OULUN AMMATTIKORKEAKOULU

Valtteri Göös

PROGRESSIIVISEN WEB-APPLIKAATION KEHITTÄMINEN KUNTOSALIHARJOITTELUN SEURANTAAN

PROGRESSIIVISEN WEB-APPLIKAATION KEHITTÄMINEN KUNTOSALIHARJOITTELUN SEURANTAAN

Valtteri Göös
Opinnäytetyö
Syksy 2024
Tietotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma, ohjelmistokehityksen suuntautumisvaihtoehto

---

Tekijä: Valtteri Göös
Opinnäytetyön nimi: Progressiivisen Web-applikaation kehittäminen kuntosaliharjoittelun seurantaan
Työn ohjaaja: Teemu Korpela
Työn valmistumislukukausi ja -vuosi: Sivumäärä: 31

Tämän opinnäytetyön tavoitteena oli kehittää uusi ominaisuus progressiiviseen web-applikaatioon, jonka avulla käyttäjät voivat seurata kehitystään saliharjoituksissa. Työn tilaajana on HUR Oy, joka valmistaa voimaharjoittelulaitteita erityisesti senioriliikuntaan ja kuntoutukseen. Sovellus perustuu jo olemassa olevaan versioon. Uusi ominaisuus sisälsi neljä pääkomponenttia: historianäkymä, aikajanan valinta, treeniohjelman valinta ja datan visualisointi. Käyttäjät voivat tarkastella treenihistoriaa, valita aikarajauksia ja ohjelmia, sekä analysoida kehitystään yksityiskohtaisesti eri lihasryhmissä. Visualisointia tuetaan kuvaajilla, jotka auttavat tunnistamaan kehityskohteita.

Sovelluksen kehityksessä käytettiin TypeScript-ohjelmointikieltä ja React-kirjastoa. Palvelinpuolen ratkaisuina toimivat AWS-pilvipalvelu ja selaimen ja palvelimen välisestä tiedonsiirrosta vastasi Python Django-Framework rajapinta. Teoriaosuudessa perehdyttiin progressiivisiin applikaatioihin, käytettyihin teknologioihin, yleisesti palvelinpuolen teoriaa, ja lopuksi tarkasteltiin työn toteutusta.

Lopputulokseksi saatiin toimiva kokonaisuus. Uuteen versioon saatiin toteutettua kaikki toivotut ominaisuudet, ja testiversio on ollut asiakaskäytössä ja tähän asti palaute on ollut positiivista. Sovellusta tullaan jatkokehittämään, ja tulevaisuudessa sille suunnitellaan tämän työn jatkona uudisteltu käyttöliittymä.

Asiasanat: JavaScript, React, verkko-ohjelmointi, progressiivinen applikaatio, kuntoiluharjoituksen seuranta

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Information Technology, Option of Software Development

---

Author(s): Valtteri Göös
Title of thesis: Development of a Progressive Web Application for Gym Progress Tracking
Supervisor(s): Teemu Korpela
Term and year when the thesis was submitted:
Number of pages: 31

The aim of this thesis was to add a new feature to a progressive web application that allows users to track their progress in strength training. The client for this project is HUR Oy, a company that manufactures strength training equipment specifically for senior fitness and rehabilitation. The application is based on an existing version. The new feature consists of four main components: the training history view, timeline selection, workout program selection, and data visualization. Users can view their training history, select time frames and programs, and analyze their progress in detail across different muscle groups. The visualization is supported by charts that help identify areas for improvement.

The application development used TypeScript programming language and the React library. For the server-side solution, AWS cloud services were used, and data transfer between the browser and the server was handled by a Python Django Framework API. The theoretical section explored progressive web applications, the technologies used, general server-side theory and the implementation of the project.

The result is a functional application. All the desired features were successfully implemented in the new version, and the test version has been in use by the client with positive feedback so far. The application will continue to be developed, and in the future, it will feature a completely redesigned user interface

---

Keywords: JavaSrcipt, React, web-programming, progressive application, exercise session monitoring

SISÄLLYS

1 JOHDANTO

Tämän opinnäytetyön aiheena on kehittää progressiivista web-applikaatiota, jonka avulla asiakkaat voivat tarkastella omaa kehitystään salilla yleisellä tasolla. Työn tilaajana toimi HUR Oy, joka on suomalainen senioriliikuntaan ja kuntoutukseen erikoistunut voimaharjoittelulaitteiden valmistaja.

Sovellus pohjautuu jo olemassa olevaan versioon, joka on saanut asiakkailta palautetta, kehitykseen. Tavoitteena on lisätä uusi ominaisuus, joka tuo käyttöliittymään monipuolisuutta ja visuaalista houkuttelevuutta sekä parantaa käyttäjäkