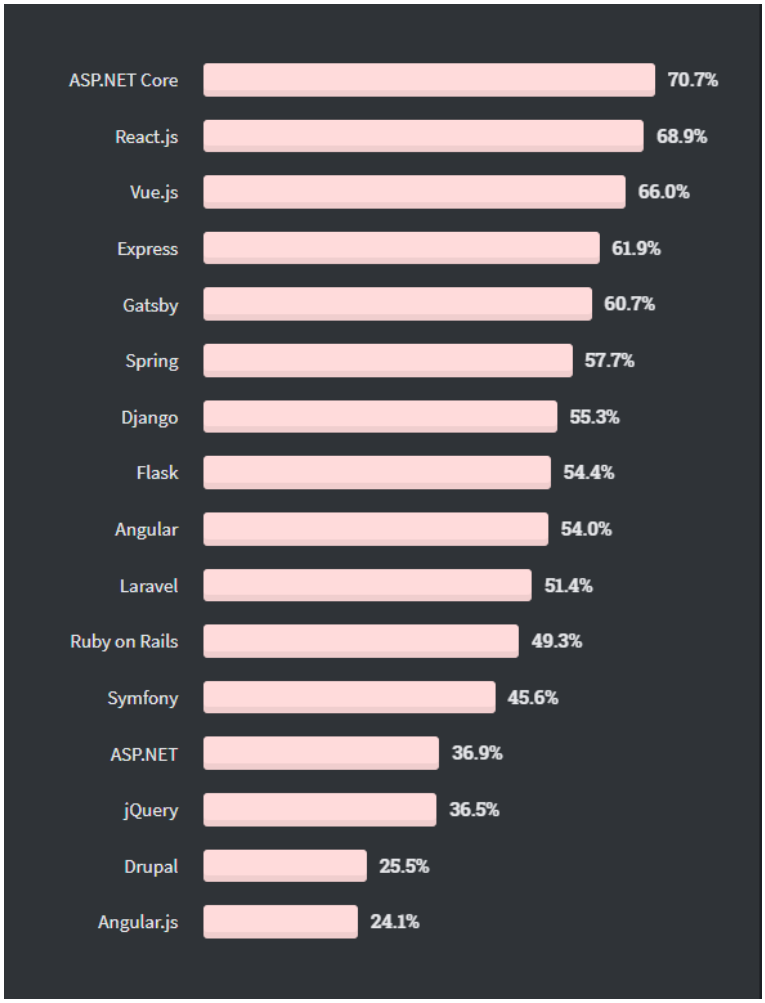
Ensimmäisen version julkaisun jälkeen 1995, JavaScriptiä on päivitetty useita kertoja ja sen toimintaa on parannettu huomattavasti (History of JavaScript 2024). JavaScript on yksisäikeinen ohjelmointikieli, joka tarkoittaa sitä, että se pystyy suorittamaan vain yhden tehtävän kerralla. Tästä

syystä uusimmissa JavaScript versioissa yksisäikeisyyden pystyy ohittamaan luomalla asynkronisia tehtäviä eli käytännössä laittamaan suorituksessa olevan tehtävän odottamaan, kunnes haluttu lopputulos tai haettu tieto on toimitettu. (Long 2023.)

JavaScriptin alkuaikoina sen käyttötarkoitus oli luoda interaktiivisuutta sovelluksen ja käyttäjän välille. Nykyään JavaScriptiä käytetään myös moneen muuhun tehtävään, esimerkiksi palvelinohjelmointiin. (History of JavaScript 2024.) Vuonna 2009 mies nimeltä Ryan Dahl loi ensimmäisen version JavaScript-ajoympäristöstä nimeltä Node.js. Hän halusi luoda kevyen ja nopean ympäristön suorittamaan JavaScript koodia ja Node.js otettiinkin kehitysyhteisössä vastaan suurella innokkuudella, koska se mahdollisti JavaScript koodin ajamisen verkkoselaimen ulkopuolella. (Uzoma 2023.)

Kuten nykyään lähes kaikille ohjelmointikielille, myös JavaScriptille on tarjolla lukuisia erilaisia ohjelmointikehyksiä, joiden tehtävänä on yksinkertaistaa, nopeuttaa sekä parantaa kehitysprosessia. Ohjelmointikehys on käytännössä ladattava sovellus, jonka sisään uusi sovellus luodaan. Kehykseen voidaan sisällyttää JavaScript koodia ja erilaisia sääntöjä tai ohjeita. Ohjelmointikehyksen voi ajatella olevan yksittäinen auto, jonka moottorina toimii JavaScript ajoympäristö. Erilaiset ohjelmointikehykset ovat eri värisiä tai näköisiä autoja, mutta samalla JavaScript-moottorilla. (What is A JavaScript Framework 2024.)

Seuraavassa kuviossa 9 on listattuna StackOverflow-verkkosivuston tuottaman 2020 Developer Survey (2020) mukaan maailmanlaajuisesti suosituimmat verkko-ohjelmointikehykset. Lähes poikkeuksetta listattuna olevat ohjelmointikehykset käyttävät JavaScript-ohjelmointikieltä pohjanaan.



Kuvio 9. Käytetyimmät sovelluskehykset 2020 (2020 Developer Survey 2020).

4.1.2 TypeScript

TypeScript on Microsoftin vuonna 2012 julkaisema, pääasiassa verkkosovelluksiin käytettävä ohjelmointikieli, joka tunnetaan sen staattisesta tyypityksestä. Tämä tarkoittaa, että kehittäjällä on mahdollisuus määritellä ohjelmistossa liikkuvien muuttujien, funktioiden ja olioiden tyypit etukäteen, mikä mahdollistaa virheiden löytämisen jo ohjelman käännösvaiheessa. TypeScript on JavaScript -ohjelmointikielen laajennus, mikä tarkoittaa, että kaikki voimassa oleva JavaScript-koodi on myös kelvollista TypeScript-koodia. (Chen 2024.)

TypeScript-ohjelmointikielen staattisen tyypityksen etuna on se, että kun jonkin muuttujan tyyppi on julistettu, siihen ei voi tallentaa minkään muun tyypin tietoa kuin julistetun tyypin. Tämä tarkoittaa siis sitä, että jos muuttujan X tyypiksi on määritelty merkkijono, kyseiseen muuttujaan ei voi tallentaa numeroita tai muita muuttujia, kuin merkkijonoja. (Chen 2024.)

Kuten JavaScriptiä, myös TypeScriptiä voidaan käyttää ja soveltaa erilaisilla ohjelmointikehyksillä. StackOverflow-verkkosivuston tuottaman 2020 Developer Surveyn (2020) mukaan (ks. kuvio 3) puhtaasti verkkosovellusten käyttöliittymän kehittämiseen käytettävä React.js on listauksen sijalla 2 ja aiemmin mainitut Django- ja Flask-ohjelmointikehykset hieman alempana, sijoilla 6 ja 7.

4.1.3 Python

Python on monipuolinen ja suosittu korkean tason ohjelmointikieli, jota käytetään laajasti eri sovelluksissa, kuten verkkokehityksessä, datan analysoinnissa, koneoppimisessa sekä automaatiossa. Pythonin selkeä syntaksi ja helppokäyttöisyys tekevät siitä hyvän valinnan sekä aloittelijoille, että kokeneille ohjelmoijille. Kieli tukee laajaa valikoimaa kirjastoja ja työkaluja, jotka helpottavat ohjelmistojen kehittämistä ja ratkaisujen luomista monimutkaisiin ongelmiin. (What Is Python Used For? A Beginner’s Guide 2024.)

Pythonin kehitys alkoi vuonna 1989 ja sitä alkoi kehittämään hollantilainen ohjelmoija Guido van Rossum, joka oli urallaan aiemmin ollut mukana kehittämässä ABC ohjelmointikieltä. ABC:lla oli ominaisuus, josta myöhemmin muodostui Pythonin yksi keskeisimmistä ominaisuuksista: poikkeusten käsittely. (History of Python 2024; Python History and Versions N.d.)

Verkkosovelluksien kehityksessä Pythonia käytetään monesti palvelimen ohjelmointiin. Varsinkin jos haluaa sovelluksen hyödyntävän esimerkiksi koneoppimisen ja tekoälyn algoritmiikkaa, on Python tehtävään hyvin soveltuva ohjelmointikieli. Pythonin valintaa palvelimen ohjelmointiin tukee myös erittäin laaja ohjelmointikirjastojen määrä sekä aktiivinen kehitysyhteisö. Käytetyimmät verkkokehitykseen soveltuvat ohjelmointikehykset Pythonilla ovat Django ja Flask. (Talekar 2023; Pandya 2024.)

4.2 Palvelin ja ajoympäristöt

4.2.1 Node.js

Node.js on avoimeen lähdekoodiin perustuva ajoympäristö JavaScript -ohjelmointikielelle. Ajoympäristö on kuin tulkki, jonka tarkoituksena on kääntää ohjelmakoodi tietokoneen ymmärtämälle konekielelle, jonka prosessori voi suorittaa. Kun käytät nykyaikaista verkkosovellusta, lähes kaikki vuorovaikutus luo pyyntöjä sovelluksen palvelimelle joihin palvelin tarjoaa jonkinlaista vastausta.

(Sufiyan 2024b). Nämä pyynnöt ja vastaukset voivat välittää tietoja, esimerkiksi kirjautumistietoja, palvelimelle, joka käsittelee pyynnöt ja sen mukana kulkevan tiedon ja palauttaa vastauksen sovelluksen käyttöliittymälle. Pyynnön tarkoituksen mukaan palvelimen on myös todennäköisesti tehtävä erilaisia laskutoimituksia tai tiedonhakua esimerkiksi tietokantaan, ennen pyyntöön vastaamista. (Sharma 2023.)

Ajoympäristö siis mahdollistaa edellä mainitun pyyntö-vastausmallin toiminnan. Se on kuin malja, jonka sisään sovelluksen eri osat rakentuvat ja joka yhdistää sovelluksen ja tietokoneen yhteen mahdollistaen laskutoimitukset ja yhteydet tietolähteisiin, esimerkiksi tietokantoihin. (Sufiyan 2024b.)

Yksi Node.js:n suurimmistä hyödyistä verkkosovellusten kehityksessä on sen käyttämä kieli eli JavaScript, mikä tarkoittaa, että ohjelmistokehittäjät pystyvät luomaan koko sovelluksen yhdellä ainoalla kielellä. Tämä luo saumatonta yhteensopivuutta ja helppokäyttöisyyttä sovelluksen eri osien välille. Kuten Boruń (2023) kertookin, saman ohjelmointikielen käyttö sekä käyttöliittymässä, että palvelimella luo ennen kaikkea tehokkuutta, koska voit hyödyntää jo kirjoittamaasi koodia sovelluksen eri osissa uudelleen. (Boruń 2023.)

Koska Node.js on maailmanlaajuisesti käytetty ajoympäristö, on sille muodostunut myös valtava ekosysteemi erilaisia moduuleja ja kirjastoja. Moduulit ja kirjastot ovat kuin lisäosia Node.js ajoympäristölle, jotka tuovat lisää toiminnallisuutta ja valmiiksi kirjoitettua koodia valmiina käytettäväksi. (Askarov 2023.)

Moduulit

Moduulit ovat yksittäisiä itsenäisiä funktionaalisia yksiköitä, joiden tehtävänä on suorittaa jokin laskutoimitus tai toimenpide. Moduuli voi olla pienimmillään muutaman koodirivin mittainen yksittäinen tiedosto tai laaja kokoelma erilaisia funktioita ja tiedostoja. Moduulien tarkoitus on luoda abstraktiota ohjelmakoodiin ja näin ollen hajottaa koodi paremmin hahmoteltaviin kokonaisuuksiin, jolla vältetään saman koodin uudelleen kirjoittaminen. (Node.js Modules 2024; Kinsman 2022.) Kuviossa 10 on esitelty yksinkertainen Node.js moduuli ja sen käyttö.

```
// sumModule
function sum(a, b) {
    return a + b;
}
module.exports = { sum };

// Usage
const { sum } = require('./sumModule')

const a = 1;
const b = 2;
const result = sum(a,b);
```

Kuvio 10. Esimerkki Node.js moduulista ja sen käytöstä

Yllä olevaa esimerkki-moduulia *sumModule* ja sen sisältämää funktiota *sum()* voidaan nyt siis käyttää kaikkialle sovelluksessa laskemaan kahden arvon yhteenlasku ilman, että laskutoimitusta tarvitsee kirjoittaa uudelleen.

Kirjastot

Node.js kirjastot toimivat lähes samoin kuin moduulit, mutta luovat jälleen uuden kerroksen abstraktiota Node.js ajoympäristöön. Voidaankin ajatella, että ohjelmiston kehittäjä on vuorovaikutuksessa kirjaston kanssa ja kirjasto käyttää, ja tuo kehittäjän käytettäväksi, moduuleja tuottamaan halutun lopputuloksen. Yksittäisellä kirjastolla on yleensä jokin käyttötarkoitus luomaan jokin osa sovelluksesta. Tällainen käyttötarkoitus voisi olla vaikkapa palvelinyhteyden luominen pyyntö-vastausmallin mukaisesti. (Sufiyan 2024b.) Kuviossa 11 on esiteltynä yksinkertainen Express -kirjastolla luotu verkkosovelluksen palvelinmalli yhdellä reittipisteellä, johon voi tehdä pyyntöjä. Kuviossa 12 on esitelty saman toiminnallisuuden omaava palvelinmalli ilman Express-kirjastoa ja kuviossa 13 on esitelty hypoteettinen ja fiktiivinen pseudokoodi samasta toiminnallisuudesta ilman minkäänlaisia kirjastoja tai moduuleja.

```
const express = require('express');
const app = express();

app.get('/', (req, res) => {
    res.send("Terveppä terve!");
});

const PORT = 3000;
app.listen(PORT, () => {
    console.log(`Palvelin kuuntelee pyyntöjä osoitteessa http://localhost:${PORT}`);
})
```

Kuvio 11. Palvelinmalli Express-kirjastolla

```
const http = require('http');
const url = require('url');

const server = http.createServer((req, res) => {
    const parsedUrl = url.parse(req.url, true);

    res.setHeader('Content-Type', 'text/plain');

    if (parsedUrl.pathname === '/' && req.method === 'GET') {
        res.statusCode = 200;
        res.end("Terveppä terve!");
    } else {
        res.statusCode = 404;
        res.end("Sivua ei löytynyt");
    }
});

const PORT = 3000;
server.listen(PORT, () => {
    console.log(`Palvelin kuuntelee pyyntöjä osoitteessa http://localhost:${PORT}`);
});
```

Kuvio 12. Palvelinmalli ilman Express-kirjastoa käyttäen moduuleja

```javascript
function acceptConnection() {
    console.log("Simulating acceptance of a new connection...");
    return {
        connectionId: Math.random().toString(36).substring(7),
        dataBuffer: "",
    };
}
function receiveRequest(connection) {
    console.log(`Simulating data reception on connection ${connection.connectionId}...`);
    connection.dataBuffer = "GET / HTTP/1.1\r\nHost: localhost\r\n\r\n";
    return connection.dataBuffer;
}
function sendResponse(connection, response) {
    console.log(`Sending response on connection ${connection.connectionId}...`);
    console.log("Response sent:\n" + response);
}
function closeConnection(connection) {
    console.log(`Closing connection ${connection.connectionId}...`);
    console.log(`Connection ${connection.connectionId} closed.`);
}
function handleRequest(request) {
    const [method, path] = parseRequest(request);
    let response = '';
    if (method === 'GET' && path === '/') {
        response = `HTTP/1.1 200 OK\r\nContent-Type: text/plain\r\n\r\nTerveppä terve!`;
    } else {
        response = `HTTP/1.1 404 Not Found\r\nContent-Type: text/plain\r\n\r\nSivua ei löytynyt`;
    }
    return response;
}
function parseRequest(request) {
    const requestLine = request.split('\n')[0];
    const [method, path] = requestLine.split(' ');
    return [method, path];
}
function createServer(callback) {
    console.log("Simulating a server listening on port 3000...");
    while (true) {
        const connection = acceptConnection();
        const request = receiveRequest(connection);
        const response = callback(request);
        sendResponse(connection, response);
        closeConnection(connection);

        break;
    }
}
createServer(handleRequest);
```

Kuvio 13. Hypoteettinen palvelinmalli ilman moduuleja tai kirjastoja

Edellä olevista kuvioista huomataan, että mitä vähemmän abstraktiota sovellukseen tuodaan, sitä monimutkaisemmaksi ohjelma muodostuu ja sitä työläämpää sen hallitseminen on.

4.2.2 CPython

Aivan kuten JavaScript, myös Python ohjelmointikieli tulee muuntaa matalan tason konekieleksi, jotta tietokoneiden prosessorit pystyvät sitä käsittelemään. Siihen tarkoitukseen Pythonin osalta käytetään yleisesti virallista CPython-ajoympäristöä. CPythonin ensimmäinen versio julkaistiin vuonna 1994 ja kuten silloin, myös tänä päivänä, CPython on kirjoitettu C-kielellä. (Shaw n.d; Ahuja 2024.) C-kieli on yksi vanhimmista ohjelmointikielistä ja se toimii pohjana usealle uudemmalle ja myöhemmin luodulle ohjelmointikielelle (C Programming Basics Explained 2024).

CPython ei muunna ohjelmakieltä suoraan konekieleksi kuten Node.js vaan se käyttää niin kutsuttua välitasoa, muuttaen ohjelmakielen ensin tavukoodiksi, joka syötetään prosessorille konekielenä käyttäen Python virtuaalikonetta PVM. Tämä tuo esiin sekä hyötyjä, että haittoja. Ensinnäkin ylimääräinen vaihe ohjelman ajossa lisää luonnollisesti suorituksen kestoa, joka voi olla hyvinkin tärkeässä roolissa verkkosovelluksissa, joissa aikaa voidaan pitää rajoittavana tekijänä. Toisekseen CPythonin rajoite verrattuna Node.js:ään on tapa, jolla CPython ajaa ohjelmaa: rivi kerrallaan. (Shaw n.d; Ahuja 2024; Kareliya 2024.)

Myös CPythonille on saatavilla laaja kattaus erilaisia moduuleja ja kirjastoja, jotka tuovat abstraktiota ohjelmointiin. Python onkin erittäin tunnettu kirjastoistaan, jotka keskittyvät koneoppimisen ja vaativien matemaattisten tehtävien suorittamiseen. Python-kirjastojen toiminta ja käyttö on hyvin samankaltaista kuin Node.js:llä. (Libraries in Python 2024.)

Vaikka CPython on erittäin kyvykäs verkkosovellusten palvelinarkkitehtuurissa, nykyajan voimakkaasti interaktiiviset ja käyttäjälle palautetta jatkuvasti tuottavat verkkosivustot vaativat korkean skaalautuvuuden palvelinarkkitehtuuria ja siihen Node.js on luonnollisempi vaihtoehto. (Kareliya 2024.) Mainittakoon vielä, että yhdistämällä nämä kaksi ajoympäristöä, voidaan luoda sekä korkean skaalautuvuuden, että suorituskyvyn omaavia palvelimia, jotka kykenevät suorittamaan tehtävän kuin tehtävän (Rao 2023).

4.3 Sovelluskehykset

4.3.1 React.js (Node.js)

React.js on Facebookin vuonna 2011 kehittämä JavaScript-pohjainen ohjelmointikirjasto, joka keskittyy käyttöliittymien ohjelmointiin. React.js on komponenttipohjainen kirjasto, joka ratkaisee uudelleenkäytettävyyden ongelman käyttöliittymien ohjelmoinnissa ja kykenee muodostamaan käyttöliittymistä responsiivisiä ja käyttäjäystävällisiä. (Deshpande 2024.)

Hieman samoin kuin Node.js moduulit, myös React.js komponentit ovat valmiita ladattavia, tai itse luotavia, paketteja, joiden avulla vältetään koodin uudelleen kirjoittaminen. Tässä tapauksessa komponentti voi olla vaikkapa yksittäinen visuaalisesti tyylitelty tekstikenttä, jonne käyttäjä voi

käyttöliittymässä kirjoittaa. Paketin tekijä varmistaa, että kaikki komponentin tarvitsemat moduulit ja muut komponentit ovat paketissa mukana ennen kuin paketin voi ladata sovelluksen käyttöön. Latauksen jälkeen komponenttia voidaan kutsua ohjelmakoodissa sen nimellä, jolloin käyttöliittymään piirtyy ladattu tekstikenttä. (Deshpande 2024.) Kuviossa 8 on esitelty yksinkertainen React.js komponentti ja sen käyttö sovelluksessa.

React.js:n ydintoiminta perustuu tilanhallintaan, jossa yksittäisen komponentin sisällä käytettävän muuttujan tilaa pystytään hallinnoimaan komponentin elinaikana. Tähän muuttujaan ja sen tilaan voidaan tallentaa mitä tahansa komponentin kannalta oleellista tietoa ja sitä pystytään muokkaamaan ja lähettämään esimerkiksi palvelimelle käsiteltäväksi. Yleisesti komponenttien tilaa muutetaan silloin kun käyttäjä on vuorovaikutuksessa komponentin kanssa. Esimerkiksi kun tekstikenttäkomponenttiin kirjoitetaan jotain, voidaan käyttäjälle näyttää toisessa komponentissa hänen kirjoittama teksti. (Sufiyan 2024a.)

Toinen React.js:n tärkeistä kulmakivistä on komponenttien ominaisuudet tai lyhyemmin propsit, joiden avulla komponenttien välillä voidaan välittää tietoa. React.js käyttää niin kutsuttua yksisuuntaista tietovirtaa eli sisäkkäin luoduille komponenteille voidaan valuttaa tietoa korkeamman tason komponenteista ja tarvittaessa muokata sitä matkan varrella. Tämä on tärkeää uudelleenkäytettävyyden näkökulmasta, jolloin esimerkiksi tietokantapyyntöjä tarvitsee tehdä vain kerran ja saman tiedon pystyy välittämään kaikille komponenteille, jotka tietoa tarvitsevat. (Sufiyan 2024a.) Kuviossa 15 on esiteltynä *label*-ominaisuuden välittäminen kuvion 14 tekstikomponentille, joka muuttaa tekstikomponentin niin kutsuttua *placeholder* -arvoa.

```
// Komponentti
import React, { useState } from 'react';

function TextInput() {
    const [inputValue, setInputValue] = useState('');
    const handleInputChange = (event) => {
        setInputValue(event.target.value);
    };
    return (
        <div>
            <label htmlFor="inputField">Kirjoita jotain: </label>
            <input
                id="inputField"
                type='text'
                value={inputValue}
                onChange={handleInputChange}
            />
        </div>
    );
};

export default TextInput;

// Komponentin käyttö
import React from 'react';
import TextInput from './TextInput';

function App() {
    return (
        <div className='App'>
            <TextInput />
        </div>
    )
}
```

Kuvio 14. React.js komponentti ja sen käyttö

```
import React, { useState } from 'react';

function TextInput({ label }) {
    const [inputValue, setInputValue] = useState('');

    const handleInputChange = (event) => {
        setInputValue(event.target.value);
    };

    return (
        <div>
            <label htmlFor="inputField">{label}</label>
            <input
                id="inputField"
                type='text'
                value={inputValue}
                onChange={handleInputChange}
            />
        </div>
    );
};

export default TextInput;

import React from 'react';
import TextInput from './TextInput';

function App() {
    return (
        <div className='App'>
            <TextInput label="Kirjoita nimesi: " />
            <TextInput label="Kirjoita sähköpostiosoitteesi: " />
        </div>
    );
}
```

Kuvio 15. Ominaisuuden välittäminen komponentille

On huomioitavaa, että kuvion 15 esimerkissä käytetään samaa komponenttia kahdesti eri *label*-arvolla. React.js:ssä tämä on mahdollista, koska jokaisen ohjelmaan kutsutun ja ladatun komponentin elinkaari alkaa ja päättyy niiden omassa instanssissa eli omassa erillisessä tilassaan, mikä mahdollistaa samojen komponenttien riippumattoman toiminnan ja tilan- sekä ominaisuuksien muutokset. (Thulo 2023.)

4.3.2 Django (Python)

Django on kokonaisvaltainen kehys verkkosovelluksen kehitykseen, joka perustuu MVC (Model-View-Controller) -arkkitehtuurimalliin. MVC-arkkitehtuuri jakaa sovelluksen kolmeen pääkomponenttiin: malliin (Model), näkymään (View) ja kontrolleriin (Controller), jonka tarkoituksena on eriyttää sovelluksen eri osat toisistaan, mikä helpottaa ylläpitoa, testaamista ja laajentamista sekä mahdollistaa hyvin nopean kehityksen ja prototyypin luomisen. (Spain 2022.)

4.3.3 Flask (Python)

Flask on kevyt ja joustava niin kutsuttu mikrokehys, mikä tarkoittaa, että se tarjoaa vain olennaiset työkalut verkkosovellusten kehittämiseen ja jättää monimutkaiset ominaisuudet ja laajennokset kehittäjän päätettäväksi. Flaskin etuna on sen kyky mukautua useisiin erilaisiin järjestelmiin ja ohjelmistoihin sekä koodin luettavuus. (What is Flask Python N.d.)

4.4 Tietoturvallisuus verkkosovelluksissa

4.4.1 Tietoturva

Tietoturvallisuus on käsite, joka kattaa kaikki prosessit, työkalut ja menetelmät, joita käytetään tai voidaan käyttää tietojärjestelmien, sovellusten tai tietovarastojen ja niiden sisältämien tietojen turvaamiseksi. Tietoturvallisuus käsittää kaikki turvaamisen muodot kuten fyysiset turvaamistoimet eli kulkurajoitetut tilat tai vartijat sekä muun muassa kyberturvallisuuden, joka on metodologia suojata verkossa toimivia järjestelmiä samaan tapaan kuin fyysiset turvaamistoimet. Tietoturvallisuuden päätavoitteena on taata tiedon luotettavuus, eheys ja saatavuus kaikille niille osapuolille, jotka tietoa tarvitsevat. (Holdsworth & Kosinski 2024.)

Open Worldwide Application Security Project eli OWASP on voittoa tavoittelematon verkkoyhteisö, joka tuottaa vapaasti saatavilla olevia dokumentteja, neuvoja ja ohjeita verkossa toimivien sovellusten ja tietojärjestelmien kyberturvallisuuteen. Vuonna 2021 OWASP julkaisi OWASP Top Ten -verkkojulkaisun, johon on analysoitu ja listattu kymmenen yleisintä turvallisuusriskiä liittyen verkkosovelluksiin. (About the OWASP Foundation n.d; OWASP Top Ten 2011.) Kuviossa 16 on esitelty vuoden 2021 OWAS Top Ten.

2021

A01:2021-Broken Access Control
A02:2021-Cryptographic Failures
A03:2021-Injection
A04:2021-Insecure Design
A05:2021-Security Misconfiguration
A06:2021-Vulnerable and Outdated Components
A07:2021-Identification and Authentication Failures
A08:2021-Software and Data Integrity Failures
A09:2021-Security Logging and Monitoring Failures*
A10:2021-Server-Side Request Forgery (SSRF)*

* From the Survey

Kuvio 16. OWASP Top Ten 2021 (OWASP Top Ten 2011).

4.4.2 Tietosuoja

Tietosuojalla tarkoitetaan rekisteröidyn yksityisyyden suojaa kaikissa tietojärjestelmissä, sovelluksissa tai tietokannoissa, joita käytetään rekisteröidyn tietojen säilyttämiseen. EU:n yleinen tietosuoja-asetus GDPR, Suomen perustuslaki (L 11.6.1991/731) sekä muun muassa tietosuojalaki (L 5.12.2018/1050) ja henkilötietolaki määrittelevät tiukat rajat rekisteröityjen henkilöiden tietojen käsittelyyn ja tallentamiseen. (Muurinen 2019.)

Erityiset henkilötietoryhmät

Sen lisäksi, että tietojärjestelmiin tallennetaan henkilötietoja, sinne voidaan tallentaa myös erityisiä henkilötietoja. Erityisistä henkilötiedoista on kyse silloin kun näistä tiedoista ilmenee esimerkiksi rekisteröidyn terveyttä koskevia tietoja. (Erityisten henkilötietoryhmien käsittely n.d.)

Erityisten henkilötietoryhmien käsittelyä rajataan EU:n tietosuoja-asetuksessa muita henkilötietoja tiukemmin ja niiden käsittelyssä tuleekin aina huomioida asetuksen soveltamisala. Jokainen sovellus tai tietojärjestelmä ei saa käsitellä näitä henkilötietoryhmiä, vaan niiden käsittelyn täytyy täyttää sekä EU:n tietosuoja-asetuksen määräykset, että kansallisten lakien ja asetusten määräykset. Tallennettaessa erityisiä henkilötietoja, tulee ne aina salata ja niitä käsittelevällä henkilöllä tulee olla lakisääteinen vaitiolovelvollisuus. (Erityisten henkilötietoryhmien käsittely n.d; L 523/1999.)

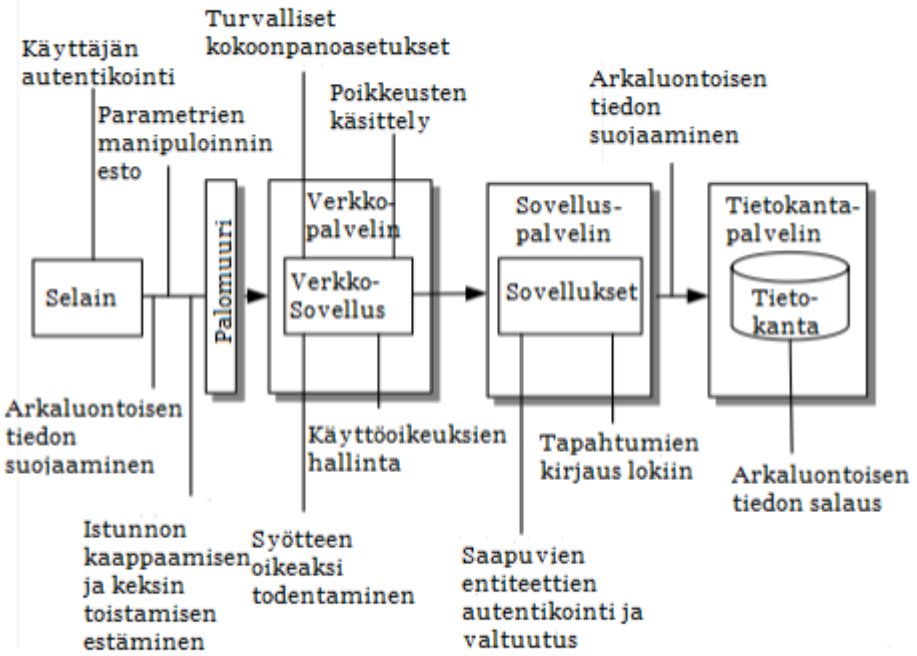
4.4.3 Tietoturvan työkalut verkkosovelluksissa

Erilaisia työkaluja ja tekniikoita toteuttaa tietojärjestelmien tietoturvaa on monia ja useat niistä ovatkin tarkoitettu yksittäisen tiedon suojaamiseen tai mahdollisen tietojärjestelmään kohdistuvan uniikin tietoja muuttavan tai varastavan tapahtuman estämiseen. (Holdsworth & Kosinski 2024.)

Helpoimmin tällaiseksi tapahtumaksi mielletään esimerkiksi kyberhyökkäykset, joissa hyökkääjä pyrkii esimerkiksi kaatamaan tietojärjestelmän ja näin vaikuttamaan tiedon saavutettavuuteen, mutta uhkia voi esiintyä myös organisaation sisällä. Näistä merkittävimpiä ovat muun muassa käyttäjän tai työntekijän virheet. (Holdsworth & Kosinski 2024.)

Kerrostettu suojaus

Kerrostetussa suojauksessa tarkoituksena on hieman kuin vastata kaikkiin mahdollisiin uhkiin yksittäisillä toimenpiteillä, koska monesti järjestelmä tai sen kehittäjä on aina altavastaajan roolissa hyökkäysten sattuessa. On siis parempi osata ennakoida kuin taistella. Kerrostettu suojaus on kuin avain ja lukko: jokaisen avaimen haitan on vastattava lukon haittoja, jotta lukko aukeaa. Kerrostetun suojauksen oletus on, että mahdollinen hyökkääjä pystyy kenties murtamaan muutaman kerroksen, mutta ei kaikkia. (Hyvärinen 2018.) Kuviossa 17 on esitelty tavanomainen verkkosovelluksen kerrostettu suojaus.



Kuvio 17. Kerrostettu suojaus (Into 2011)

Istunnon hallinta ja autentikointi

Verkkosovelluksista puhuttaessa istunnolla tarkoitetaan juuri sen hetkistä aktiivista kanssakäymistä verkkosovelluksen kanssa. Eli kun käyttäjä saapuu verkkosivustolle, istunto alkaa, ja kun hän kirjautuu ulos tai poistuu muuten sovelluksesta, istunto päättyy. Istunto pitää siis sisällään kaiken kanssakäymisen ja tiedon välityksen sovelluksen kanssa. Väärin rakennettu tai rikkinäinen istunnonhallinta on OWASP Top Ten -listauksen (2021) sijalla 1. (Session Management Cheat Sheet n.d.)

Istunnon hallinnassa kyse on istuntoon liittyvien parametrien tai verkkoselaimeen tallennettavan tiedon manipuloinnista ja istuntokohtaisten avainten hallinnasta. Erilaisilla hallintakeinoilla käyttäjille voidaan tuottaa yksilökohtaisia istuntoja, joissa pystytään näyttämään käyttäjäkohtaista tietoa sovelluksen eri osissa. Tyypillinen esimerkki istunnonhallinnasta alkaa, kun käyttäjä kirjautuu verkkosovellukseen sisään käyttäen käyttäjätunnusta ja salasanaa: kun kirjautuminen on onnistunut, käyttäjälle voidaan tarjota tietynlaista istunnon uusiutumisavainta, jonka avulla käyttäjän ei esimerkiksi tarvitse kirjautua sovellukseen uudelleen tiettyyn aikarajaan asti. Nämä tokeneiksi (token) kutsutut avaimet myönnetään yleensä pareissa: ensimmäinen avain on tarkoitettu varmentamaan

kaikki kanssakäynti sovelluksen palvelimen kanssa, ja toinen avain on tarkoitettu uusimaan ensimmäinen avain. (Session Management Cheat Sheet n.d.)

Uusiutumiseen käytettävä token tallennetaan lähes poikkeuksetta sovelluksen tietokantaan salattuna ja sen varmennus tapahtuu aina palvelimelta eikä käyttöliittymältä. Tämä lisää yhden kerroksen aiemmin käsiteltyyn kerrostetun suojauksen malliin. Ensimmäinen token eli varmenne voidaan tallentaa verkkoselaimen paikallismuistiin tai evästeisiin ja sen voimassaoloaika on yleisesti melko lyhyt, noin 20–30 minuuttia. Verkkosovellusten arkkitehtuurissa lähes poikkeuksetta varmenneavaimet tallennetaankin niin kutsuttuihin HttpOnly-evästeisiin, joihin ei pääse käsiksi käyttöliittymän kautta, vaan sovellus lisää ne automaattisesti palvelimelle lähetettyihin pyyntöihin. Jos siis mahdollinen hyökkääjä saisikin istuntokohtaisen varmenteen haltuunsa, sen lyhyen voimassaolon vuoksi hyökkäysaika jää hyvin lyhyeksi. (Session Management Cheat Sheet n.d.)

Varmenne- ja uusiutumisavainten käyttö sovelluksessa estää tietyntyyppisiä hyökkäystapoja, mutta ei kaikkia. Siitä syystä niiden antamaa suojaa voidaan parantaa lisäämällä jälleen kerroksittaista suojausta. Yksi näistä keinoista on luoda uudenlainen avain tai token, niin kutsuttu CSRF (Cross-Site-Request-Forgery) -avain. Se on uniikki avain, joka myönnetään yksittäistä uniikkia verkkoselainta kohden. Se siis käytännössä identifioi tietyn verkkoselaimen tietylle istunnolle. Jos nyt palataan aiempaan esimerkkiin hyökkääjästä, joka saa haltuunsa varmenneavaimen, CSRF-avain estää tämän varmenteen käytön, jos hyökkääjä ei käytä täysin samaa istuntoa samalla verkkoselaimella kuin mille varmenne on luotu. (Session Management Cheat Sheet n.d.)

Kryptografia

Kryptografia on tiedon salaukseen ja suojaukseen kehitetty menetelmäoppi, jonka tavoitteena on salata selväkielinen tieto niin, että sitä ei pysty tulkita. Kryptografiassa tarkoituksena ei ole salata tiedon olemassaoloa, vaan salata sen sisältö. Yleisiä kryptografian käsitteitä ovat selväkieli, salakieli, salain ja avain. Näistä salain on algoritmi tai menetelmä, jolla selväkieli salataan salakieleksi käyttämällä avainta. Kryptografiasta johtuvat ongelmat verkkosovelluksissa on OWASP Top Ten -listauksen (2021) sijalla 2. (Johdatus kryptografiaan n.d, luvut 2.1-2.4.)

Verkkosovelluksissa tai tietojärjestelmissä kryptografiaa käytetään yleisesti sovelluksen palvelimella enkryptaamaan ja dekryptaamaan haluttua tietoa ennen tietokantaan tallentamista ja sen jälkeen. Myös verkkosovelluksissa käytetty tiedonsiirtoprotokolla HTTPS salaa lähetettävät ja vastaanotettavat tiedot käyttöliittymän ja palvelimen välillä käyttäen TLS-protokollaa. (Arrays 2023.)

Jos tallennettua tietoa ei tarvitse enää koskaan muuntaa selkokieliseksi, yksi vaihtoehto on myös niin kutsuttu 'hashing' eli suomeksi hajautus. Hajautuksen tarkoituksena on muuntaa selkokielinen tieto salakieleksi, jota ei pysty muuntamaan enää takaisin selkokieliseksi. Salasanat ovat yleisesti tämän tyyppisiä tietoja, jotka hajautetaan. Kirjautumistapahtumassa annamme käyttäjänimen lisäksi salasanan, joka lähetetään palvelimelle ja palvelin käyttää samaa hajautusalgoritmia, jolla alkuperäinen salasana on hajautettu, hajauttamaan annetun salasanan, jonka jälkeen salasanoja verrataan keskenään. Jos salasanat täsmäävät niin kirjautuminen onnistuu, muuten ei. (What is the purpose of hashing passwords in web applications? 2023.)

Parametrisoidut SQL-kyselyt

Vuoden 2021 OWASP Top Ten -listauksen (2021) mukaan niin kutsutut SQL-injektiot ovat kolmanneksi yleisimpiä turvallisuusriskejä ja vuoden 2013 sekä 2017 listausten kaikista yleisimpiä turvallisuusriskejä verkkosovelluksissa. SQL-injektio käytännössä tarkoittaa, että sovelluksen käyttämiin tietokantapyyntöihin ujutetaan argumentteja tai arvoja, joiden avulla voidaan varastaa tai muokata tietoa tietokannasta (SQL Injection Prevention Cheat Sheet n.d).

SQL-injektioita vastaan taistelu onnistuu parhaiten käyttämällä parametrisoituja SQL-kyselyitä. Parametrisointi tarkoittaa käytännössä sitä, että tietokantapyyntöjä ja niiden mukana välitettäviä käyttäjän antamia tai tuottamia syötteitä käsitellään kuten mitä tahansa muuta tietotyyppiä sovelluksessa. Ennen kuin lopullinen tiedon haku tietokantaan suoritetaan, kaikki haussa käytettävät parametrit tuotetaan joko automaattisesti sovelluksen toimesta ilman käyttäjän antamaa syötettä tai jos käyttäjän syöte tarvitaan, sen oikeellisuus ja validiteetti voidaan tarkastaa. Parametrisoidussa SQL-kyselyssä parametrejä käsitellään myös aina kuin ne ovat. (SQL Injection Prevention Cheat Sheet n.d.) Kuviossa 18 on esitelty parametrisoimaton ja parametrisoitu SQL-kysely.

```
// Esimerkki parametrisoimattomasta SQL-kyselystä
const username = $_POST['username'];
const password = $_POST['password'];

const sql1 = "SELECT * FROM users WHERE username = '$username'";
const data1 = query(sql1);

// Esimerkki parametrisoidusta SQL-kyselystä
const sql2 = "SELECT * FROM users WHERE username = ?";
const data2 = query(sql2, [username, password]);
return [data1, data2]
```

Kuvio 18. Parametrisoimaton ja parametrisoitu SQL-kysely

Parametrisoimattomassa SQL-kyselyssä hakuun käytettävät argumentit tai arvot, kuten käyttäjätunnus tai salasana, lisätään suoraan osaksi kyselyä. Tämä avaa hyökkääjille mahdollisuuden syöttää haitallisia arvoja, koska kyselyt eivät erota käyttäjän syötettä ja varsinaista SQL-koodia toisistaan. Kuvion 12 esimerkissä hyökkääjä voisi siis syöttää käyttäjätunnuksena esimerkiksi merkkijonon *”janne’ OR ’1’ = ’1”,* jolloin SQL-kyselystä muodostuisi *SELECT * FROM users WHERE username = ’janne’ OR ’1’ = ’1’.* SQL-kyselykielessä looginen vertailuoperaattori *’1’ = ’1’* palauttaa aina arvon ’TRUE’, joten edellä mainittu kysely palauttaisikin jokaisen käyttäjän tietokannasta, koska OR-operaattori etsii joko käyttäjää nimeltä ’janne’ TAI kaikki käyttäjät. (SQL Injection Prevention Cheat Sheet n.d.)

Parametrisoidussa SQL-kyselyssä edellä mainittu esimerkki ei ole mahdollinen, koska kyselyn arvot muutetaan kyselyn suorittavan funktion parametreiksi, jolloin ne siirtyvät muuttumattomina kyselyyn. Kuvion 12 parametrisoidussa SQL-kyselyssä edellisen tekstikappaleen esimerkki *”janne’ OR ’1’ = ’1”* muodostaisikin kyselyn *SELECT * FROM users WHERE username = 'janne'' OR ''1'' = ''1'*, joka etsisikin silloin vain käyttäjänimeä *janne” OR ”1” = ”1*. (SQL Injection Prevention Cheat Sheet n.d.)

Käyttäjätapahtumien seuranta

Tapahtumien seuranta eli loggaus on tehokas tapa seurata sovelluksen tapahtumia. Lokitiedostot ovat tekstitiedostoja, joihin voidaan kirjoittaa tietoa kaikesta toiminnasta sovelluksen sisällä. Modernissa verkkosovelluksessa erilaisia käyttäjätapahtumia istuntoa kohden voi tapahtua satoja tai jopa tuhansia, joten seurannan tulee olla suunniteltua ja ainoastaan tärkeät asiat tulisi kirjata lokitiedostoihin. Siinä missä lokitiedostot voivat paljastaa hyökkäysyrityksiä, ne voivat paljastaa myös sovelluksen virheitä tai poikkeustilanteita. (Mireles 2022.) Ei siis tule yllätyksenä, että OWASP Top Ten -listauksen (2021) mukaan puutteellinen tapahtumien seuranta onkin listauksen sijalla 9.

Lokitiedostoja kirjoittaessa tulisikin miettiä mitä tietoja lokiin halutaan kirjata ja kirjattujen tietojen pitäisi hyödyttää lukijaa jollain tapaa. Esimerkiksi virheiden osalta tulisikin pyrkiä kirjaamaan virheen sijainti ja ajankohta, mutta jos mahdollista niin myös tarkat tiedot siitä missä kohdassa ohjelmakoodia virhe tapahtuu. Tämä helpottaa virheen korjausta, kun tarpeelliset tiedot ovat saatavilla suoraan lokitiedostosta. Mitään arkaluonteista tietoa lokitiedostoihin ei saa tallentaa, esimerkiksi sovelluksen reittipisteisiin tulevia henkilötietoja. (Mireles 2022; Knupfer 2022.)

Lokitiedostojen kerääminen ja analysointi voikin olla jopa ainoa keino saada mahdolliset hyökkäysyritykset, onnistuneet tai epäonnistuneet, sovelluksen ylläpitäjän tietoon. Hyökkäysyritysten lisäksi lokitiedostot auttavat ymmärtämään sovellusta paremmin sekä keräämään statistiikkaa ja suorituskykytietoja. (Knupfer 2022.)

5 Verkkosovelluksen kehitys

5.1 Suunnittelu

5.1.1 Ohjelmointikielet

Sovelluksen ohjelmointikieleksi valittiin JavaScript palvelimen ohjelmointiin ja TypeScript käyttöliittymän ohjelmointiin. Tämä valinta perustui vaatimukseen luoda sovellus, joka on helposti saatavilla, skaalautuu erilaisille päätelaitteille ja tarjoaa sujuvan käyttäjäkokemuksen määritellyille käyttäjäryhmille.

JavaScript valittiin palvelimen ohjelmointikieleksi sen keveyden ja nopeuden vuoksi, erityisesti kun kyseessä ei ole koneoppimisen tai muiden laskennallisesti raskaiden toimintojen integrointi. Sovellus on rakennettu keskittyen nopeuteen, skaalautuvuuteen ja reaaliaikaiseen interaktiivisuuteen, minkä takia JavaScriptin asynkronisten ominaisuuksien ja yksisäikeisen ajoympäristön valitseminen palvelimelle oli luontainen valinta. Se soveltuu erinomaisesti verkkosovelluksiin, jotka eivät edellytä raskasta taustaprosessointia tai tilaa vievää laskentaa.

Python voisi olla hyvä valinta, jos sovelluksen käyttötapauksen sisältäisivät esimerkiksi koneoppimista tai suurten tietomassojen käsittelyä, mutta tässä tapauksessa JavaScriptin kevyt arkkitehtuuri, sen mahdollisuus toteuttaa yksinkertaisia ja nopeita toimintoja, sekä sen sulava integrointi sekä palvelimen, että käyttöliittymän välillä tarjoaa tehokkaan ratkaisun sovelluksen tarpeisiin.

5.1.2 Palvelin ja ajoympäristö

Palvelinympäristön toteutuksessa päädyttiin käyttämään Node.js -ajoympäristöä sen kyvyn vuoksi tarjota skaalautuvaa ja tehokasta suorituskykyä verkkosovelluksille. Node.js käyttää myös samaa JavaScript -ohjelmointikieltä, jolloin koko sovelluksen arkkitehtuuri rakentuu samalle kielelle, mikä tekee toteutuksesta ja ylläpidosta hieman nopeampaa ja yksinkertaisempaa.

Node.js perustuu tapahtumavetoiseen arkkitehtuuriin, joka mahdollistaa asynkronisen tapahtumien käsittelyn. Tämä tekee Node.js:stä erinomaisen valinnan erityisesti verkkosovelluksiin, jotka vaativat reaaliaikaisia toimintoja ja pystyvät käsittelemään samanaikaisia pyyntöjä minimaalisella viiveellä. Node.js:n avulla palvelimen suorituskyky pysyy korkeana, vaikka käyttäjäkuorma kasvaa, sillä se käsittelee pyyntöjä samanaikaisesti. Tämä poistaa tarpeen luoda useita prosesseja jokaiselle käyttäjäpyynnölle, kuten monissa muissa palvelinympäristöissä esimerkiksi CPythonissa.

CPythonin etu tämän sovelluksen kehityksessä olisi ollut laaja kirjasto- ja tukiympäristö esimerkiksi tekoälyn tai numeerisen raskaan laskennan osalta, mutta koska sovelluksessa ei näitä tarvittu, Node.js oli luonnollinen valinta.

Node.js tarjoaa myös laajan yhteisön tuen ja jatkuvasti kehittyvän ekosysteemin, mikä helpottaa kehitystyötä ja varmistaa ajantasaiset ratkaisut verkkosovelluksiin liittyvissä teknologioissa. Nämä edut tekivät siitä optimaalisemman valinnan verrattuna CPythonin kaltaisiin ajoympäristöihin,

jotka ovat hyödyllisempiä laskennallisesti vaativissa ympäristöissä. Jos sovellukseen ikinä integroidaan koneoppimista tai tekoälyä, niin CPythonin pystyy helposti integroimaan olemassa olevan sovelluksen rinnalle.

5.1.3 Sovelluskehys

Sovelluksen käyttöliittymän toteutukseen valittiin React.js yhdessä TypeScript-ohjelmointikielen kanssa. React.js on suosittu JavaScript-kirjasto, joka tarjoaa tehokkaan tavan rakentaa moderneja, responsiivisia ja komponenttipohjaisia käyttöliittymiä. Tämä oli keskeinen tekijä valinnassa, sillä sovelluksen tavoitteena oli skaalautua erilaisille päätelaitteille ja tarjota sujuva käyttäjäkokemus kaikille käyttäjäryhmille.

Django ja Flask ovat suosittuja palvelinpuolen verkkosovellusten kehyksiä, jotka ovat vahvasti sidoksissa Python-ohjelmointikieleen. Vaikka Django ja Flask tarjoavat kattavia ratkaisuja erityisesti palvelimien rakentamiseen, ne eivät ole optimoituja moderneihin verkkosovellusten käyttöliittymätoteutuksiin samalla tavalla kuin React.js. Django tai Flask soveltuvat hyvin tilanteisiin, joissa tarvitsee käsitellä monimutkaisia taustalogiikoita tai tietojärjestelmäintegraatioita. Ne eivät kuitenkaan tarjoa Reactin kaltaista vapaata komponenttirakennetta ja dynaamista käyttöliittymän päivitystä.

React.js:n komponenttipohjainen lähestymistapa mahdollisti uudelleenkäytettävien osien rakentamisen, mikä nopeutti kehitystyötä ja tulevaisuudessa helpottaa ylläpitoa ja sovelluksen muokkaamista. TypeScriptin käyttö React.js:n rinnalla toi lisäetuna staattisen tyypityksen, joka parantaa koodin luotettavuutta ja vähentää käyttöliittymän virheitä. Tämä on erityisen tärkeää sovelluksissa, joissa halutaan ylläpitää selkeää ja turvallista koodipohjaa.

React.js:n ja TypeScriptin valinta mahdollisti kevyen ja dynaamisen käyttöliittymän toteutuksen, joka kommunikoi suoraan palvelimen kanssa samalla ohjelmointikielellä. Tällä lähestymistavalla sovellus kykenee täyttämään reaaliaikaisuuden vaatimukset sekä tarjoamaan sujuvan käyttökokemuksen loppukäyttäjille.

5.1.4 Tietoturvallisuus

Sovelluksen tietoturva-arkkitehtuuri perustettiin kerrostetun suojauksen mukaisesti, jossa eri turvatoimenpiteet jaettiin useisiin tasoihin varmistamaan sovelluksen ja sen käyttäjien tietojen suojaaminen. Näin järjestelmä pystyy puolustautumaan mahdollisilta hyökkäyksiltä ja haavoittuvuuksilta, vaikka yksi suojakerros pettäisikin. Tärkeimmät tietoturvaominaisuudet, kuten istunnon hallinta, autentikointi, kryptografia ja tietokantakyselyiden parametrisointi, toteutettiin ottaen huomioon OWASP Top Ten -listauksen (2021) suositukset.

Istunnon hallinta ja autentikointi

Sovellus käyttää token-pohjaista autentikointia ja istunnon hallintaa. Käyttäjän kirjautuessa hyväksytysti, hänelle myönnetään kirjautumistoken sekä uusiutumistoken. Uusiutumistoken tallennetaan tietokantaan ja luetaan silloin, kun kirjautumistoken on vanhentunut. Jos uusiutumistoken on lukuhetkellä voimassa, myönnetään uusi kirjautumistoken.

Kirjautumistapahtumaan käytetään käyttäjän sähköpostiosoitteen ja salasanan yhdistelmää. Salasanat tallennetaan tietokantaan salattuina ja salaukseen käytetään yksisuuntaista hashing-algoritmia, jolloin salasanan palauttaminen selkokieleksi ei ole mahdollista. Kirjautumistapahtumassa käyttäjän antama salasana ajetaan hashing-algoritmin läpi, jonka jälkeen sitä verrataan tietokantaan tallennettuun salasanaan.

Istunnon hallinnan osalta sovellus käyttää React.js -kehykselle tarkoitettua Redux-kirjastoa, jonka avulla selaimen muistiin voidaan tallentaa istuntotietoja. Istuntotiedot salataan käyttäen kryptografian keinoja sekä niitä tallennetaan niin kauan, kunnes käyttäjä kirjautuu ulos tai tiedot muuttuvat. Tämä mahdollistaa sujuvan käyttäjäkokemuksen sekä vähentää palvelimelle kohdistuvaa liikennettä.

5.2 Palvelin

5.2.1 Rakenne

Sovelluksen palvelinarkkitehtuuri kehitettiin selkeän ja modulaarisen kansiorakenteen ympärille, joka tukee helposti ylläpidettävää, laajennettavaa ja turvallista ratkaisua. Palvelin toimii keskeisenä komponenttina sovelluksen backendissä, halliten tietoturvaa, liikenteen ohjausta ja tiedon välitystä käyttöliittymän ja tietokannan välillä. Koko sovellus sijaitsee projektin juurikansiossa, josta palvelinosio on sijoitettu omaan **server/** -kansioonsa. Kuviossa 19 esitellään palvelimen visuaalinen hakemistorakenne.

```
├── server/
│   ├── config/
│   │   ├── config.js
│   │   └── messagesFI.js
│   ├── db/
│   │   └── db.js
│   ├── middleware/
│   │   ├── auth.js
│   │   └── errorHandler.js
│   ├── routes/
│   │   ├── auth.js
│   │   ├── available.js
│   │   ├── bookings.js
│   │   ├── error.js
│   │   ├── statistics.js
│   │   └── users.js
│   ├── utils/
│   │   ├── encryptIndex.js
│   │   ├── formatDateTime.js
│   │   ├── generatePassword.js
│   │   ├── logger.js
│   │   └── reservationNotifications.js
│   └── server.js
```

Kuvio 19. Palvelimen hakemistorakenne

Keskeiset kansiot

server/: Tämä kansio sisältää kaikki palvelimen toiminnallisuudet ja on organisoitu selkeisiin osioihin:

- **config/:** Tässä kansiossa sijaitsevat konfiguraatiotiedostot, kuten *config.js*, joka sisältää ympäristöön liittyvät asetukset ja määritelmät sekä *messagesFi.js*, joka sisältää pyyntö-vastausmallin suomenkieliset vastausviestit.

- **db/:** Sisältää *db.js* -tiedoston, joka vastaa tietokantayhteyksistä ja tietokantatoimintojen käsittelystä. Tämä rakenne eriyttää tietokantatoiminnot palvelimen muusta logiikasta.

- **middleware/:** Middleware-kansiossa ovat palvelimen keskeiset tietoturvaan ja virheenhallintaan liittyvät komponentit, kuten *auth.js* sekä *errorHandler.js*. Middleware-rakenne mahdollistaa palomuurityylisen turvallisuusrakenteen käyttöliittymän ja palvelimen välille.

- **routes/:** Tämä kansio sisältää kaikki palvelimen rajapinnat ja reitit. Kukin tiedosto, kuten *auth.js, bookings.js* tai *users.js***,** vastaa tietyn toiminnallisuuden reitityksestä. Reittien eriyttäminen tiedostokohtaisesti parantaa koodin luettavuutta ja hallittavuutta.

- **utils/:** Utils-kansio sisältää apuohjelmia ja funktioita, jotka ovat keskeisiä palvelimen toiminnan kannalta, mutta eivät ole suoraan sidottuja reititykseen tai middlewareen. Esimerkkinä *logger.js***,** joka vastaa sovelluksen loggauksesta tai ajastettujen muistuksien lähettämiseen tarkoitettu *reservationNotifications.js*.

- **server.js**: Palvelimen pääkäynnistystiedosto, joka kokoaa yhteen kaikki reitit, middleware-ohjelmat sekä apuohjelmat. Tiedoston kautta luetaan ympäristömuuttujat, konfiguroidaan parametrit ja asetukset, kuten missä portissa sovellus ottaa vastaan pyyntöjä sekä tarjotaan staattisia tiedostoja, kuten esimerkiksi käyttöliittymää.

5.2.2 Server.js

Tämä tiedosto on palvelimen pääkäynnistystiedosto, joka kokoaa yhteen kaikki palvelimen toiminnallisuudet ja määrittelee sen perusrakenteen. Tiedosto vastaa sovelluksen palvelimen käynnistämisestä, reittien hallinnasta, middleware-ohjelmien määrittelemisestä sekä sovelluksen turvallisuustoimintojen toteuttamisesta.

Keskeiset toiminnot

- **Ympäristömuuttujien hallinta**: Palvelin käyttää *dotenv*-kirjastoa lukemaan ympäristömuuttujia *.env*-tiedostosta. Ympäristömuuttujat ovat salassa pidettäviä muuttujia, jotka pitävät sisällään palvelimen konfiguraatioita, kuten käytössä olevia portteja, CORS-alkuperiä ja tietoturvaan liittyviä salaisuuksia, kuten salausavaimia. Näitä tietoja ei saa esiintyä suoraan ohjelmakoodissa.

```
require('dotenv').config({ path: '../.env' });

// Set CORS policy
app.use(cors({
  origin: CORS_ORIGINS,
  credentials: true,
}));
```

- **Middleware-rakenne**: Palvelin käyttää monia middleware-komponentteja, kuten *cookieParser* evästeiden käsittelyyn sekä *bodyParser* JSON-muotoisten pyyntöjen käsittelyyn. Nämä middlewaret on määritelty selkeästi osana palvelimen aloitusprosessia, ja ne tarjoavat perustan palvelimen tietoturvalle ja käyttäjäviestinnälle. Palvelimen liikenteen hallintaan ja suojaukseen käytetään *express-*

rate-limit-kirjastoa, joka rajoittaa käyttäjän tekemien pyyntöjen määrää määritellyn ajanjakson aikana, mikä vähentää palvelinresursseihin kohdistuvaa kuormitusta ja suojaa palvelua mahdollisilta palvelunestohyökkäyksiltä.

```
const bodyParser = require('body-parser');
const cookieParser = require('cookie-parser');
const rateLimit = require('express-rate-limit');
const logger = require('./utils/logger');

// Middleware for parsing cookies
app.use(cookieParser());

// Apply rate limiter
const limiter = rateLimit({
  windowMs: 10 * 60 * 1000, // 10 minutes
  max: 300, // limit each IP to 100 requests per windowMs
  handler: (req, res, /*next*/) => {
    res.status(429).json({ message: messagesFI.error.tooManyRequests });
  }
});
app.use(limiter);

// Body parser middleware
app.use(bodyParser.json());

// Middleware to log incoming requests
app.use((req, res, next) => {
  logger.info(`Incoming req: ${req.method} ${req.url}`);
  next();
});

// Middleware to log responses
app.use((req, res, next) => {
  const originalSend = res.send;
  res.send = function(body) {
    logger.info(`Response status: ${res.statusCode} for ${req.method} ${req.url}`);
    originalSend.call(this, body);
  }
  next();
});
```

- **CSRF-suojaus**: CSRF eli Cross-Site-Request-Forgery -suojaus on toteutettu käyttämällä *csrf*-kirjastoa, joka luo ja tarkistaa CSRF-tunnisteet kaikissa pyynnöissä palvelimelle. Tämä sitoo käyttäjäistunnon tiettyyn selaimeen, joka estää ulkopuolisen tahon istunnonaikaisen toiminnan sovelluksessa käyttäjän nimissä. CSRF-suojaus on käytössä kaikilla reiteillä, lukuun ottamatta */api/refresh* -reittiä, jota tulee pystyä kutsumaan ilman istuntokohtaista tunnistetta.

```
const csrf = require('csurf');
// CSRF protection middleware
const csrfProtection = csrf({ cookie: true });

// Route to get CSRF token
app.get('/api/csrf-token', csrfProtection, (req, res) => {
  const csrfToken = req.csrfToken();
  logger.info(`Issuing CSRF token: ${csrfToken}`);
  res.cookie('XSRF-TOKEN', csrfToken, {
    secure: process.env.NODE_ENV === 'production',
    sameSite: 'strict',
  });
  res.json({ csrfToken });
});

// Apply CSRF protection middleware globally to all routes except /api/refresh
app.use((req, res, next) => {
  if (req.url === '/api/refresh') {
    return next();
  }
  csrfProtection(req, res, next);
});

// CSRF error handling middleware
app.use((err, req, res, next) => {
  if (err.code === 'EBADCSRFTOKEN') {
    // CSRF token validation failed
    res.clearCookie('refreshToken', {
      httpOnly: true,
      secure: process.env.NODE_ENV === 'production',
      sameSite: 'strict',
    });
    res.status(403).json({ message: 'Invalid CSRF token. Please log in again.' });
  } else {
    next(err);
  }
});
```

- **Reititys ja rajapinnat**: Palvelin määrittelee useita reittejä, joista jokainen on jaettu omiin tiedostoihinsa, kuten *authRoutes*, *bookingRoutes* ja *userRoutes*. Tämä selkeä jako parantaa koodin hallittavuutta sekä luettavuutta. Reitit palvelevat API-pyyntöjä, ja ne on ryhmitelty loogisesti eri toimintojen mukaan, esimerkiksi autentikointiin tai käyttäjän varausten hallintaan. Palvelimen juurihakemistosta, eli kun käyttäjä saapuu sovellukseen, palvelin tarjoaa staattisesta reitistä käyttöliittymän.

```
const authRoutes = require('./routes/auth');
const bookingRoutes = require('./routes/bookings');
const availableRoutes = require('./routes/available');
const userRoutes = require('./routes/users');
const statisticRoutes = require('./routes/statistic');
const errorRoute = require('./routes/error')

// Routes
app.use('/api', authRoutes);
app.use('/api', bookingRoutes);
app.use('/api', availableRoutes);
app.use('/api', userRoutes);
app.use('/api', statisticRoutes);
app.use('/api', errorRoute);

// Serve static files
app.use(express.static(path.join(__dirname, '../build')));

// Catch-all route to handle client-side routing
app.get('*', (req, res) => {
  res.sendFile(path.join(__dirname, '../build', 'index.html'));
});
```

- **Virheenkäsittely**: Palvelimen virheenkäsittely on toteutettu käyttämällä keskitettyä virheenkäsittelyfunktiota *errorHandler*. Tämä toiminnallisuus ja rakenne takaa, että kaikki odotetut ja odottamattomat virheet käsitellään yhtenäisesti, ja virheilmoitukset lähetetään käyttöliittymään järkevästi ja informatiivisesti. CSRF-virheiden käsittely on myös otettu huomioon, mikä lisää sovelluksen tietoturvaa.

```
const logger = require('../utils/logger')

const errorHandler = (err, req, res, next) => {
  logger.error(`Error: ${err.message} - ${req.method} ${req.url}`);
  res.status(err.status || 500).json({
    message: err.message || 'Jotain meni vikaan, ole hyvä ja yritä uudelleen',
  });
};

module.exports = errorHandler;

const errorHandler = require('./middleware/errorHandler');

// Error handling middleware
app.use(errorHandler);
```

- **Ajastetut tehtävät**: Palvelin käyttää ajastettuja tehtäviä varausmuistutusten lähettämiseen sähköpostitse. Tämä toiminnallisuus on toteutettu *setupCronJob*-funktiolla, joka käynnistyy heti palvelimen käynnistyessä. Ajastetut tehtävät mahdollistavat automatisoidut ilmoitukset, jotka parantavat käyttäjäkokemusta ja sovelluksen toimivuutta.

```
const setupCronJob = (next) => {
  cron.schedule('0 * * * *', async () => {
    try {
      await sendNotifications24hours();
      await sendNotifications4days();
    } catch(error) {
      next(error)
    };
  });
};

module.exports = setupCronJob;

const setupCronJob = require('./utils/reservationNotifications');

// Start cronjob for reservation notifications
setupCronJob();
```

5.2.3 Rajapinnat

Palvelimen **routes/** -kansio sisältää kaikki palvelimelle määritellyt reitit ja rajapinnat. Erilaiset reitit mahdollistavat monipuolisen palvelimen ja käyttöliittymän toiminnan. Reitit on kerätty erilaisten tiedostojen sisään sillä ajatuksella, että ne liittyvät johonkin sovelluksen yksittäiseen teemaan. Näin ollen esimerkiksi *auth.js*-tiedostossa sijaitsevat kaikki autentikointia vaativat reitit. Tämä ei kuitenkaan poissulje autentikointia vaativien reittien olemassaoloa myös muissa tiedostoissa. Seuraavissa kappaleissa on käsitelty rajapintojen ja reittien tarkoitukset ja ominaisuudet siltä osin kuin ne sovelluksen ja tämän tutkimuksen ymmärtämisen ja sisäistämisen kannalta on järkevää. Kaikkia reittejä ei ole käsitelty aiheen laajuuden vuoksi.

Auth.js

Tämä tiedosto vastaa sovelluksen autentikointi- ja käyttäjänhallintatoiminnoista, ja se sisältää kaikki olennaiset reitit, joita tarvitaan käyttäjien kirjautumiseen, rekisteröitymiseen, sekä tietoturvaan liittyvien toimintojen hallintaan. Tämä tiedosto on keskeinen osa palvelimen turvallisuusrakennetta sillä se vastaa käyttäjien todennuksesta, pääsyoikeuksien hallinnasta ja kirjautumiseen liittyvistä toiminnoista:

Token-pohjainen autentikointi: auth.js käyttää JSON Web Token (JWT) -tekniikkaa käyttäjien autentikointiin. JWT:tä käytetään luomaan lyhytikäisiä kirjautumisavaimia tai tokeneita, jotka sallivat käyttäjien päästä käsiksi sovelluksen rajapintoihin. Lisäksi luodaan pidempiaikaisia uusiutumistoke-

neita, joiden avulla käyttäjät voivat uusia kirjautumistokenin ilman uudelleen kirjautumista. Tokenit luodaan funktiolla *generateAccessToken()*, joka hyödyntää käyttäjätunnisteita, roolia ja alueellisia tietoja.

```
# Tokenin tarkastusfunktio

const jwt = require('jsonwebtoken');
const { SECRET_KEY } = require('../config/config');
const messagesFI = require('../config/messagesFI');

const authenticateToken = (req, res, next) => {
  try {
    const authHeader = req.headers['authorization'];
    const token = authHeader && authHeader.split(' ')[1];
    if (!token) {
      const error = new Error(messagesFI.error.tokenMissing);
      error.status = 401;
      throw error;
    }

    jwt.verify(token, SECRET_KEY, (err, user) => {
      if (err) {
        const error = new Error(messagesFI.error.tokenInvalid);
        error.status = 403;
        throw error;
      }
      req.user = user;
      next();
    });
  } catch (error) {
    next(error);
  }
};

module.exports = { authenticateToken };

# Tokenien luontifunktiot

// Generate access token
const generateAccessToken = (user) => jwt.sign({ id: user.id, area: user.area, role:
user.role }, SECRET_KEY, { expiresIn: '30m' });

// Generate refresh token
const generateRefreshToken = (user) => {
  const refreshToken = jwt.sign({ id: user.id }, REFRESH_SECRET_KEY, { expiresIn: '7d'
});
  return refreshToken;
};
```

Käyttäjän kirjautuminen: Kirjautumisreittiä */login* käytetään tarkistamaan käyttäjän kirjautumistiedot, luomaan kirjautumisavaimen sekä uusiutumisavaimen, joka tallennetaan myös tietokantaan. Onnistuneen kirjautumisen jälkeen käyttäjä saa palvelimelta vastauksena edellä mainitut avaimet sekä käyttöliittymälle välitetään käyttäjän tietoja.

```
// Login route
router.post('/login', async (req, res, next) => {
  const { email, password } = req.body;
  try {
    const user = (await getUserByEmail(email))[0];

    if (user) {
      if (bcrypt.compareSync(password, user.password)) {
        const accessToken = generateAccessToken(user);
        const refreshToken = generateRefreshToken(user);

        await addRefreshToken(refreshToken, user.id, new Date(Date.now() + 7 * 24 * 60 *
60 * 1000));

        // Set the HttpOnly cookie
        res.cookie('refreshToken', refreshToken, {
          httpOnly: true,
          secure: true,
          sameSite: 'Strict',
          maxAge: 7 * 24 * 60 * 60 * 1000, // 7 days
        });

        res.json({
          message: messagesFI.success.login,
          accessToken,
          id: user.id,
          area: user.area,
          email: user.email,
          role: user.role,
          firstname: user.firstname,
          lastname: user.lastname,
          phonenumber: user.phonenumber,
          ergometry_expirary_date: formatDateTime(user.ergometry_expirary_date),
          fitness_test_expirary_date: formatDateTime(user.fitness_test_expirary_date),
          goal: user.goal,
          fire_station: user.fire_station,
        });
      } else {
        // Password does not match
        res.status(401).json({ message: messagesFI.error.invalidPassword });
      }
    } else {
      // Email not found
      res.status(401).json({ message: messagesFI.error.invalidEmail });
    }
  } catch (error) {
    next(error);
  }
});
```

Käyttäjän uloskirjautuminen: Uloskirjautumiseen käytetään reittiä */logout*, joka poistaa tietokantaan tallennetun uusiutumisavaimen sekä poistaa selaimeen tallennetut evästeet.

```
// Logout route
router.post('/logout', authenticateToken, async (req, res, next) => {
  const { userId } = req.body;
  try {
    // Remove the refresh token from the database
    await deleteRefreshToken(userId);

    // Clear the refresh token cookie
    res.clearCookie('refreshToken', {
      httpOnly: true,
      secure: process.env.NODE_ENV === 'production',
      sameSite: 'None',
    });

    res.status(200).json({ message: messagesFI.success.loggedOut });
  } catch (error) {
    next(error);
  }
});
```

Käyttäjän rekisteröinti: Rekisteröintireitti */register* vaatii autentikoidun käyttäjän. Uudet käyttäjät rekisteröidään tietokantaan ja uudelle käyttäjälle luodaan satunnainen salasana. Rekisteröinnin jälkeen käyttäjälle lähetetään sähköpostivahvistus, jossa ilmoitetaan uudesta käyttäjätilistä sekä kerrotaan käyttäjän kirjautumistiedot.

```
// Register route
router.post('/register', authenticateToken, async (req, res, next) => {
  const {
    firstname,
    lastname,
    …
  } = req.body;

  try {
    const user = (await getUserByEmail(email))[0];
    if (user) {
      const error = new Error(messagesFI.error.emailTaken);
      error.status = 400;
      next(error);
    } else {
      const password = generateRandomPassword();
      const hashedPassword = bcrypt.hashSync(password, 10);

      // Encrypt the ergometryIndex and fitnessTestIndex before saving
      const encryptedErgometryIndex = ergometryIndex ? encrypt(ergometryIndex.toString())
: null;
      const encryptedFitnessTestIndex = fitnessTestIndex ?
encrypt(fitnessTestIndex.toString()) : null;

      await createUser(
        firstname,
        lastname,
        …
      );

      const mailOptions = {
        from: SMTP_CONFIG.auth.user,
        to: email,
        subject: messagesFI.email.subject,
        text: messagesFI.email.body(firstname, lastname, email, password,
process.env.CORS_ORIGIN_URI),
      };

      transporter.sendMail(mailOptions, (error, info) => {
        if (error) {
/*          const emailError = new Error(messagesFI.error.emailSendError);
          emailError.status = 500;
          next(emailError); */
        } else {
          res.status(201).json({ message: messagesFI.success.userRegistered });
        }
      });
    }
  } catch (error) {
    next(error);
  }
});
```

Salasanan vaihto: Salasanan vaihtoon tarkoitettu reitti */change-password* vertaa kirjoitettua vanhaa salasanaa tietokantaan tallennettuun salasanaan ja jos se täsmää niin uusi annettu salasana tallennetaan vanhan tilalle käyttäen *changePassword()*-funktiota.

```
// Change password route
router.post('/change-password', authenticateToken, async (req, res, next) => {
  const { oldPassword, newPassword } = req.body;
  try {
    const password = (await getUserPassword(req.user.id))[0];

    if (password && bcrypt.compareSync(oldPassword, password.password)) {
      const hashedPassword = bcrypt.hashSync(newPassword, 10);
      await changePassword(req.user.id, hashedPassword);
      res.json({ message: messagesFI.success.passwordChanged });
    } else {
      const error = new Error(messagesFI.error.oldPasswordIncorrect);
      error.status = 401;
      next(error);
    }
  } catch (error) {
    next(error);
  }
});
```

Ilmoitustaulua koskevat rajapinnat: Sovelluksen ilmoitustaululle on *auth.js*-tiedostossa useita erilaisia rajapintoja lähettämään, hakemaan, muokkaamaan tai poistamaan ilmoituksia. Alla olevassa koodissa on esitelty ilmoituksen lisäämiseen tarkoitettu reitti */postMessage*.

```
router.post('/postMessage', authenticateToken, async (req, res, next) => {
  const { id, message, targetAreas, targetRoles, targetFireStations,
sendEmailNotification } = req.body;

  try {
    // Post the message
    await postMessage(id, message, targetAreas, targetRoles, targetFireStations,
sendEmailNotification);
    res.status(200).json({ message: messagesFI.success.notificationPosted });

    // Check if email notifications should be sent
    if (sendEmailNotification) {
      // Get the list of user emails based on the target variables
      const emails = await getUserEmails(targetAreas, targetRoles, targetFireStations);

      // Iterate over each email and send a notification
      for (const email of emails) {
        const mailOptions = {
          from: SMTP_CONFIG.auth.user,
          to: email.email,
          subject: messagesFI.newMessagePosted.subject,
          text: messagesFI.newMessagePosted.body(),
        };
        // Send email
        transporter.sendMail(mailOptions, (error, info) => {
          if (error) {
            console.error(`Failed to send email to ${email}:`, error);
          }
        });
      }
    }
  } catch (error) {
    next(error);
  }
});
```

Available.js

Tämä tiedosto pitää sisällään testausaikojen suunnittelun hallintaan liittyvät rajapinnat. Se sisältää useita reittejä, joiden avulla voidaan asettaa, päivittää tai poistaa testaajakohtaisia testausaikoja tai ilmoittaa poissaoloista. Nämä toiminnallisuudet ovat keskeisiä, jotta testejä ja varauksia voidaan hallita tehokkaasti ja koordinoidusti sovelluksen sisällä. Joidenkin reittien lähdekoodi on niin laajaa, että niiden sisältöä ja pituutta joudutaan typistämään. Typistäminen tehdään käyttäen kolmea pistettä '…'.

Uuden testausajan suunnittelu: Reitti */availability* mahdollistaa uusien testausaikojen suunnittelun ja lähettämisen palvelimelle tietokantaan tallennettavaksi.

```
// Route to set availability for a tester
router.post('/availability', authenticateToken,async (req, res, next) => {
    const { testerId, date, isAvailable, start_time, end_time, fireStation, area } =
req.body;
    try {
        await setAvailability(testerId, date, isAvailable, start_time, end_time,
fireStation, area);
        res.json({ message: messagesFI.success.eventAddedSuccess });
    } catch (error) {
        next(error);
    }
});
```

Testausaikojen hakeminen tietokannasta: Reittejä */availability/:testerId* sekä */availability/:testerId/:date/:start_time* käytetään hakemaan yksittäisen uniikin testaajan suunnitellut testausajat tai spesifisti uniikin testaajan uniikki testausaika päivämäärän ja aloitusajan perusteella. Vastaavasti samoja reittejä voidaan käyttää kutsumalla rajapintaa muilla http metodeilla. Esimerkiksi suunnitellun testausajan voi poistaa kutsumalla */availability/:testerId/:date/:start_time* -reittiä käyttäen http metodia DELETE.

```
// Route to get availability for a tester
router.get('/availability/:testerId', authenticateToken, async (req, res, next) => {
  const { testerId } = req.params;
  try {
    const availability = await getAvailabilityByTesterId(testerId);
    res.json(availability);
  } catch (error) {
    next(error);
  }
});

// Route to get specific availability event for a tester
router.get('/availability/:testerId/:date/:start_time', authenticateToken, async (req,
res, next) => {
  const { testerId, date, start_time } = req.params;
  try {
    const availability = await getSpecificAvailability(testerId, date, start_time);
    if (availability.length === 0) {
      return res.status(404).json({ message: messagesFI.error.eventNotFound });
    }
    res.json(availability[0]);
  } catch (error) {
    next(error);
  }
});
```

Poissaolon ilmoittaminen: Reittiä */mark-absence* käytetään merkitsemän jokin tietty testausaika poissaoloksi. Reitin käyttäminen peruuttaa kaikki testausajalle varatut testausajat ja lähettää sähköposti-ilmoituksen kaikille varanneille käyttäjille. Huomaa, että seuraavassa koodilohkossa joitain toiminnallisuuksia on typistetty kolmella pisteellä tilan säästämiseksi.

```
// Route to update is_available field and mark absence
router.post('/mark-absence', authenticateToken, async (req, res, next) => {
  const { availableDateId, absenceReason } = req.body;

  try {
    const result = await updateAvailabilityWithId(availableDateId);
    const testerUser = await getUserById(result.tester_id);
    const datePart = new Date(result.date)
      .toLocaleDateString('en-CA', { timeZone: 'Europe/Helsinki' })
      .split('T')[0];
    const [year, month, day] = datePart.split('-');
    const formattedDate = `${day}.${month}.${year}`;
    const tester_name = `${testerUser[0].firstname} ${testerUser[0].lastname}`;
    const bookings = await getBookingsByTesterNameAndDate(tester_name, datePart);

    // Group bookings by userId
    const bookingsByUser = bookings.reduce((acc, booking) => {
      …
      return acc;
    }, {});

    // Array to store IDs of deleted bookings
    const deletedBookingIds = [];

    // Iterate over each user and send a single email with all their bookings
    for (const userId in bookingsByUser) {
      const user = await getUserById(userId);
      const userBookings = bookingsByUser[userId];

      // Generate a list of booking details
      const bookingDetails = userBookings.map((booking) => {
        …
      }).join('');

      const mailOptions = {
        from: SMTP_CONFIG.auth.user,
        to: user[0].email,
        subject: messagesFI.testerAbsence.subject,
        text: messagesFI.testerAbsence.body(tester_name, formattedDate, bookingDetails,
absenceReason), // Pass bookingDetails to the body
      };

      transporter.sendMail(mailOptions, (error, info) => {
        if (error) {
          const emailError = new Error(messagesFI.error.emailSendError);
          emailError.status = 500;
          next(emailError);
        }
      });

      // After sending the email, delete the bookings and collect the IDs
      for (const booking of userBookings) {
        await deleteBookingById(booking.id, 'Testaajan poissaolo');
        deletedBookingIds.push(booking.id); // Collect the deleted booking ID
      }
    }

    // Respond with the success message and the IDs of the deleted bookings
    res.json({
      message: messagesFI.success.absenceMarked,
      deletedBookingIds: deletedBookingIds
    });
  } catch (error) {
    next(error);
```

Bookings.js

Tähän tiedostoon on kerätty testausaikojen varaukseen liittyvät reitit ja rajapinnat. Varauksia pystytään lisäämään, muokkaamaan tai poistamaan sekä järjestelmä rajoittaa varausten tekoa tietyin ehdoin. Esimerkiksi paloasemalle, jolla ei ole fyysistä ergometriatestiin käytettävää kuntopyörää, ei voi varata testausaikoja ergometriatestiin vaikka kyseiselle paloasemalle olisi suunniteltu testaajalle työaikaa. Koodilohkoja on jouduttu typistämään käyttäen kolmea pistettä.

Varauksien haku: Reittiä */varaukset* sekä siihen liitettävää URL-parametriä *:id* voidaan käyttää hakemaan käyttäjäkohtaiset varaukset tai varauksen ID:llä yksittäinen uniikki varaus. Reittiä */getAllBookings* käytetään pääkäyttäjärooleissa hakemaan kaikki tietokannassa olevat varaukset.

```
// Route to get bookings
router.get('/varaukset', authenticateToken, async (req, res, next) => {
  try {
    // Fetch user data by user ID
    const user = await getUserById(req.user.id);
    …

    // Fetch bookings by area
    const bookings = await getBookingsByArea(area, role);

    // Process bookings based on user role
    const userBookings = bookings.map((booking) => (
      role === 'Testaaja' || role === 'admin' || role === 'Esihenkilö' || booking.userId
=== req.user.id ? booking : {
        …
      }
    ));

    …
    res.json({ bookings: formattedBookings });
  } catch (error) {
    next(error);
  }
});



// Route to get booking details by ID
router.get('/varaukset/:id', authenticateToken, async (req, res, next) => {
  try {
    const booking = (await getBookingById(req.params.id))[0];
    if (!booking) return res.status(404).json({ message: messagesFI.error.bookingNotFound
});

    // Format dates to ISO strings for frontend
    const formattedBooking = {
     …
    };

    res.status(200).json(formattedBooking);
  } catch (error) {
    next(error);
  }
});
```

```
// Route to get all bookings with user details
router.get('/getAllBookings', authenticateToken, async (req, res, next) => {
  try {
    const bookings = await getAllBookings();

    // Convert dates to ISO strings for frontend
    const formattedBookings = bookings.map((booking) => ({
      …
    }));

    res.json({ bookings: formattedBookings });
  } catch (error) {
    next(error);
  }
});
```

Varauksien tekeminen: Reittiä */varaukset* voidaan jälleen käyttää uudelleen toisella http metodilla. Käyttäen metodia POST, voidaan samaan rajapintaan lähettää tallennettavaksi tarkoitettu varaus. Varaus sidotaan varauksen tehneeseen käyttäjään ja tallennetaan tietokantaan. Varauksesta lähetetään sähköpostivahvistus käyttäjän sähköpostiin.

```
// Route to create a new booking
router.post('/varaukset', authenticateToken, async (req, res, next) => {
  try {
    let user;
    if (req.body.email) {
      user = await getUserByEmail(req.body.email);
    } else {
      user = await getUserById(req.user.id);
    }
    …
    const newBooking = {
      start: formatDateTime(req.body.start),
      end: formatDateTime(req.body.end),
      title: req.body.title,
      userId: user[0].id,
      area: user[0].area,
      fire_station: req.body.fire_station,
      tester: req.body.tester,
      approved: req.body.approved,
    };

    const bookingId = await createBooking(newBooking);
    …

    transporter.sendMail(mailOptions, (error, info) => {
      …
    });

  } catch (error) {
    next(error);
  }
});
```

Varausten luomiseen vaikuttavat tekijät: Reiteillä */bike-location* sekä */getTitles* rajoitetaan käyttäjien kykyä luoda uusia varauksia tietyille testeille sekä tietyille testinimikkeille. Käyttäjä pystyy varamaan testausaikoja vain tiettyihin testinimikkeisiin sekä ehdollisesti paloasemille.

```
// route to get bike locations
router.get('/bike-location', authenticateToken, async (req, res, next) => {
  const user = await getUserById(req.user.id);
  if (!user) {
    return res.status(404).json({ message: messagesFI.error.userNotFound });
  }

  const role = user[0].role;
  const area = user[0].area;

  try {
    let bikeLocations;
    …
    res.status(200).json({ location: bikeLocations });
  } catch (error) {
    next(error);
  }
});
// route to get titles and labels
router.get('/getTitles', authenticateToken, async (req, res, next) => {
  const titles = await getTitles();
  return res.json(titles);
})
```

Statistics.js

Tämä tiedosto pitää sisällään ainoastaan pääkäyttäjä- ja esihenkilönäkymän Statistiikka-välilehdelle renderöitävien muuttujien ja arvojen hakemiseen tarkoitetun rajapinnan. Tiedostossa on vain yksi reitti */statistics*, jota kutsumalla palvelin palauttaa muun muassa kaikkien käyttäjien lukumäärän tietokannasta, käyttäjäjakauman sekä voimassa olevat testit, että vanhenevat testit. Palvelin palauttaa suoraan lukuarvoja, jotka käyttöliittymä käsittelee esiteltävään muotoon.

```
router.get('/statistics', authenticateToken, async (req, res, next) => {
  const { role, area } = req.query;

  try {
    const statistics = {};

    // Get total number of users
    const totalUsers = await getTotalUsers(area, role);
    // Get total number of bookings
    const totalBookings = await getTotalBookings(area, role);
    // Get user distribution
    const userDistribution = await getUserDistribution(area, role);
    // Get FireFit index data and count occurrences
    …
    // Count occurrences of each unique userIndex
    const userIndexCounts = formattedUsersWithAverageIndex.reduce((acc, user) => {
      …
    }, {})
    // Get booking trends
    const bookingTrends = await getBookingTrends(area, role);
    // Get valid tests
    const validTests = await getValidTests(area, role);
    …
    // Get expiry alerts
    const expiryAlerts = await getExpiryAlerts(area, role);
    statistics.expiryAlerts = expiryAlerts;

    // Get expiry trends
    const expiryTrends = await getExpiryTrends(area, role);
    …
    // Get percentage of reserved bookings vs. available times
    const bookingsVsAvailableTimes = await getBookingAgainstAvailableTimes(area, role);
    statistics.bookingsVsAvailableTimes = bookingsVsAvailableTimes;

    // get bookings vs. available times data grouped by date for line graph
    const lineGraphData = await getBookingVsAvailableInAreaForLineGraph(area, role);
    statistics.lineGraphData = lineGraphData;

    // get deleted bookings
    const deletedBookings = await getDeletedBookings();
    statistics.deletedBookings = deletedBookings;

    res.json(statistics);
  } catch (error) {
    next(error)
    /* res.status(500).json({ error: messagesFI.error.statisticError }); */
  }
});
```

Users.js

Users.js-tiedosto pitää sisällään rekisteröityneihin käyttäjiin liittyvät reitit ja rajapinnat. Suurin osa reiteistä käyttää http metodia GET hakemaan käyttäjiä sovelluksen eri osiin. Reittejä on suunniteltu sekä pääkäyttäjä- ja esihenkilörooleille, mutta myös muille rooleille. Eri reittipisteillä saadaan haettua tietokannasta selkeämmin esimerkiksi jonkin tietyn roolin käyttäjiä. Alla on esitelty muutamia reittipisteitä Users.js tiedostosta.

```
// Route to get all users in all areas
router.get('/getAllUsers', authenticateToken, async (req, res, next) => {
  try {
    const users = await getAllUsers();

    // Calculate and append the average index for each user
    const usersWithAverageIndex = calculateAndAppendAverageIndex(users);

    // Return the modified users array
    res.json(usersWithAverageIndex);
  } catch (error) {
    next(error);
  }
});



// Route to get unique user
router.get('/getUser/:id', authenticateToken, async (req, res, next) => {
  const { id } = req.params;
  try {
    const user = await getUserById(id);
    res.json(user);
  } catch (error) {
    next(error);
  }
});



// Route to get all regular users from area
router.get('/getRegularUsersFromArea', authenticateToken, async (req, res, next) => {
  const { area } = req.query;
  try {
    const users = await getRegularUsersFromArea(area);
    // Format the dates for each user and calculate the average index
    const formattedUsersWithAverageIndex = calculateAndAppendAverageIndex(
      users.map(user => ({
      …
      }))
    );
    // Return the modified users array
    res.json(formattedUsersWithAverageIndex);
  } catch (error) {
    next(error);
  }
});



// Route to update user info
router.post('/updateUserInfo', authenticateToken, async (req, res, next) => {
  const { id, firstname, lastname, email, phonenumber, newGoal } = req.body;
  try {
    const result = await updateUserInfo(id, firstname, lastname, email, phonenumber,
newGoal);
    if (result.affectedRows > 0) {
      res.status(200).json({ message: messagesFI.success.userInfoUpdatedSuccess });
    } else {
      res.status(404).json({ message: messagesFI.error.userInfoUpdatedError });
    }
  } catch (error) {
    next(error); // Passes the error to the error handling middleware
  }
});
```

5.2.4 Middleware

Palvelimen **middleware/**-kansio on yksi tärkeimmistä palvelimen osista. Middlewaren tehtävänä on olla asioiden välissä ja näin sen käyttö on rakennettu myös tässä sovelluksessa. Sovelluksen middleware käsittää käytännössä kaksi lyhyttä funktiota, joilla on kuitenkin todella tärkeä tehtävä. Toinen funktio *authenticateToken()* on kaikkien *auth.js*-tiedostossa mainittujen, sekä myös muiden reittien, kutsuma funktio silloin kun reittiä kutsuvan identiteetti on autentikoitava.

```
const authenticateToken = (req, res, next) => {
  try {
    const authHeader = req.headers['authorization'];
    const token = authHeader && authHeader.split(' ')[1];
    if (!token) {
      const error = new Error(messagesFI.error.tokenMissing);
      error.status = 401;
      throw error;
    }

    jwt.verify(token, SECRET_KEY, (err, user) => {
      if (err) {
        const error = new Error(messagesFI.error.tokenInvalid);
        error.status = 403;
        throw error;
      }
      req.user = user;
      next();
    });
  } catch (error) {
    next(error);
  }
};
```

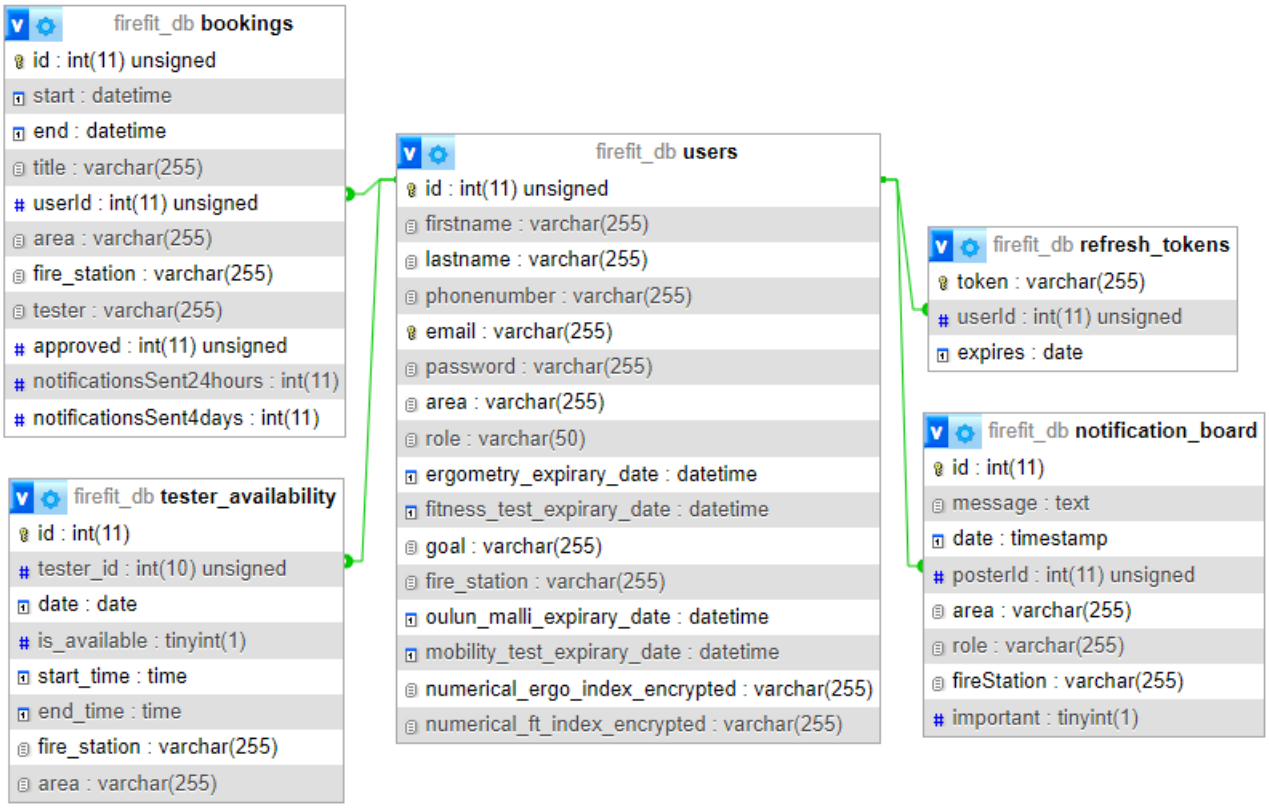
authenticateToken()-funktio kutsutaan siis ennen kuin pyyntö pääsee palvelimelle ja funktio tarkastaa istunnon- ja käyttäjänhallintaan liittyvät tokenit tai avaimet. Jos näitä ei ole niin pyyntö hylätään.

errorHandler()-funktio toimii samaan tapaan niin, että funktiota voidaan kutsua aina kun halutaan hallita jonkin toisen funktion tai toiminnon tuottamia virheitä. Eli sen sijaan, että jokainen toiminnallisuus tuottaa omanlaisen virheilmoituksen, *errorHandler()*-funktiolla yhtenäistetään virheilmoitusten tyyli ja esitystapa.

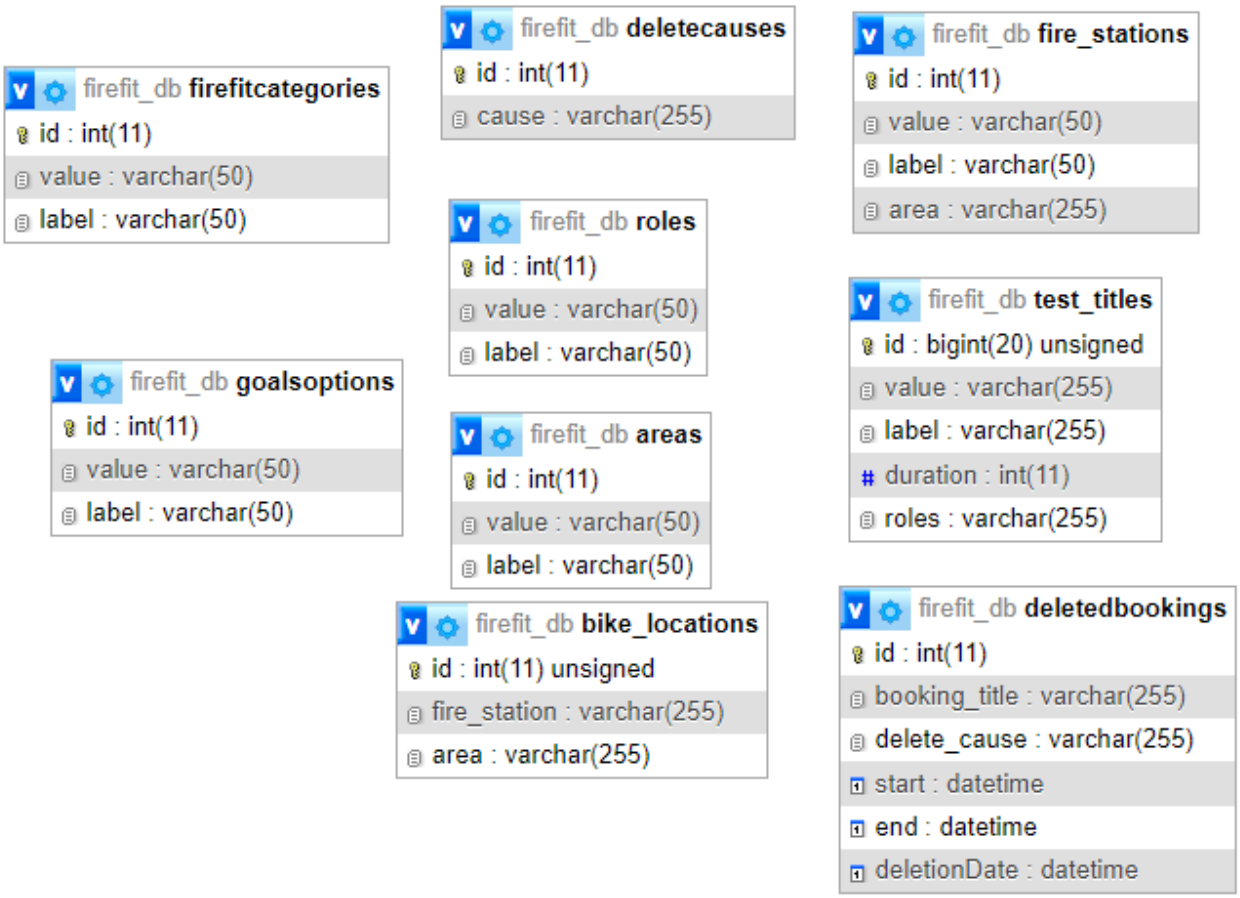
```
const errorHandler = (err, req, res, next) => {
  logger.error(`Error: ${err.message} - ${req.method} ${req.url}`);
  res.status(err.status || 500).json({
    message: err.message || 'Jotain meni vikaan, ole hyvä ja yritä uudelleen',
  });
};
```

5.2.5 Tietokanta

Sovelluksen tietokanta rakentuu julkaisualustan domainilla olevalle MariaDB-tietokannalle. MariaDB on SQL (Structured Query Language) -tietokanta, joka mahdollistaa relaatioiden muodostamisen tietueiden välille. Sovelluksessa on erilaisia tietokantatauluja 14 kappaletta, joihin tallennetaan spesifisti sovelluksen tuottamaa dataa tai sovelluksen muutoin hyödyntämiä staattisia tietoja esimerkiksi paloasemien nimiä tai käyttäjärooleja. Tämä lähestymistapa helpottaa tulevaisuudessa tietojen muokkaamista tai muuntamista, kun sovellus hakee staattiset tiedot tietokannasta. Kuviossa 20 on esitelty tietokantataulut, joissa esiintyy relaatioita ja kuviossa 21 on esitelty staattisia muuttujia ylläpitävät taulut.



Kuvio 20. Tietokantataulujen relaatiot



Kuvio 21. Tietokantataulut staattisille muuttujille

Tietokantaan palvelin pääsee käsiksi **db/**-kansion kautta. Siellä sijaitsee *db.js*-tiedosto, joka hallitsee kaikkia tietokantayhteyksiä keskitetysti. Kun sovellus käynnistyy, tietokantayhteyksiä varten luodaan ikään kuin ”jono”, joka pystyy käsittelemään useita tietokantapyyntöjä samanaikaisesti. Jonon tai ohjelmointikielellä ”poolin” avulla kaikki tietokantahaut saadaan yhtenäistettyä ja palvelin lähettää pyyntöjä suoraan pooliin eikä tietokantaan eli näin tekemällä saadaan eristettyä tietokanta kaikesta muusta liikenteestä kuin poolin kautta tulevasta, parametrisoidusta, liikenteestä.

```
const pool = mysql.createPool({
  host: DB.host,
  user: DB.user,
  password: DB.password,
  database: DB.name,
  waitForConnections: true,
  connectionLimit: 20,
  queueLimit: 0;});
async function query(sql, params) {
  const [results] = await pool.query(sql, params);
  return results;}
```

Kaikki kyselyt on luotu parametrisoituna, joka estää SQL-injektio-hyökkäysten suorittamisen. Alla on muutamia esimerkkikyselyitä. *db.js*-tiedoston pituus on noin 1100 riviä, joten suurinta osaa kyselyistä ei tässä raportissa käsitellä.

```
async function addRefreshToken(refreshToken, userId, expires) {
  const sql = 'INSERT INTO refresh_tokens (token, userId, expires) VALUES (?, ?, ?)';
  return query(sql, [refreshToken, userId, expires]);
}
```

```
async function createBooking(newBooking) {
  const sql = 'INSERT INTO bookings SET ?';
  const [result] = await pool.query(sql, newBooking);
  return result.insertId;
}
```

```
async function getUserById(id) {
  const sql = 'SELECT id, firstname, lastname, phonenumber, email, area, role,
ergometry_expirary_date, fitness_test_expirary_date, goal  FROM users WHERE id = ?';
  return query(sql, [id]);
}
```

5.2.6 Konfiguraatio ja apuohjelmat

Kokonaisvaltaisissa verkkosovelluksissa on lähes pakollista ylläpitää keskitettyä konfiguraatiota ja hyödyntää apuohjelmia. Keskittämällä konfiguraatiot voidaan niitä muuttaa keskitetysti niin, että koko sovellus siirtyy käyttämään uusia konfiguraatioita, kun ne muokataan keskitettyyn konfiguraatioon.

Palvelimen **config/-**kansio ja sen sisällä *config.js*-tiedosto lukee sovelluksen käynnistyessä ympäristömuuttujat ja asettaa ne konfiguraation arvoiksi. Ympäristömuuttujia ei missään tilanteessa voi paljastaa ulkopuolelle, joten tällainen keskitetty konfiguraatio abstraktoi ko. muuttujien arvot.

```
require('dotenv').config({ path: '../../.env' });

module.exports = {
  PORT: process.env.PORT,
  SECRET_KEY: process.env.SECRET_KEY,
  REFRESH_SECRET_KEY: process.env.REFRESH_SECRET_KEY,
  SMTP_CONFIG: {
    host: process.env.SMTP_HOST,
    port: process.env.SMTP_PORT,
    secure: process.env.SMTP_SECURE_BOOLEAN === 'true',
    auth: {
      user: process.env.SMTP_AUTH_USER,
      pass: process.env.SMTP_AUTH_PASSWORD,
    },
  },
  CORS_ORIGINS: [
    process.env.CORS_ORIGINS,
  ],
  DB: {
    host: process.env.DB_HOST,
    name: process.env.DB_NAME,
    user: process.env.DB_USER,
    password: process.env.DB_PASSWORD,
  },
  CRYPTO_KEY:
    process.env.CRYPTO_KEY,
  TRAIL_API_KEY:
    process.env.TRAIL_API_KEY
};
```

messagesFI.js-tiedoston tarkoitus on muuntaa sovelluksen ilmoitukset suomen kielelle sekä ymmärrettävämpään muotoon käyttäjille. *messagesFI.js*-tiedostoon on myös sisällytetty automaattisissa sähköpostiviesteissä käytettävät viestipohjat, jotka pystyvät hyödyntämään yksilöiviä tietoja, kuten etunimeä ja sukunimeä, jolloin viestejä pystytään kohdentamaan käyttäjälle.

```
success: {
      login: 'Kirjautuminen onnistui',
      userRegistered: 'Käyttäjä rekisteröity onnistuneesti ja vahvistussähköposti
lähetetty!',
      passwordChanged: 'Salasana vaihdettu onnistuneesti',
      loggedOut: 'Uloskirjautuminen onnistui',
      eventAddedSuccess: 'Tapahtuma lisätty onnistuneesti',
      eventUpdatedSuccess: 'Tapahtuma päivitetty onnistuneesti',
      eventRemovedSuccess: 'Tapahtuma poistettu onnistuneesti',
      bookingDeletedSuccess: 'Varaus poistettu onnistuneesti',
      userInfoUpdatedSuccess: 'Käyttäjätiedot päivitetty onnistuneesti',
      notificationPosted: 'Ilmoitus lisätty onnistuneesti',
      notificationDeleted: 'Ilmoitus poistettu onnistuneesti',
      notificationUpdated: 'Ilmoitus päivitetty onnistuneesti',
      confirmationEmailSent: 'Testiaika varattu onnistuneesti. Varausvahvistus
lähetetty',
      messagesNotificationSent: 'Sähköposti-ilmoitukset lähetetty onnistuneesti',
      bookingCreatedButEmailFailed: 'Varaus tehtiin onnistuneesti, mutta
sähköpostivahvistuksen lähetyksessä ilmeni virhe',
      absenceMarked: 'Poissaolo merkitty onnistuneesti ja sähköposti-ilmoitukset
lähetetty',
      testMarked: 'Testi merkattu onnistuneesti'
    },
```

```
email: {
      subject: 'Uusi käyttäjätunnus KSpela FireFit-ajanvarauskalenteriin',
      body: (firstname, lastname, email, password, originUri) => `
        Hei ${firstname} ${lastname},

        Sinulle on luotu uusi käyttäjätunnus Keski-Suomen pelastuslaitoksen FireFit-
ajanvarauskalenteriin.

        Käyttäjätunnuksesi on ${email}
        Salasanasi on ${password}

        Voit nyt kirjautua osoitteessa ${originUri}.
        Pyydämme, että vaihdat oletussalasanan välittömästi ensimmäisen kirjautumisen
yhteydessä
        klikkaamalla oikean yläkulman käyttäjähallinnan painiketta ja valitsemalla Omat
tiedot -> Vaihda salasana.

        Keski-Suomen pelastuslaitoksen uusi FireFit-ajanvarauskalenteri on suunniteltu
kaikille
        sopimuspalokuntalaisille helpottamaan ja yhtenäistämään kuntotestaukseen
liittyvää
        ajanvarausta ja yhteydenpitoa.

        Ystävällisin terveisin,
        Keski-Suomen pelastuslaitos

        Tämä on automaattiviesti, joten älä vastaa tähän viestiin.
        Ongelmatilanteissa ole yhteydessä oman alueesi esihenkilöön tai FireFit-
testaajaan.
      `,
    },
```

Apuohjelmia palvelin hyödyntää **utils/**-kansiosta. Tämän sovelluksen tapauksessa apuohjelmat liittyvät johonkin usein toistuvaan ominaisuuteen tai toiminnallisuuteen, jonka tapauksessa on luontevaa luoda ulkoisia funktioita, joita voi kutsua ohjelmakoodista. Sovelluksen toiminnan kannalta tärkeimpiä näistä ovat *logger.js*, *formatDateTime.js* sekä *reservationNotification.js*-apuohjelmat. *encryptIndex.js* on terveystietoa olevan kuntoisuusindeksin salaukseen käytettävä apuohjelma.

Logger.js

Logger.js on nimensä mukaisesti sovelluksen loggaukseen ja valvontaan tarkoitettu apuohjelma, joka kirjoittaa ulkoisille tiedostoille sovelluksen tapahtumia. Näitä tapahtumia voidaan lukea ja analysoida tiedostoista myöhemmin.

```
// server/utils/logger.js
const { createLogger, format, transports } = require('winston');
const { combine, timestamp, printf, colorize } = format;

const logFormat = printf(({ level, message, timestamp }) => {
  return `${timestamp} [${level}]: ${message}`;
});

const logger = createLogger({
  format: combine(
    colorize(),
    timestamp(),
    logFormat
  ),
  transports: [
    new transports.File({ filename: 'logs/error.log', level: 'error' }),
    new transports.File({ filename: 'logs/combined.log' }),
/*     new transports.Console({
      format: combine(
        colorize(),
        logFormat
      )
    }) */
  ],
});
module.exports = logger;
```

FormatDateTime.js

formatDateTime.js-apuohjelma on päivämäärien ja kellonaikojen muuntamiseen tarkoitettu ohjelma, josta voi kutsua kolmea eri funktiota. Tarve tällaiselle ohjelmalle on, koska sovelluksen käyttämä MariaDB:n instanssi sovelluksen domainilla toimii vain GMT +0 aikavyöhykkeellä eli kaikki ajat, joita kyseinen instanssi tarjoaa, on GMT+0 ajassa. Tästä syystä kellonaikoja ja päivämääriä joudutaan muokkaamaan monessa sovelluksen osassa.

```
// Function to format datetime to YYYY-MM-DD HH:MM:SS
const formatDateTime = (dateString) => {
    const date = new Date(dateString);
    const year = date.getFullYear();
    const month = String(date.getMonth() + 1).padStart(2, '0');
    const day = String(date.getDate()).padStart(2, '0');
    const hours = String(date.getHours()).padStart(2, '0');
    const minutes = String(date.getMinutes()).padStart(2, '0');
    const seconds = String(date.getSeconds()).padStart(2, '0');
    return `${year}-${month}-${day} ${hours}:${minutes}:${seconds}`;
  };
  const formatDate = (dateTime) => {
    const date = new Date(dateTime);
    const day = String(date.getDate()).padStart(2, '0');
    const month = String(date.getMonth() + 1).padStart(2, '0');
    const year = date.getFullYear();
    return `${day}.${month}.${year}`;
  };
  const formatTime = (dateTime) => {
    const date = new Date(dateTime);
    const hours = String(date.getHours()).padStart(2, '0');
    const minutes = String(date.getMinutes()).padStart(2, '0');
    return `${hours}:${minutes}`;
  };
  module.exports = { formatDateTime, formatDate, formatTime };
```

ReservationNotifications.js

ReservationNotifications.js-tiedosto on niin kutsuttua cronjobia ajava ohjelma, joka muistuttaa käyttäjiä tulevista varauksista 24 h ja 4 päivää aikaisemmin. Kun sovellus käynnistyy, tämä niin kutsuttu cronjob käynnistyy ja tarkastaa tietokannasta varaukset, joiden aloitusaika on annettujen parametrien sisällä. Jos sellaisia varauksia löytyy niin ohjelma lähettää sähköpostimuistutuksen sekä kirjaa tietokantaan, että muistutus on lähetetty, jolloin uutta vastaavaa muistutusta ei enää lähetetä. Alla olevasta koodilohkosta on jätetty funktiot pois niiden laajuuden vuoksi.

```
const setupCronJob = (next) => {
  cron.schedule('0 * * * *', async () => {
    try {
      await sendNotifications24hours();
      await sendNotifications4days();
    } catch(error) {
      next(error)
    };
  });
};
```

5.3 Käyttöliittymä

5.3.1 Rakenne

Sovelluksen käyttöliittymä noudattaa modulaarista komponenttipohjaista arkkitehtuuria, joka on tyypillistä moderneille React-sovelluksille. Jokainen käyttöliittymän osa on jaettu selkeästi määriteltyihin komponentteihin, mikä parantaa uudelleenkäytettävyyttä, ylläpidettävyyttä sekä komponenttien vastuiden eriyttämistä. Alla on yksityiskohtainen kuvaus käyttöliittymän rakenteesta. Monista alaotsikoista jätetään koodilohkot pois, koska useimmat käytetyistä komponenteista ja rakenteista ovat niin laajoja kokonaisuuksia, että niitä ei ole mielekästä esittää tässä raportissa. Osaa komponenteista havainnollistetaan myös kuvioin.

src/components/-kansio sisältää alikansioita ja tiedostoja, jotka edustavat eri käyttöliittymän osia. Jokainen alikansio vastaa yleensä tietystä käyttöliittymän toiminnallisuudesta tai osiosta. Modulaaristen komponenttien käyttö tekee sovelluksesta skaalautuvan ja helposti ylläpidettävän. Keskeisen alikansiot ovat:

- **API/**: Sisältää komponentit, jotka liittyvät pyyntöihin palvelimelle. Kaikki käyttöliittymän liikenne ohjataan näiden komponenttien kautta.
- **Dashboard/**: Tässä kansiossa on komponentit, jotka muodostavat pääkäyttäjille ja esihenkilöille suunnatun hallintapaneelin.
- **Layout/**: Tämä kansio sisältää komponentteja, jotka vastaavat sovelluksen yleisestä asettelusta ja rakenteesta, kuten navigointipalkista, otsikkoelementeistä ja muista käyttöliittymäelementeistä.
- **Loading/**: Näitä komponentteja käytetään visualisoimaan käyttäjälle odotusaikaa, kun käyttöliittymä hakee sovelluksen tarvittavia tietoja sovelluksen latautumisen yhteydessä.
- **Login/**: Sovelluksen käyttöliittymän kirjautumiseen käyttämät komponentit löytyvät tästä kansiosta.
- **Register/**: Sovellukseen rekisteröimiseen käytettävät komponentit löytyvät tästä kansiosta.
- **UserView/**: Tämä kansio pitää sisällään normaalin käyttäjän käyttöliittymän komponentit.
- **utils/**: Utils kansio pitää sisällään apuohjelmia, joita käytetään yhtenäistämään palvelimen lähettämää viestintää käyttöliittymään sekä laskemaan tai tarkastamaan jotain useasti tapahtuvaa toimintoa käyttöliittymässä.

src/slices/-kansio on koko sovelluksen istunnonhallinnan kannalta hyvin tärkeä osa ja se pitää sisällään istunnonhallinnassa käytettävän *Redux Store*-kirjaston loogiset osat. Loogiseen osaan kuuluu kolme keskeistä elementtiä: *Initial State, Reducer, Action* ja tämä sovellus käsittelee kahta eri

loogista osaa eli varauksiin liittyvää osaa sekä käyttäjiin liittyvää osaa. Käyttäjän kirjautuessa sisään, käyttäjätiedot haetaan ja tallennetaan palvelimelta käyttöliittymän loogiseen userSlice-osaan, jonka kautta kaikki käyttöliittymän osat voivat hyödyntää tallennettuja tietoja.

```
interface UserState {
  id: string;
  firstname: string;
  lastname: string;
  email: string;
  role: string;
  area: string;
  phonenumber: string;
  ergometry_expirary_date: string;
  fitness_test_expirary_date: string;
  oulunmalli_expirary_date: string;
  mobility_test_expirary_date: string;
  goal: string;
  fire_station: string;
}
```

Varauksiin liittyvään osaan tallennetaan kirjautumisen ja käyttäjätietojen ohella tiedot kaikista tehdyistä varauksista, siten, että vain käyttäjän itsensä varaamat tiedot ovat hänelle itselleen näkyvillä.

```
interface Booking {
  id: string;
  start: string;
  end: string;
  title: string;
  area: string;
  userId: string;
  userRole: string;
  firstname: string;
  lastname: string;
  phonenumber: string;
  fire_station: string;
  tester: string;
  approved: number;
}
```

Store.ts ja encryptedLocalStorage.tsx

Store.ts sekä encryptedLocalStorage.tsx liittyvät edellä mainittuihin sliceihin sitten, että Store.ts kokoaa kaikki loogisiin osiin tehtävät toiminnot yhteen ja ottaa ne käyttöön universaalisti koko sovelluksessa ja encryptedLocalStorage.tsx-tiedostoa käytetään salaamaan Store.ts:n tallentamat tiedot selaimen paikallismuistiin.

```
// Combine your reducers
const rootReducer = combineReducers({
  bookings: bookingReducer,
  user: userReducer,
});

// Configure persist
const persistConfig = {
  key: 'root',
  version: 1,
  storage: encryptedLocalStorage,
};

const persistedReducer = persistReducer(persistConfig, rootReducer);

// Create the store
export const store = configureStore({
  reducer: persistedReducer,
  middleware: (getDefaultMiddleware) =>
    getDefaultMiddleware({
      serializableCheck: {
        ignoredActions: [FLUSH, REHYDRATE, PAUSE, PERSIST, PURGE, REGISTER],
      },
    }),
});
```

App.tsx

App.tsx on React-komponentti, joka toimii sovelluksen ytimenä. Tämä tiedosto yhdistää erilaiset osat yhteen, kuten reitityksen, tilanhallinnan ja muut tarvittavat kontekstit, kuten näytön kokoon mukautuminen. React-komponentit muodostavat koko sovelluksen käyttöliittymän, ja App-komponentti on yleensä ylin taso, josta kaikki muut komponentit saavat tarvittavat resurssit ja logiikan.

Sovellus käyttää *react-router-dom*-kirjastoa reitittämään komponenttien väliset yhteydet. Tämä mahdollistaa esimerkiksi etusivun, kirjautumissivun tai hallintapaneelin renderöimisen tietyin ehdoin. Eri näkymiä ja komponentteja pystytään reitittämään kuten URL-polkua käytettäessä.

```
<Provider store={store}>
  <PersistGate loading={null} persistor={persistor}>
    <ScreenSizeProvider>
      <Router>

        <Routes>
          <Route path="/login" element={<Login />} />
          <Route
            path="/"
            element={
              <AuthChecker>
                <Homepage />
              </AuthChecker>
            }
          />
          <Route path="/dashboard" element={
            <AuthChecker>
              <Dashboard />
            </AuthChecker>
          } />
        </Routes>
      </Router>
    </ScreenSizeProvider>
  </PersistGate>
</Provider>
```

App.tsx-komponentin rakenne muodostuu seuraavista osista:

- **Provider**: Tämä komponentti tarjoaa koko sovellukselle pääsyn Redux-tilaan, joka tallennetaan edellä läpikäytyyn store.ts-komponenttiin, jonka tilaa ja logiikkaa pystytään muokkaamaan kaikkialta sovelluksesta.
- **PersistGate**: Tätä komponenttia käytetään ylläpitämään Redux-tilaa tallennettuna läpi istuntojen sekä varmistamaan, että tarvittavat tiedot ja data on latautunut ennen kuin muut komponentit renderöidään.
- **Router ja Routes:** Julistaa koko sovelluksen laajuisen reititystopologian, jota pystytään käyttämään ja hyödyntämään reiteillä eli Routes.
- **AuthChecker**: Tämä komponentti toimii middleware-ohjelmana käyttöliittymässä tarkistamaan onko kirjautumistapahtuma onnistunut vai ei ja sen perusteella renderöi käyttäjälle joko normaalin näkymän tai hallintapaneelinäkymän.
- **ScreenSizeProvider**: Tämä komponentti tarjoaa koko sovelluksen käyttöön käyttäjän näyttökoon, jota pystytään hyödyntämään kun tarjotaan responsiivistä sisältöä.

5.3.2 Layout

Layout/-kansiosta löytyy pääsääntöisesti sovelluksen käyttöliittymän asetteluun liittyvät komponentit sekä komponentteja, jotka eivät muuten sovi muihin kansiorakenteisiin. Kehityksessä ajatuksena oli se, että layout-rakenne toimii kulhona, johon muut komponentit kasataan. Layout-komponenttien tyylimääritelmät on myös tehty vaatimusmäärittelyn mukaisista mockup-suunnitelmista johdettuna. Monista alaotsikoista jätetään koodilohkot pois, koska useimmat käytetyistä

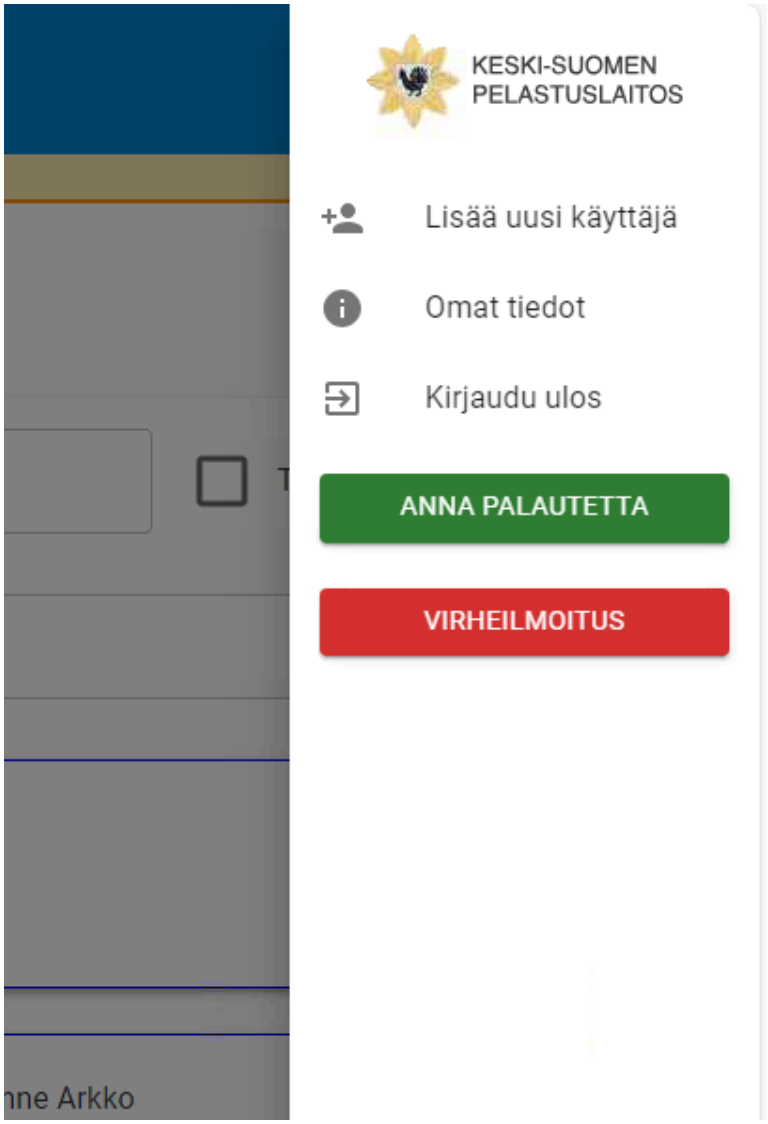
komponenteista ja rakenteista ovat niin laajoja kokonaisuuksia, että niitä ei ole mielekästä esittää tässä raportissa. Osa havainnollistavista kuvioista löytyy liitteestä 1.

Layout.tsx-tiedosto toimii koko layout-rakenteen isäntänä, joka kasaa muut kaikissa käyttöliittymissä ja näkymissä yhteisenä toimivat komponentit yhteen paikkaan. Näitä yhteisiä komponentteja ovat esimerkiksi *HeaderBar.tsx***,** *UserInfo.tsx* tai *ChangePasswordModal.tsx*. Layout.tsx-tiedoston ydin on kuitenkin Box-komponentissa, joka renderöi kaikki *{children}* -komponentit. Tähän komponenttiin pystytään siis renderöimään mitä vain muita komponentteja:

```
<Box component="main" sx={{ flexGrow: 1, width: '100%' }}>
   {children}
</Box>
```

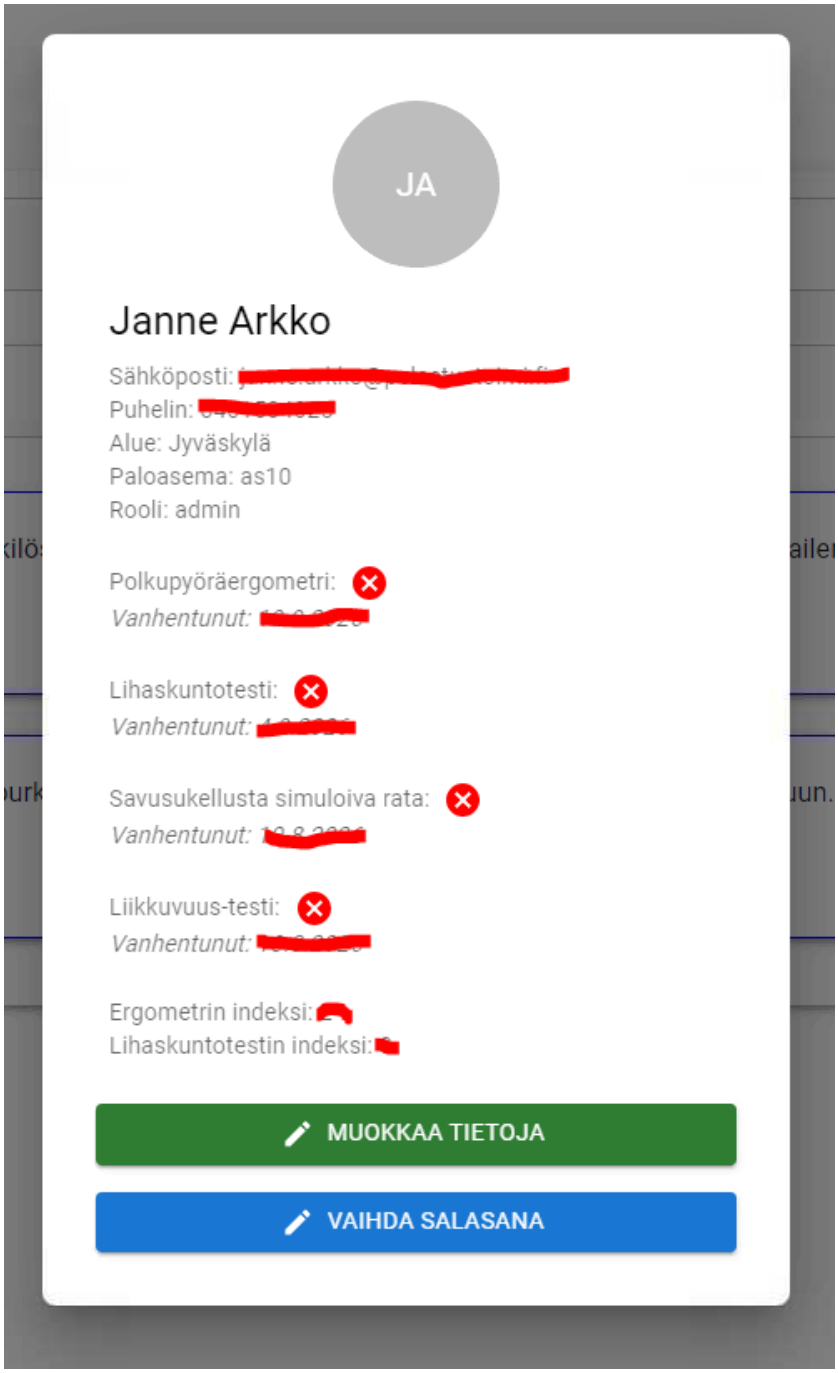
HeaderBar.tsx-tiedosto kerää yhteen sovelluksen yläreunassa sijaitsevan navigaatio- tai otsikkokomponentin ja sen ylläpitämät toiminnot. Tärkeimpinä komponentteina toimii sivulta aukeava menu, joka pitää sisällään käyttäjäkeskeiset toiminnot sekä palautejärjestelmän toiminnot (kts kuvio 22). Tuohon laatikkoon pystytään renderöimään eri rooleille erilaisia toimintoja, kuten kaikille muille rooleille paitsi testattavat-roolille löytyy uuden käyttäjän lisäämiseen toiminto. *HeaderBar*-komponentti renderöidään muutoin täysin samanlaisena kaikille käyttöliittymän osille.

```
<Drawer anchor="right" open={drawerOpen} onClose={toggleDrawer(false)}>
    {drawerList}
</Drawer>
<ErrorReportModal
    isMobile={isMobile}
    isOpen={errorModalOpen}
    onClose={() => setErrorModalOpen(false)}
/>
<FeedbackModal
    isMobile={isMobile}
    isOpen={feedbackModalOpen}
    onClose={() => setFeedbackModalOpen(false)}
/>
```



Kuvio 22. Navigaatio- ja sivupalkki

UserInfo.tsx-komponentti (kts. kuvio 23) pitää sisällään kaiken toiminnallisuuden ja tyylit, joita käytetään käyttäjätietoikkunassa. Tämä komponentti pyytää palvelimelta rajapinnasta */getValidTestDatesAndIndex/:id* käyttäjäkohtaiset kuntotestaustiedot ja esittää muutoin Redux-tilanhallintaan tallennetut käyttäjätiedot. **EditUserInfo.tsx**-komponentti on oma komponenttinsa, mutta liittyy vahvasti tämän komponentin toimintaan. Kyseisen komponentin kautta käyttäjä pystyy muokkaamaan omia tietojaan.



Kuvio 23. UserInfo-komponentti

ChangePasswordModal.tsx-komponentti nimensä mukaisesti on tarkoitettu salasanan vaihtoon. Komponentti kutsuu palvelimen rajapintaa */change-password*, ja näyttää palvelimen vastauksen.

```
    <Dialog open={isOpen} onClose={onRequestClose} maxWidth="sm" fullWidth>
      <DialogTitle>Vaihda salasana</DialogTitle>
      <DialogContent>
        <form onSubmit={handleChangePassword}>
          <TextField
            label="Vanha salasana"
            type="password"
            value={oldPassword}
            onChange={(e) => setOldPassword(e.target.value)}
            required
            fullWidth
            margin="normal"
          />
          <TextField
            label="Uusi salasana"
            type="password"
            value={newPassword}
            onChange={(e) => setNewPassword(e.target.value)}
            required
            fullWidth
            margin="normal"
          />
          {message && (
            <Typography color={isError ? 'error' : 'success'} sx={{ mt: 2 }}>
              {message}
            </Typography>
          )}
            <Box sx={{ mt: '10px', display: 'flex', justifyContent: 'space-between',
width: '100%' }}>
              <Button type="submit" variant="contained" color="success">Vaihda</Button>
              <Button onClick={onRequestClose} color="error">Peruuta</Button>
            </Box>
        </form>
      </DialogContent>
    </Dialog>
```

5.3.3 API-komponentti

API/-kansiosta löytyy **API.tsx**-tiedosto, joka toimii käyttöliittymän niin sanottuna reitittimenä. Kaikki käyttöliittymän lähettämät kyselyt ja pyynnöt menevät aina API-komponentin kautta. Tämän komponentin ideana on yhtenäistää palvelinpyyntöjen rakenne sekä keskittää tietoturvallisuuden aspektit samaan komponenttiin.

Ensimmäisellä palvelinpyynnöllä API-komponentti pyytää automaattisesti palvelinta lähettämään CSRF-avaimen, joka sitoo sen hetkisen istunnon käyttäjän selaimeen, vahvistaen tietoturvaa. Muutoin komponentti liittää käyttäjäkohtaisen kirjautumisavaimen palvelinpyyntöihin autentikaatiota varten sekä tarvittaessa uusii avaimen käyttäen uusiutumisavainta.

API-komponentti siis automatisoi ja abstraktoi käyttöliittymältä palvelimelle kohdistuvan liikenteen ja helpottaa rajapintojen kutsumista sekä tietoturvan toteutumista. Alla olevassa koodilohkossa pyydetään palvelimelta ilmoitustaulun ilmoitukset ja pyyntö tehdään kutsumalla API-komponenttia ja antamalla kutsuttava rajapinta sekä mahdolliset parametrit.

```
  const fetchNotifications = async (area: string, role: string, fireStation: string) => {
    try {
      const response = await API.get('/getMessages', { params: { area, role, fireStation
} });
      setNotifications(response.data);
    } catch (error) {
      setSnackbarMessage('Error fetching notifications');
      setSnackbarSeverity('error');
      setSnackbarOpen(true);
    }
  };
```

5.3.4 Hallintapaneeli

Dashboard/-kansiosta löytyy pääkäyttäjä- ja esihenkilönäkymään renderöitävän käyttöliittymän komponentit. Komponentit on jaettu välilehdittäin henkilöstöön, varauksiin, suunnitteluun, statistiikkaan sekä polkupyörien sijainteihin. Jokaisen kansion alta löytyy siihen välilehteen käytetyt komponentit.

Dashboard.tsx-komponentti toimii isäntänä muille käyttöliittymän komponenteille ja kun komponentti latautuu, se hakee palvelimelta suurimman osan käyttöliittymän hyödyntämästä ja renderöimästä datasta sekä jakaa sitä alaspäin muille komponenteille. Dashboard-komponentti hallinnoi välilehtirakennetta ja käyttäjän valintojen perusteella lataa muita alakomponentteja käyttöliittymään. Alla olevassa koodilohkossa on esitelty välilehtien renderöinti.

```
const renderTabs = () => (
  <Tabs
    value={selectedTab}
    onChange={handleTabChange}
    orientation={isMobile ? 'vertical' : 'horizontal'}
    variant={isMobile ? 'scrollable' : 'standard'}
    style={{ marginBottom: '16px' }}
  >
    <Tab label="Statistiikka" />
    <Tab label="Henkilöstö" />
    <Tab label="Varaukset" />
    <Tab label="Suunnittelu" />
    <Tab label="Pyörien sijainnit" />
    <Tab label='Ilmoitustaulu' />
  </Tabs>
);
```

Statistiikka

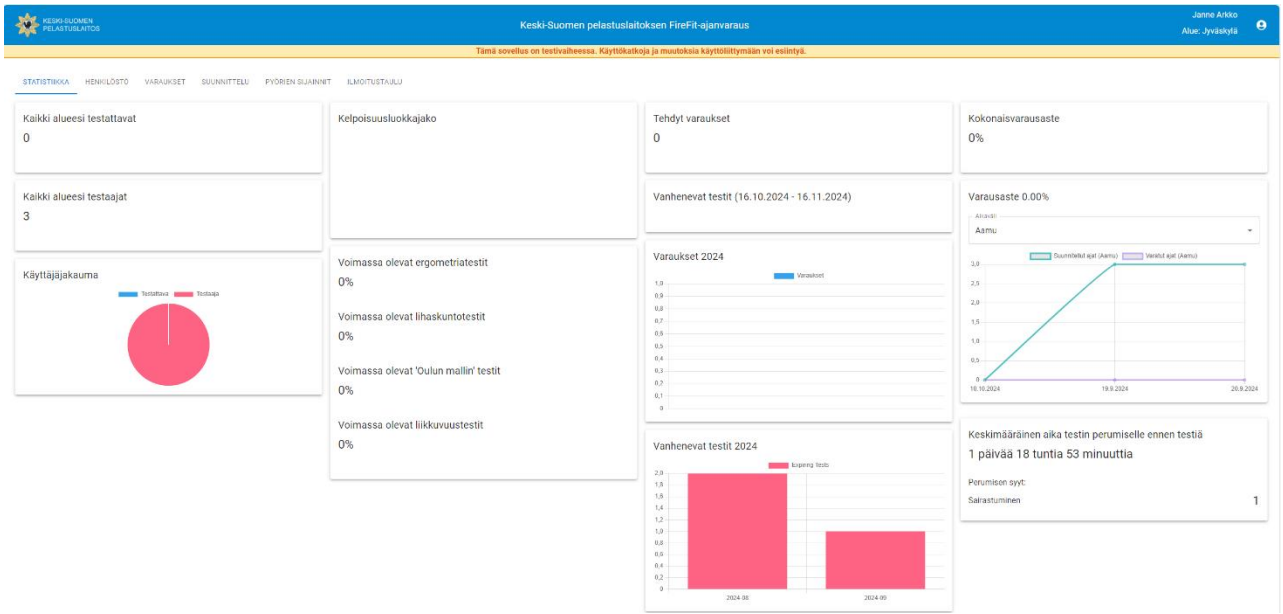
Statistics/-kansio pitää sisällään kaikki Statistiikka-välilehdelle renderöitävät komponentit. Jokainen välilehden osa on oma komponenttinsa ja hyödyntää palvelimelta haettua statistiikka-dataa hieman eri tavoin. Statistiikka-välilehti on esitelty kuviossa 24.

- **DashboardTabOverview.tsx** toimii kokoojakomponenttina ja se tarjoaa välilehdelle sen käyttämän rungon eli lohkot joihin eri komponentit renderöidään. Alla olevassa koodissa on esitelty yksi lohko, johon renderöidään eri komponentteja käyttämällä Grid-komponentteja.

```
{/* Middle Right Section */}
<Grid item xs={12} md={3}>
    <Grid container spacing={2}>
        <Grid item xs={12}>
           <UniversalStatsCard
              items={[{ title: "Tehdyt varaukset", value:
statistics.totalBookings.toString() }]}
              />
        </Grid>
        <Grid item xs={12}>
           <TestExpiryAlerts alerts={statistics.expiryAlerts} />
        </Grid>
        <Grid item xs={12}>
           <BookingTrends labels={statistics.bookingTrends.labels}
data={statistics.bookingTrends.data} />
        </Grid>
        <Grid item xs={12}>
           <TestExpiryTrends labels={statistics.expiryTrends.labels}
data={statistics.expiryTrends.data} />
        </Grid>
     </Grid>
</Grid>
```

- **BookingTrendsTable.tsx** renderöi varaustrendejä esittelevän taulukon eli vuositasolla kuukausittaisen varauslukumäärän.
- **BookingVsAvailableLineGraph.tsx** renderöi varausasteita esittelevän komponentin eli suunniteltujen testauspäivien tuntimäärien ja varattujen testausaikojen suhteen.

- **TestExpiryAlerts.tsx**-komponentille renderöidään vanhaksi menevät testit.
- **TestExpiryTrendsTable.tsx**-komponentti renderöi vanhaksi menevien testien trendit helpottamaan testausaikojen suunnittelua kuukausille, joilla on eniten vanhenevia testejä.
- **UniversalStatsCard**.tsx käytetään renderöimään mitä tahansa pientä tietoa esimerkiksi lukumääriä tai prosentteja.
- **UserDistributionTable**.tsx käytetään näyttämään ympyräkaaviossa testattavien ja testaajien lukumäärät.
- **UserIndexDistributionTable**.tsx renderöi ympyräkaavion testattavien kelpoisuusluokituksista.



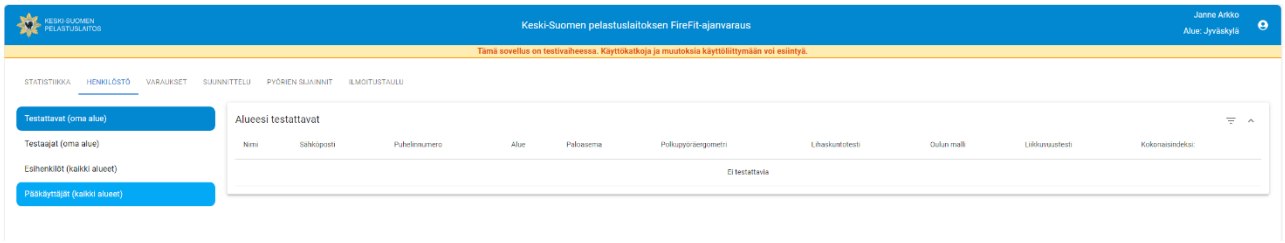
Kuvio 24. Statistiikka-välilehti

Henkilöstö

Users/-kansiosta löytyy Henkilöstö-välilehden (kts. kuvio 25) käyttämät komponentit. Näiden komponenttien rakenne vastaa muita komponentteja ja kansiosta löytyy jälleen isäntäkomponentti, joka antaa rungon ja datan muille komponenteille.

DashboardTabUsers.tsx toimii isäntäkomponenttina ja tarjoaa rungon muille komponenteille sekä saa käyttämänsä ja alakomponenteille jakamansa datan **Dashboard.tsx**-komponentilta ja lisäksi se hakee käyttäjän roolin perusteella täydentävää dataa renderöitäväksi.

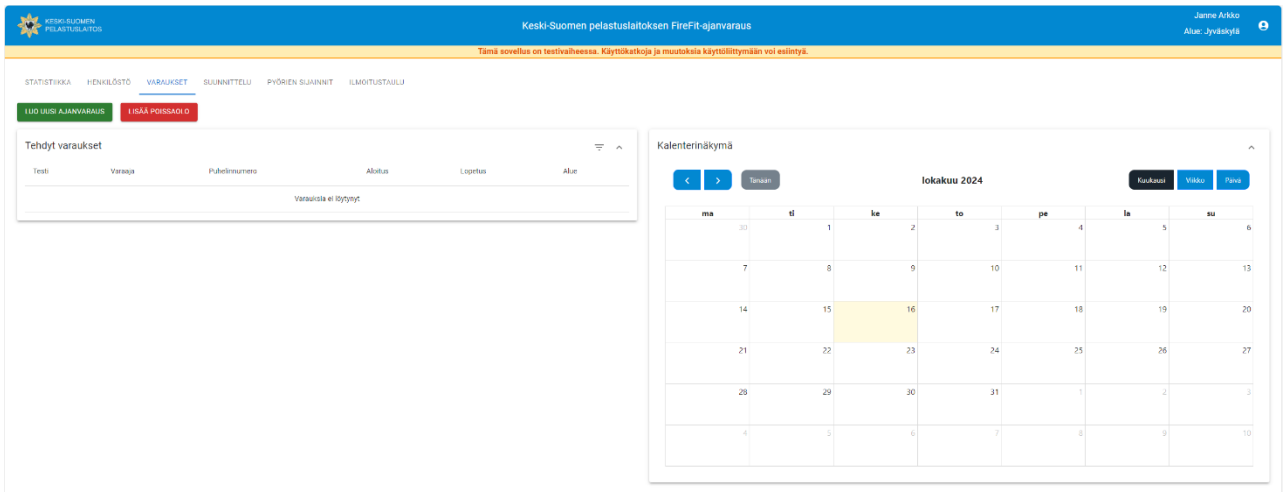
RegularUsersTable.tsx, TestersTable.tsx, SupervisorsTable.tsx sekä **AdminsTable.tsx** renderöivät kaikki eri käyttäjärooleihin sidotut käyttäjät joko alueellisesti tai koko sovelluksen laajuudelta pääkäyttäjille.



Kuvio 25. Henkilöstö-välilehti

Varaukset

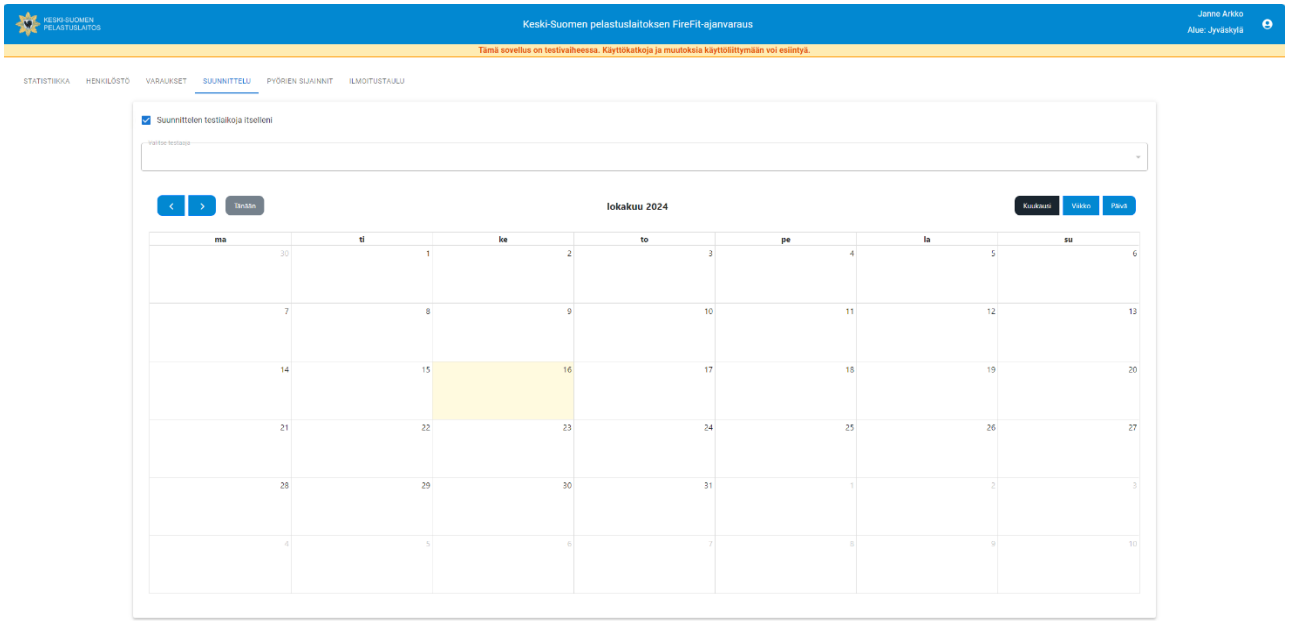
Bookings/-kansio pitää sisällään varauksiin liittyvät komponentit pääkäyttäjä- ja esihenkilönäkymässä. Kansiosta löytyy isäntäkomponentti **DashboardTabBookings.tsx** sekä käyttöliittymän Varaukset-välilehdelle (kts. kuvio 26) renderöitävät varaustaulukko sekä kalenterikomponentti. Myös poissaolojen merkkaamiseen tarkoitettu modaali löytyy tästä kansiosta.



Kuvio 26. Hallintapaneelin Varaukset-välilehti

Suunnittelu

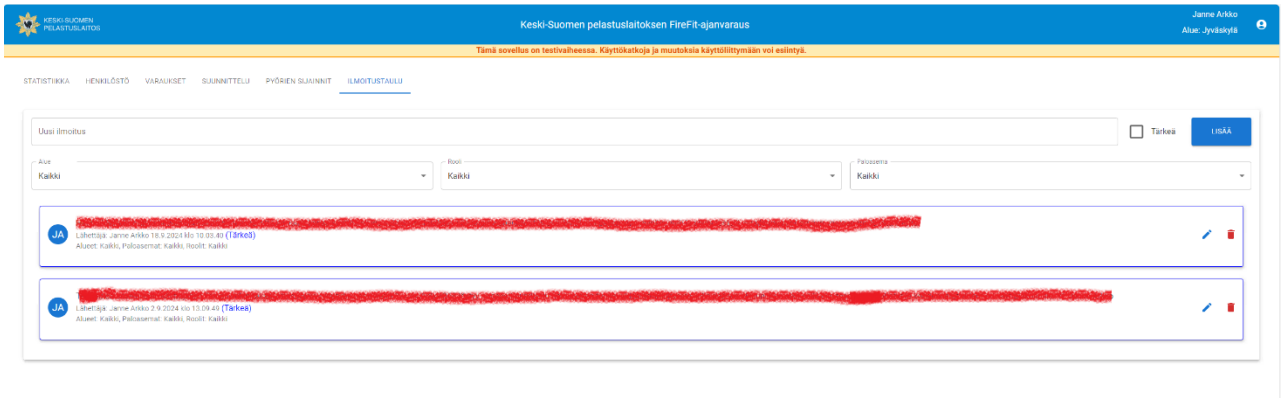
Planning/-kansiosta löytyy käyttöliittymän Suunnittelu-välilehden komponentit ja toiminnallisuudet. Komponenttien rakenne ja hierarkia noudattaa samaa kaavaa kuin aiemmin eli kansiosta löytyy isäntäkomponentti, joka on vastuussa muiden komponenttien renderöimisestä käyttöliittymään. Suunnittelu-välilehti on esitelty alla olevassa kuviossa 27.



Kuvio 27. Suunnittelu-välilehti

Ilmoitustaulu

NotificationBoard/-kansiosta löytyy yksittäinen tiedosto **NotificationBoard.tsx**, joka on vastuussa kaikesta Ilmoitustaulu-välilehden (kts. kuvio 28) tapahtumista. Ilmoitustaulun toiminta perustuu yksinkertaisesti siihen, että tietokantaan tallennetaan tehdyt ilmoitukset ja ilmoituksille voi antaa parametrejä, joiden perusteella ilmoituksen renderöidään per käyttäjä eli ilmoitustaulu toimii myös eräänlaisena tiedotuskanavana tarvittaessa rajoitetulle yleisölle. Käyttäjänäkymään renderöidään samanlainen ilmoitustaulu ilman mahdollisuutta luoda uusia ilmoituksia.



Kuvio 28. Ilmoitustaulu-välilehti

5.3.5 Käyttäjänäkymä

UserView/-kansio kerää sovelluksen toisen käyttöliittymän eli niin kutsutun käyttäjänäkymän komponentit samaan kansioon. Kansion rakenne ja hierarkia ei poikkea aiemmin esitetyn hallinta-paneelin toiminnasta ja komponentit ovat jälleen rakennettu puumaisesti isäntäkomponenttien alle.

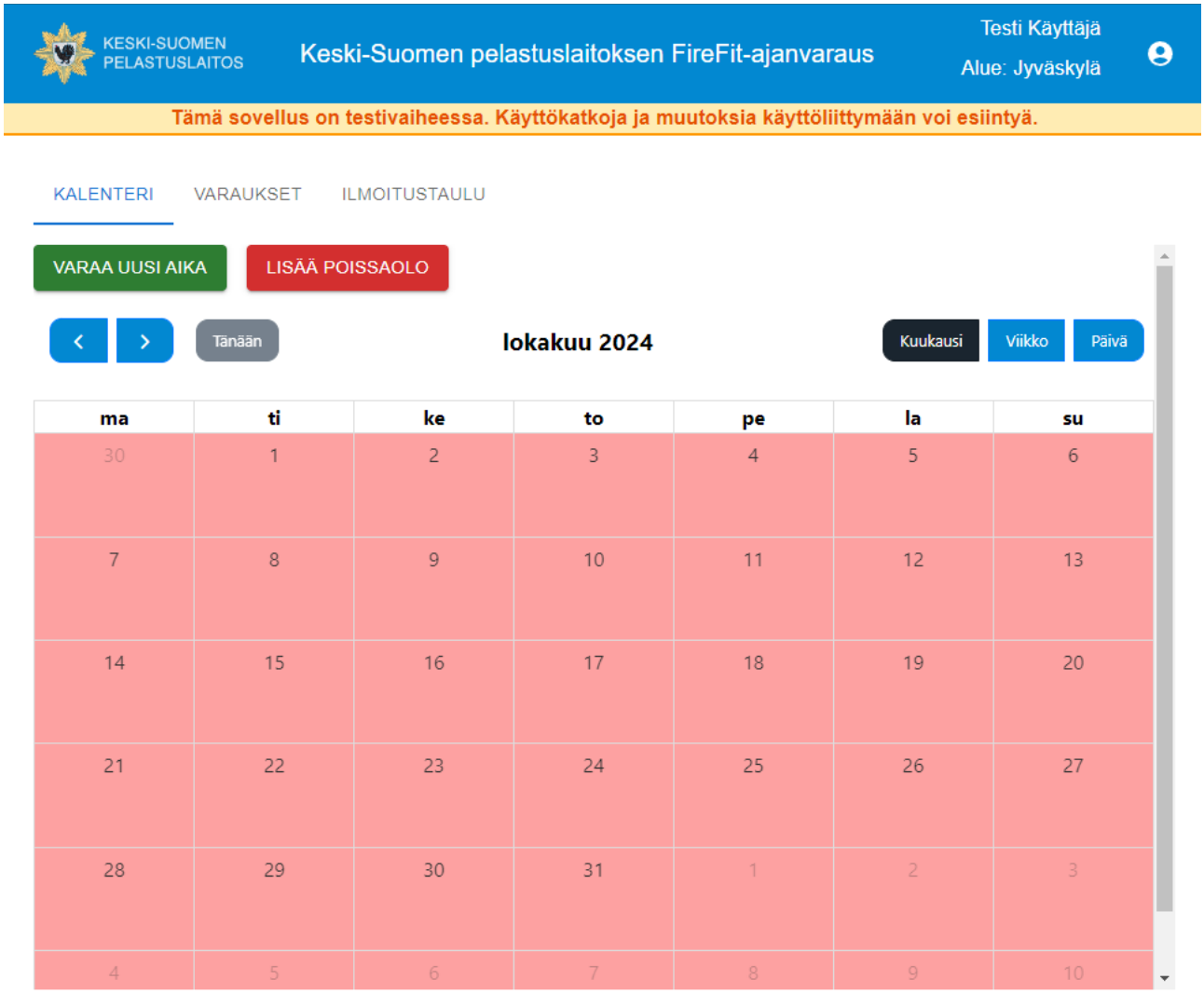
UserViewDashboard.tsx-tiedosto toimii käyttäjänäkymän isäntäkomponenttina ja on vastuussa välilehtien renderöimisestä ja niiden arvojen näyttämisestä sekä sisällön vaihtamisesta käyttöliitty-mään perustuen käyttäjän rooliin tai toimintaan sovelluksessa.

Kalenteri

Calendar/-kansio pitää sisällään käyttäjänäkymää dominoivan komponentin ja sovelluksen sielun eli ajanvarauskalenterin. Kalenteri on toteutettu käyttämällä *FullCalendar*-moduulia sekä rakenta-malla omia uniikkeja tapahtumatyylejä kalenterin käytettäväksi. Kalenteri-välilehti on esitelty kuvi-ossa 29.

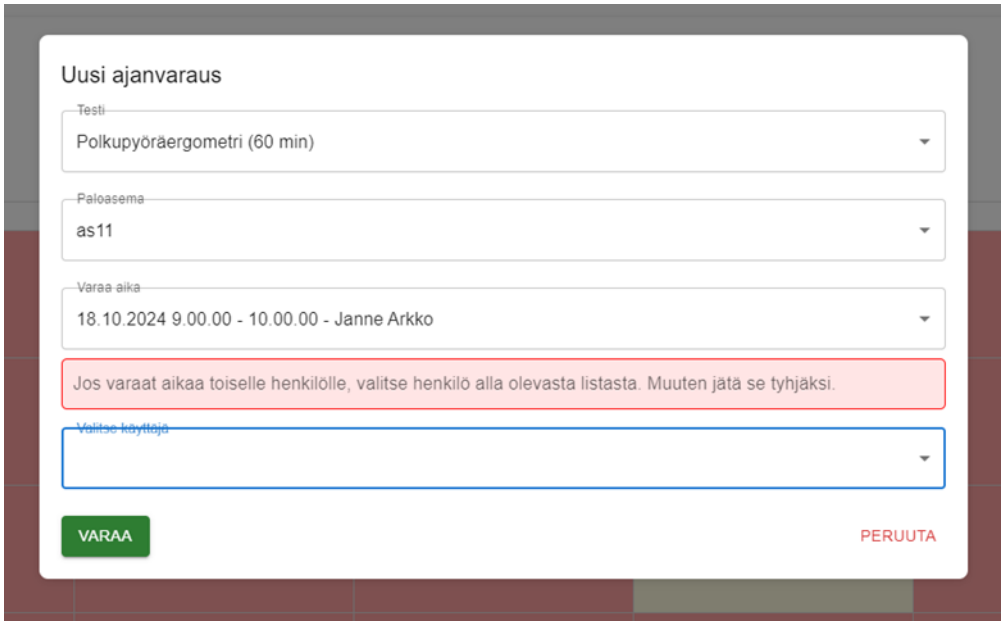
BookingCalendar.tsx-tiedosto renderöi itse kalenterin kaikkine toimintoineen. Kalenterin data haetaan kirjautumisen yhteydessä palvelimelta, tallennetaan Redux-storeen selaimen välimuistiin ja luetaan kalenterin toimesta sieltä. Tämä komponentti vastaanottaa sekä tehdyt varaukset, että

suunnitellut testauspäivät ja jakaa tätä dataa eteenpäin esimerkiksi varauksista vastaavalle komponentille sekä renderöi vapaat testausajat visuaalisesti kalenteriin.



Kuvio 29. Kalenteri-välilehti

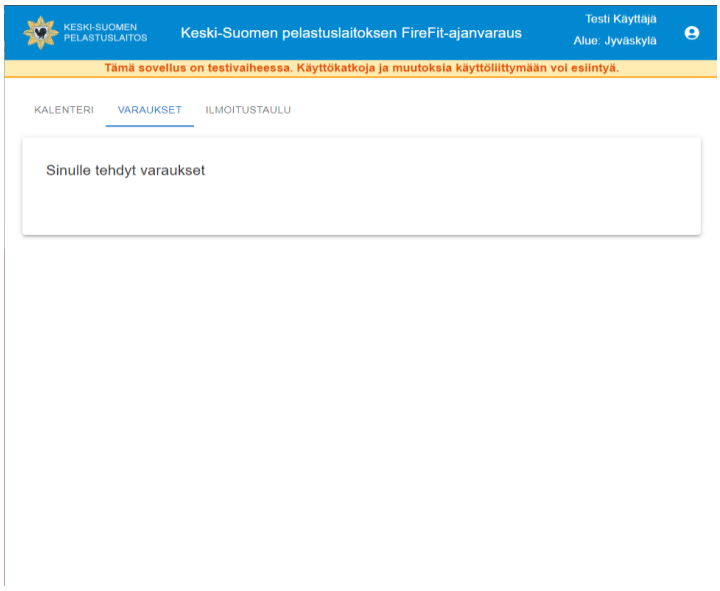
BookingFormModal.tsx on komponentti, jonka kautta käyttäjät hoitavat kaiken varaustoiminnan sekä käyttäjänäkymässä, että hallintapaneelinäkymässä (kts. kuvio 30). Koko sovelluksen varauksiin liittyvä logiikka on rakennettu tämän komponentin sisään. Tällaisia loogisia ominaisuuksia on muun muassa tiettyjen varausten teko vain tietyille paloasemille, joilla on esimerkiksi kuntotestaukseen käytettävä pyörä. Eli käyttäjät eivät voi varata aikaa polkupyöräergometria-testiin asemalle, jolta ei pyörää löydy, vaikka kyseiselle asemalle olisi suunniteltu testausaikoja.



Kuvio 30. Käyttäjän uusi ajanvaraus

Varaukset

Reservations/-kansio ja sen sisällä oleva **BookingItem.tsx**-tiedosto käytännössä renderöivät vain joko käyttäjäkohtaiset tai testaajakohtaiset varaustiedot Varaukset-välilehdelle. Kuviossa 31 on esitelty Varaukset-välilehti.



Kuvio 31. Käyttäjänäkymän Varaukset-välilehti

5.3.6 Kirjautumissivu

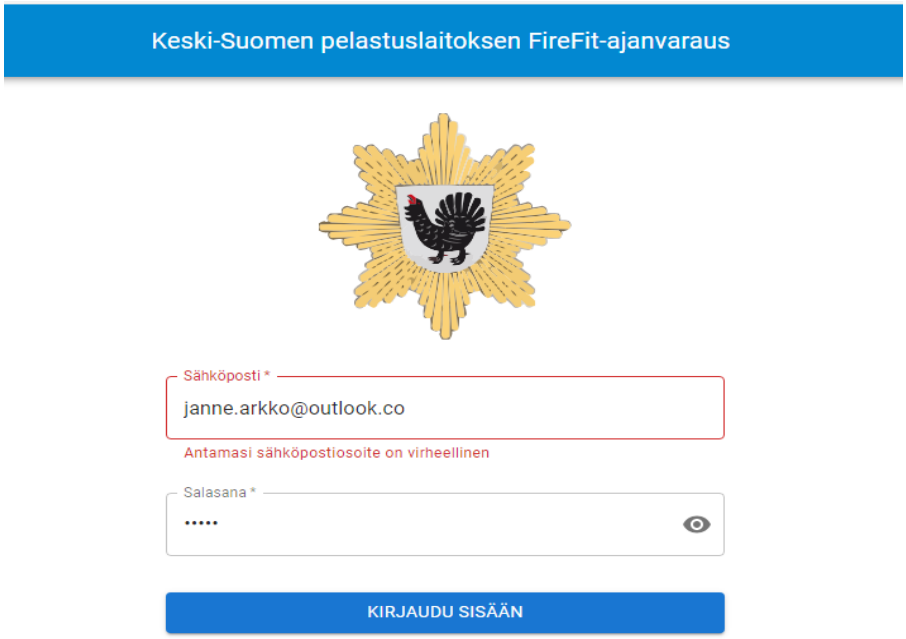
Niin kutsuttu laskeutumissivu tai englanniksi ”Landing page” eli kirjautumissivu on rakennettu sovellukseen yksittäisen komponentin kautta säilyttäen yksinkertainen muotoilu ja toiminta. Käyttöliittymän middleware-ohjelma ohjaa käyttäjän aina tähän komponenttiin, jos muiden komponenttien autentikaatio ei onnistu.

Login.tsx-komponentti renderöi syötealueet käyttäjätunnukselle ja salasanalle. Kun käyttäjä yrittää kirjautua, komponentti lähettää palvelimen kirjautumisesta vastaavaan rajapintaan pyynnön ja jos se menee läpi niin palvelin vastaa käyttäjän tiedoilla, jotka tallennetaan salattuna selaimen paikallismuistiin, tämä on ensimmäinen linkki ketjussa, kun käyttäjätietoja palvelimelta lähetetään käyttöliittymään. Kirjautumisen syötealueet on esitelty kuviossa 32.

Kirjautumisen logiikkaan on rakennettu erilaisia sitä tukevia keinoja, esimerkiksi palvelimen palautteen perusteella voidaan renderöidä erilaisia tekstejä ja korostaa esimerkiksi salasanan syötekenttää, jos salasana on kirjoitettu esimerkiksi väärin. Virheenhallinnan logiikkaa on esitelty kuviossa 33.

```
if (error.response) {
        const { message } = error.response.data;
        setMessage(message);
        if (message === 'Antamasi sähköpostiosoite on virheellinen') {
          setEmailError(true);
        } else if (message === 'Antamasi salasana on virheellinen') {
          setPasswordError(true);
        } else {
          setOpenSnackbar(true);
        }
```

Kuvio 32. Kirjautumissivu



Kuvio 33. Virheenhallinta kirjautumissivulla

5.3.7 Middleware ja apuohjelmat

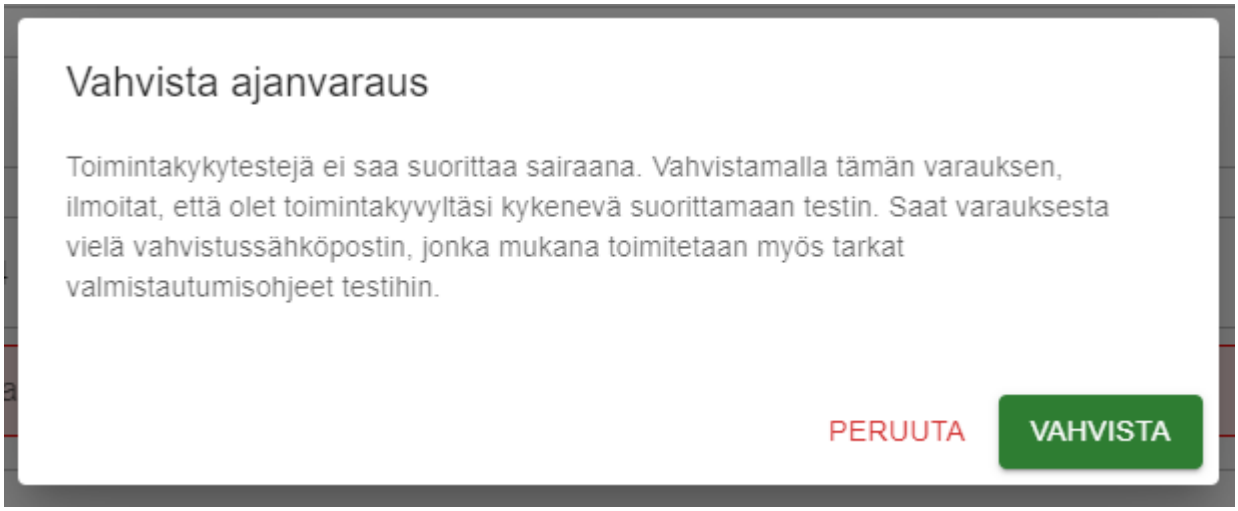
Käyttöliittymän **middleware/**-kansiossa sijaitsee **AuthChecker.tsx**-komponentti, joka on käyttöliittymän autentikaation kulmakivi. Molemmat käyttöliittymät renderöidään tämän komponentin sisälle sen jälkeen, kun autentikaatio on suoritettu eli **AuthChecker**-komponentti tarkistaa kirjautumisen onnistumisen, jonka jälkeen käyttäjän rooliin perustuen renderöi käyttöliittymän. Komponentti luo ja käyttää *authenticated*-tilaa varmentamaan käyttäjän autentikoinnin ja mikäli kyseisen tilan arvo muuttuu sovelluksen käytön aikana niin komponentti jälleen ohjaa käyttäjän kirjautumisnäkymään.

```
if (loading) {
    return <Loading />
}
if (!authenticated) {
    return <Navigate to="/login" />
}
if (role === "admin" || role === "Esihenkilö") {
    return <Dashboard />
}
return <Homepage />;
```

Utils-kansio

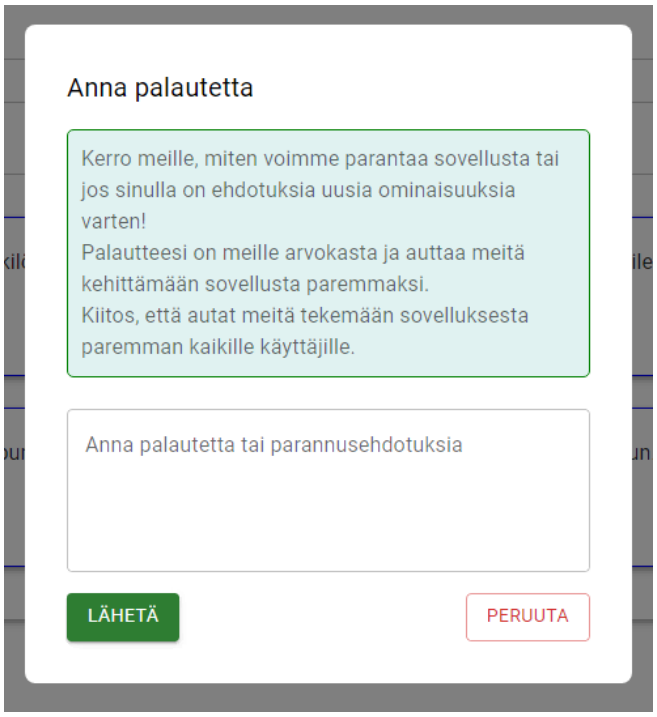
Utils/-kansio pitää sisällään erilaisia ohjelmia ja komponentteja, joita ei tarvitse renderöidä jatkuvasti, mutta kuitenkin säännöllisesti. Tällaisia komponentteja ovat esimerkiksi käyttäjälle esitettävät tarkennukset ennen tietojen lähettämistä.

DialogBox.tsx-komponentti on hyvin laajalti muokattavissa oleva sovelluksen yhtenäinen ponnahdusikkuna, jonka kautta voidaan ohjata esimerkiksi testausajan vahvistaminen ennen sen varausta tai varausta poistettaessa, voidaan vaatia antamaan syy poistamiselle. Erilaiset toiminnallisuudet annetaan komponentille sitä kutsuttaessa. Kuviossa 34 on esitelty DialogBox-komponentin yksi variaatio.



Kuvio 34. DialogBox-komponentti

ErrorReportModal.tsx sekä **FeedbackModal.tsx**-komponentit renderöi nimensä mukaisesti virheilmoitus- sekä palautemodaalit. Toiminnallisuutena komponenteilla on lähettää virheilmoitukset tai palautteet määriteltyyn sähköpostiin. Kuviossa 35 on esitelty palautteeseen käytetty modaali.



Kuvio 35. Palaute-modaali

isWithinRange.tsx-komponentin tarkoitus on tarkistaa ja palauttaa suoritetun testin voimassaolopäivä testin suorittamisen jälkeen. Komponentti tarkistaa myös testiin sidotun käyttäjän tavoitetason ja sen mukaisesti merkkaa testin voimassaolopäivän joko vuoden tai kahden vuoden päähän.

statusConverterFunctions.tsx-komponentin sisältämiä funktioita käytetään muuntamaan numeerisia arvoja kirjallisiksi sekä tarjoamaan eri taustan värejä eri testistatuksille.

6 Pohdinta

Tutkimustyön tavoitteena oli kehittää eri päätelaitteille skaalautuva verkkosovellus, joka keskittää ja helpottaa vuosittaisen lakisääteisen FireFit-testauksen suorittamista toimeksiantajan alueella. Tutkimuskysymyksiä johdettiin vaatimusmäärittelystä kolme kappaletta ja kysymysten kautta sovelluskehityksen keskeisiksi ohjaaviksi teemoiksi nousi saavutettavuus, käytettävyys ja turvallisuus.

Arkkitehtuurisesti oli alusta asti selvää, että kehityksessä tullaan pysymään JavaScript-ohjelmointikielessä, koska missään vaiheessa prosessia ei tullut esiin tarvetta tuoda sovellukseen koneoppimisen malleja tai muuta tekoälyyn viittaavaa toiminnallisuutta, joka olisi voinut kallistaa ennen kaikkea palvelinkielen valintaa Pythonin suuntaan. Toisaalta palvelin suunniteltiin modulaarisesti niin, että uuden rajapinnan lisääminen esimerkiksi koneoppimisen mallia tarjoavaan Python palvelimeen on yksinkertaista ja toteuttamiskelpoista.

Saavutettavuuden osalta merkittävimpinä tuloksina voidaan pitää sovelluksen julkaisua pelastuslaitoksen domainin alle osoitteessa https://firefit.kspela.fi. Julkaisualusta mahdollistaa sovelluksen saavutettavuuden sekä virkahenkilöstön, että sopimushenkilöstön osalta kaikilta laitteilta ja alustoilta. Sovelluksen palvelin rakennettiin myös tukemaan rekisteröityneiden käyttäjien käyttökokemusta ja saavutettavuutta ylläpitämällä kirjautumissessioita niin, että käyttäjien ei tarvitse jokaisella käyttökerralla kirjautua sovellukseen uudelleen.

Käytettävyyden osalta merkittävimpiä tuloksia ovat sovelluksen skaalautuvuus eri päätelaitteille, joka mahdollistaa vaatimusmäärittelyssä mainittujen ominaisuuksien tuomisen eri kohderyhmien saataville sekä ominaisuuksien automatisoinnin niin, että esimerkiksi ajanvarauksia voidaan tehdä ainoastaan ennalta määritetyille ajankohdille ja paloasemille. Pääkäyttäjä- ja esihenkilörooleille

hallintapaneelin osalta myös Suunnittelu-välilehti ja sen työkalut olivat hyvin onnistuneita kokonaisuuksia ja näiden toiminnallisuuksien siirtäminen käyttäjänäkymän kalenteriin oli onnistunut kokonaisuudessaan.

Turvallisuuden osalta sovellus rakennettiin vastaamaan tutkimuksen teoriaosuudessa selvitettyä kerrostettua suojausta ja kaikki tarpeelliset toimet pyrittiin sovellukseen implementoimaan. Liitteessä 2 on esitelty sovelluksen kerrostettu suojaus verrattuna teoriaosuuden Inton (2011) esittämään kerrostettuun suojaukseen.

Kokonaisuudessaan tutkimuksen kehittämisvaihe oli haastava ja aikaa vievä prosessi, jonka haastavuutta lisäsi vielä se, että sovelluksen laajuus kasvoi kehitystyön aikana lähes päivittäin ja erilaisia ominaisuuksia tuli alun karkeaan vaatimusmäärittelyyn jatkuvasti lisää. Toisaalta uusien ominaisuuksien lisääminen oli helppoa sovelluksen ollessa kehitysvaiheessa, mutta myös sen vuoksi, että allekirjoittanut on tehnyt kaiken itse. Laajuudeltaan tällaisen työn läpivienti on valtava työ, jonka aikana nopeasti ymmärtää miksi ohjelmistokehitystä tehdään tiimeissä.

Tässä vaiheessa on myönnettävä, että kehitystyöprosessi alkoi intuitiosta ja tarpeesta kehittää jonkinlainen sovellus havaittuun työelämän ongelmaan, eikä varsinaista vaatimusmäärittelyä alkuun ollut laisinkaan. Jälkeenpäin ajatellen on ensiarvoisen tärkeää, että ennalta tehdään jonkinlaista dokumentaatiota sidosryhmät huomioiden, koska varsinainen ohjelmointityö helpottuisi huomattavasti, mikäli edes osa ominaisuuksista on ennalta määritelty. Tähän verraten oli suuri ilo huomata jo implementoidun kerrostetun suojauksen täsmäävän lähes yksi yhteen Inton (2011) Pro Gradu-tutkielmassa esitetyn kerrostetun suojauksen kanssa.

Kehitetyn sovelluksen viimeisin versio valmistui syyskuun 2024 alkupuolella, jonka jälkeen piti käynnistyä testausvaihe, johon oli valmiiksi kerätty testausprosessissa mukana olevia virkahenkilöitä, että sopimushenkilöstöä, mutta testausvaihe ei kuitenkaan ole vielä tämän kappaleen kirjoitushetkellä alkanut. Harmillisesti siis yksi kehitystyön oleellisista osista on vielä käymättä läpi.

Kehitetty sovellus on laajuudeltaan ja ominaisuuksiltaan sellainen, että väistämättäkin tulevaisuudessa sen ylläpito, huolto ja jatkokehitys tulee vaatimaan toimeksiantajalta resursseja ja ammattitaitoa. Tästä syystä sovelluksesta luodaan tämän opinnäytetyön ulkopuolella dokumentaatiota sen

ylläpidosta ja jatkokehityksestä. Jos taas sovelluksen käyttö on sujuvaa ja se helpottaa nimenomaan prosessia, jonka vuoksi se on kehitetty, niin näen, että sovellusta voisi tarjota myös muille pelastuslaitoksille ja alueille. Arkkitehtuuristen valintojen sekä tietovarastoinnin keinojen ansiosta sovelluksen integraatio muille pelastuslaitoksille on yksinkertaista ja toteuttamiskelpoista. Sovelluksen käytöstä tulisikin siis kerätä käyttäjäpalautetta säännöllisin välein käyttöönoton jälkeen.

Lähteet

2020 Developer Survey. 2020. StackOverflow käyttäjäkysely. Viitattu 21.08.2024. https://survey.stackoverflow.co/2020#overview.

About the OWASP Foundation. N.d. OWASP-verkkosivusto. Viitattu 4.9.2024. https://owasp.org/about/.

Ahuja, A. 2024. The Supreme Guide To Understand The Workings Of CPython. Tutoriaali verkkosivustolla. Viitattu 29.8.2024. https://www.simplilearn.com/tutorials/python-tutorial/understand-the-workings-of-cpython.

Arrays, B. 2023. Securing Your Data: Encryption and Privacy in Modern Web Applications. Artikkeli verkkosivustolla. Viitattu 3.9.2024. https://medium.com/@betaarrays/securing-your-data-encryption-and-privacy-in-modern-web-applications-e54f503efae9.

Askarov, O. 2023. All You Need to Know About NPM and Node Packages as a Beginner. Artikkeli verkkosivustolla. Viitattu 29.8.2024. https://medium.com/@oroz.askarov/all-you-need-to-know-about-npm-and-packages-as-a-beginner-b6fcea8b3519.

Austin, D. 2023. A Brief History of TypeScript: From Origin to Moden Adoption. Artikkeli. Viitattu 21.8.2024. https://medium.com/totally-typescript/a-brief-history-of-typescript-from-origin-to-modern-adoption-791368ec4b91.

Boruń, Ł. 2023. Why use Node JS for your enterprise project?. Artikkeli verkkosivustolla. Viitattu 29.8.2024. https://www.miquido.com/blog/why-use-node-js/.

C Programming Basics Explained. 2024. Artikkeli verkkosivustolla. Viitattu 29.8.2024. https://www.shiksha.com/online-courses/what-is-c-programming-st619-tg1436.

Cass, S. 2023. The Top Programming Languages 2023. IEEE Spectrum. Viitattu 20.8.2024. https://spectrum.ieee.org/the-top-programming-languages-2023.

Chen, A. 2024. TypeScript vs Python: Which One is Best for You?. Blogi PureCode.ai -verkkosivustolla. Viitattu 21.08.2024. https://purecode.ai/blogs/typescript-vs-python.

Deshpande, C. 2024. The Best Guide to Know What is React. Opas Simplilearn-verkkosivustolla. Viitattu 21.08.2024. https://www.simplilearn.com/tutorials/reactjs-tutorial/what-is-reactjs.

Erityisten henkilötietoryhmien käsittely. N.d. Tietosuojavaltuutetun toimisto. Ohje tietojen käsittelystä. Viitattu 3.9.2024. https://tietosuoja.fi/erityisten-henkilotietoryhmien-kasittely.

History of JavaScript. 2024. Artikkeli verkkosivustolla. Viitattu 21.08.2024. https://www.geeksforgeeks.org/history-of-javascript/.

History of Python. 2024. Artikkeli verkkosivustolla. Viitattu 21.08.2024. https://www.geeksforgeeks.org/history-of-python/.

Holdsworth, J., Kosinski, M. 2024. What is information security?. IBM verkkoartikkeli. Viitattu 3.9.2024. https://www.ibm.com/topics/information-security.

Hyvärinen, N. 2018. Miksi tietoturva tyritään jo perusasioissa?. Artikkeli verkkosivustolla. Viitattu 3.9.2024. https://blog.f-secure.com/fi/miksi-tietoturva-tyritaan-jo-perusasioissa/.

Into, T. 2011. Sosiaalisten verkkosovellusten tietoturva. Pro Gradu-tutkielma. Viitattu 3.9.2024. https://jyx.jyu.fi/bitstream/handle/123456789/26923/1/URN%3ANBN%3Afi%3Ajyu-2011050810765.pdf.

Johdatus kryptografiaan. N.d. CS-EJ4404-kurssi. Aalto-yliopiston verkkosivut. Viitattu 3.9.2024. https://plus.cs.aalto.fi/cs-ej4404/2022/?hl=fi.

Kareliya, A. 2024. NodeJS vs Python: The Great Backend Dilemma. Artikkeli verkkosivustolla. Viitattu 29.8.2024. https://radixweb.com/blog/nodejs-vs-python.

Keski-Suomen pelastuslaitoksen palvelutasopäätös vuosille 2021–2024. 2021. Keski-Suomen pelastuslaitoksen www sivut. Viitattu 20.8.2024. https://pelastustoimi.fi/documents/25266713/133174990/keskisuomen_pelastuslaitoksen-palvelutasopaatos_2021_2024.pdf/1a0dabb6-5ad7-ab2b-3f8b-03096794ed46/keskisuomen_pelastuslaitoksen-palvelutasopaatos_2021_2024.pdf?t=1665482080691.

Kinsman, C. 2022. Introduction to Node Modules. Artikkeli verkkosivustolla. Viitattu 29.8.2024. https://www.codemag.com/article/1709061/Introduction-to-Node-Modules.

Knupfer, F. 2022. Complete guide to application logging. Artikkeli verkkosivustolla. Päivitetty 2024. Viitattu 4.9.2024. https://newrelic.com/blog/best-practices/how-to-manage-an-application-log.

Kuntien avainluvut. 2023. Keski-Suomen väkiluku 2023. Julk. 2.10.2023. Helsinki: Tilastokeskus. Suomen virallinen tilasto. Viitattu 22.10.2024. Kuntien avainluvut muuttujina Alue ja Tiedot. PxWeb (stat.fi)

L 379/2011. Pelastuslaki. Viitattu 20.8.2024. https://www.finlex.fi/fi/laki/ajantasa/2011/20110379.

L 523/1999. Henkilötietolaki. Viitattu 3.9.2024. https://www.finlex.fi/fi/laki/alkup/1999/19990523#Lidm46111191447664.

L 5.12.2018/1050. Tietosuojalaki. Viitattu 5.9.2024. https://www.finlex.fi/fi/laki/ajantasa/2018/20181050.

L 11.6.1999/731. Suomen perustuslaki. Viitattu 5.9.2024. https://www.finlex.fi/fi/laki/ajantasa/1999/19990731.

Libraries in Python. 2024. Artikkeli verkkosivustolla. Viitattu 29.8.2024. https://www.geeksforgeeks.org/libraries-in-python/.

Liimatainen, L., Hautamäki, J., Kirjalainen, E., Kokko, M., Korhonen, K., Laitinen-Väänänen, S., Norvapalo, K., Törn-Laapio, A. & Hyvätti, S. 2024. Eettiset periaatteet. Jyväskylän ammattikorkeakoulu. Viitattu 20.9.2024. https://www.jamk.fi/fi/media/41106.

Lindström, S. 2020. Suosituimmat ohjelmointikielet 2020. Itewiki-julkaisualusta. Viitattu 20.8.2024. https://www.itewiki.fi/blog/2020/08/suosituimmat-ohjelmointikielet-vuonna-2020/.

Long, M. 2023. Why JavaScript is Single Threaded?. Artikkeli verkkosivustolla. Viitattu 21.08.2024. https://groovetechnology.com/blog/why-javascript-is-single-threaded/.

Lusa, S., Halonen, J., Punakallio, A., Wikström M., Lindholm, H. & Luukkonen, R. 2015. Pelastajien fyysisen toimintakyvyn arviointijärjestelmän käytettävyys ja FireFit-indeksin kehittäminen. Julkari. Viitattu 20.8.2024. https://www.julkari.fi/bitstream/handle/10024/129628/FireFit-j%C3%A4rjestelm%C3%A4n%20k%C3%A4ytett%C3%A4vyys%20ja%20FireFit-indeksi.pdf?sequence=1.

Maricheva, A. 2023. A Trip Back in Time: the History of JavaScript. Artikkeli Softteco-verkkosivustolla. Viitattu 21.08.2024. https://softteco.com/blog/history-of-javascript.

Meistä. N.d. Keski-Suomen pelastuslaitoksen www sivut. Viitattu 20.8.2024. https://pelastustoimi.fi/keski-suomi/meista.

Mireles, Y. 2022. See everything in your tech stack with these logging best practices. Artikkeli verkkosivustolla. Päivitetty 2024. Viitattu 4.9.2024. https://newrelic.com/blog/best-practices/best-log-management-practices.

Monthan, M. 2024. Keskustelu palopäällikön kanssa 11.9.2024. Keski-Suomen pelastuslaitoksen keskuspaloasema.

Morris, S. N.d. Tech 101: What Is JavaScript?. Blogikirjoitus verkkosivustolla. Viitattu 21.08.2024. https://skillcrush.com/blog/javascript/.

Muurinen, M. 2019. Tietosuoja vai tietoturva?. Visma-blog. Viitattu 3.9.2024. https://www.visma.fi/blog/tietosuoja-tietoturva/.

Node.js Modules. 2024. Opas verkkosivustolla. Viitattu 29.8.2024. https://www.geeksforgeeks.org/node-js-modules/.

OWASP Top Ten. 2021. OWASP. Verkkojulkaisu. Viitattu 4.9.2024. https://owasp.org/www-project-top-ten/.

Pandya, J. 2024. Top 10 Reasons to Choose Python for Artificial Intelligence Project Backend Development. ExpertAppDevs -blogi. Viitattu 21.08.2024. https://www.expertappdevs.com/blog/choose-python-for-ai-project.

Pernaa, J. 2013. Kehittämistutkimus tutkimusmenetelmänä. Jyväskylä: PS-kustannus. Viitattu 20.9.2024. https://helda.helsinki.fi/server/api/core/bitstreams/fd4fcd23-2d7c-474a-a426-438c93075ff3/content.

Python History and Versions. N.d. Python opas Javatpoint -verkkosivustolla. Viitattu 21.08.2024. https://www.javatpoint.com/python-history.

Rao, H. 2023. Leveraging the Power of Node.js and Python for Full-Stack Backend Development. Artikkeli verkkosivustolla. Viitattu 29.8.2024. https://medium.com/@hammadrao891/leveraging-the-power-of-node-js-and-python-for-full-stack-backend-development-f1ce7c01b8ec.

Robie, J. N.d. What is the Document Object Model?. Artikkeli verkkosivustolla. Viitattu 21.8.2024. https://www.w3.org/TR/WD-DOM/introduction.html.

Session Management Cheat Sheet. N.d. OWASP. Cheat Sheet Series. Viitattu 4.9.2024. https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html.

Sharma, A. 2023. Understanding Request And Response Model: A General Overview. Artikkeli verkkosivustolla. Viitattu 29.8.2024. https://medium.com/@imakashsharma135/understanding-request-and-response-model-a-general-overview-831c24d2288.

Shaw, A. N.d. Your Guide to the CPython Source Code. Artikkeli verkkosivustolla. Viitattu 29.8.2024. https://realpython.com/cpython-source-code-guide/.

Spain, J. 2022. What is Django?. Videotallenne Youtube-palvelussa. Viitattu 21.08.2024. https://www.youtube.com/watch?v=t_p4ZyAYyaY.

SQL Injection Prevention Cheat Sheet. N.d. OWASP. Cheat Sheet Series. Viitattu 4.9.2024. https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html.

Sufiyan, T. 2024a. ReactJS State: SetState, Props and State Explained. Tutoriaali verkkosivustolla. Viitattu 29.8.2024. https://www.simplilearn.com/tutorials/reactjs-tutorial/reactjs-state.

Sufiyan, T. 2024b. What is Node.js? A Complete Guide for Developers. Tutoriaali Simplilearn-verkkosivustolla. Viitattu 28.8.2024. https://www.simplilearn.com/tutorials/nodejs-tutorial/what-is-nodejs.

Talekar, S. 2023. 6 Reasons To Choose Python For Backend Development. Thinkitive verkkosivusto. Viitattu 21.08.2024. https://www.thinkitive.com/blog/6-reasons-to-choose-python-for-backend-development/.

Thulo, C. 2023. React Components, Instances, and Elements. Artikkeli verkkosivustolla. Viitattu 29.8.2024. https://medium.com/@codethulo/react-components-instances-and-elements-786a10fc3a18.

Uzoma, O. 2023. Brief History of Nodejs. Artikkeli verkkosivustolla. Viitattu 21.08.2024. https://medium.com/@ogbuuzoma413/brief-history-of-nodejs-de0cac0af448.

What is Flask Python. N.d. Python opas Pythonbasics.org verkkosivustolla. Viitattu 21.08.2024. https://pythonbasics.org/what-is-flask-python/.

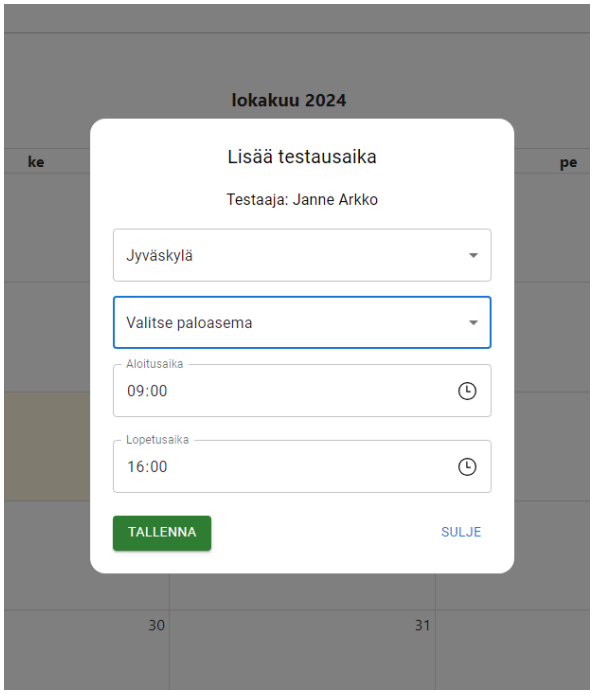
What is the purpose of hashing passwords in web applications?. 2023. EITC/IS/WASF Cybersecurity-sertifikaatti. Viitattu 3.9.2024. https://eitca.org/cybersecurity/eitc-is-wasf-web-applications-security-fundamentals/authentication-eitc-is-wasf-web-applications-security-fundamentals/webauthn/examination-review-webauthn/what-is-the-purpose-of-hashing-passwords-in-web-applications/.

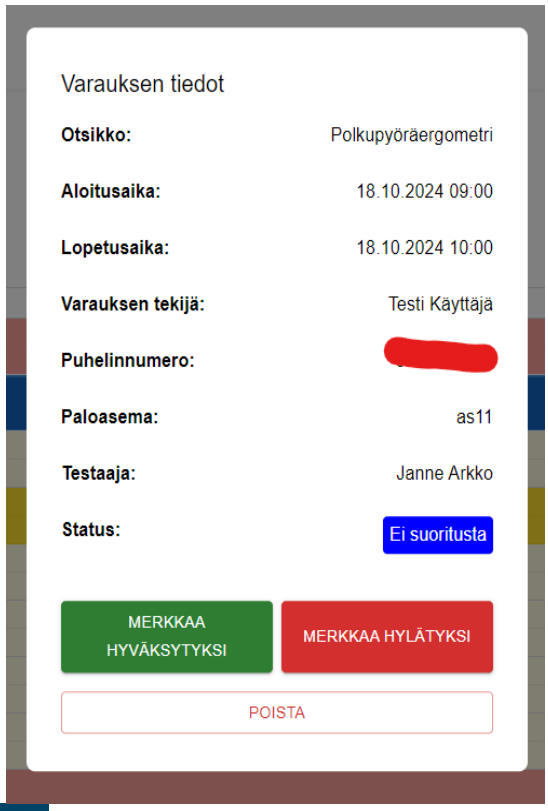
What Is Python Used For? A Beginner's Guide. 2024. Artikkeli Coursera.org -verkkosivustolla. Viitattu 21.08.2024. https://www.coursera.org/articles/what-is-python-used-for-a-beginners-guide-to-using-python.

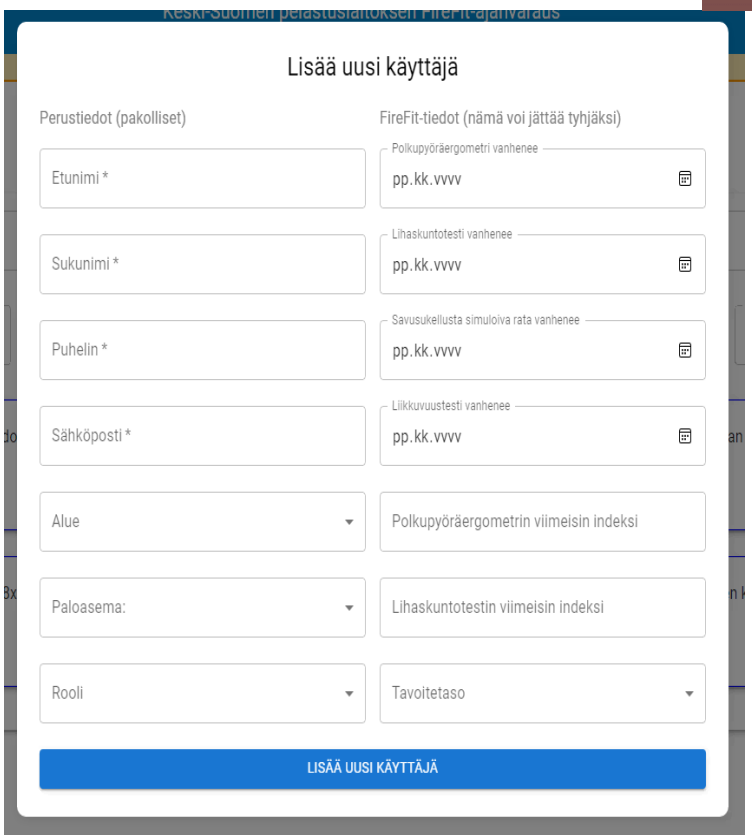
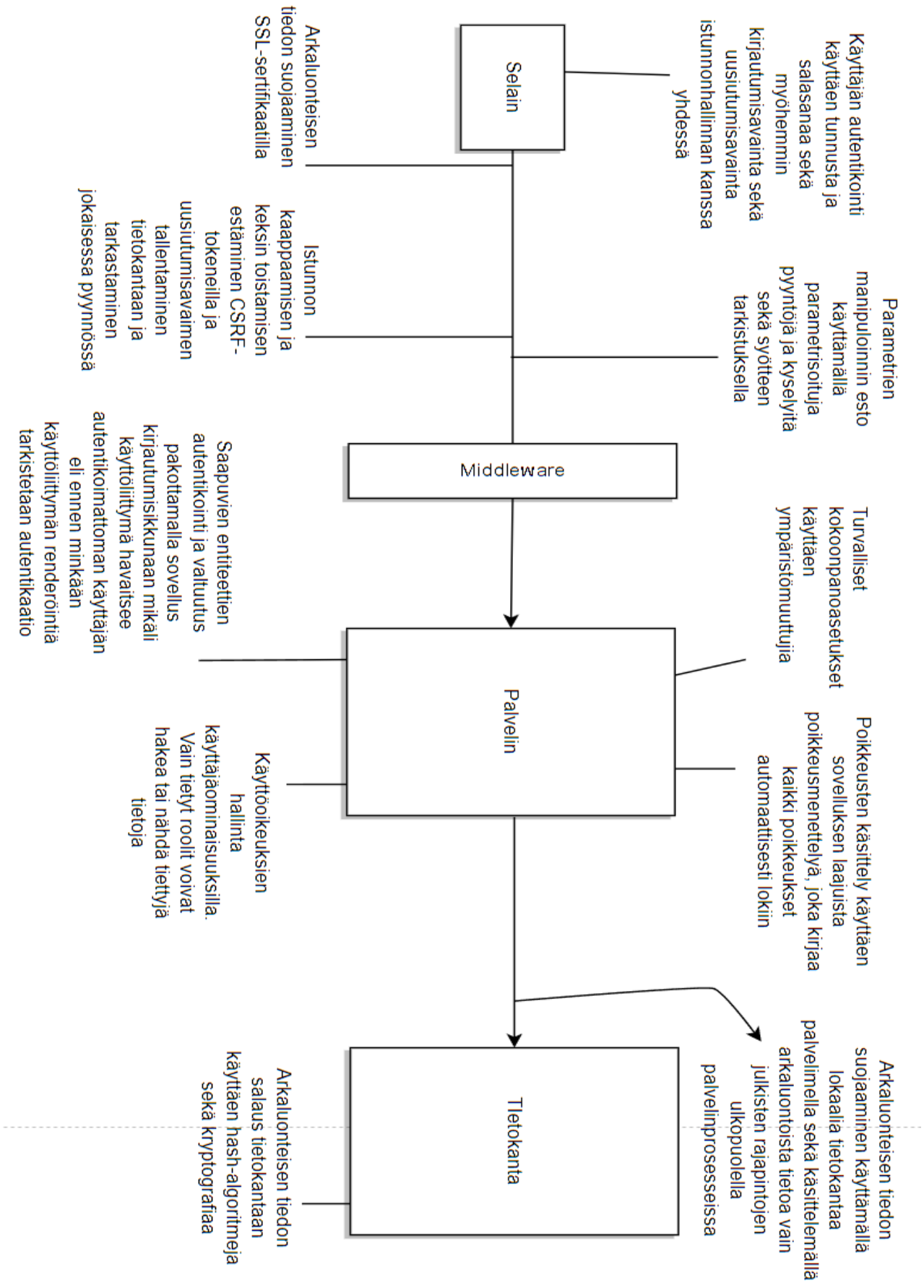
What Is A JavaScript Framework?. 2024. Artikkeli verkkosivustolla. Viitattu 21.08.2024. https://generalassemb.ly/blog/what-is-a-javascript-framework/.

Liitteet

Liite 1. Käyttöliittymän kuvankaappauksia







Liite 2. Implementoitu kerrostettu suojaus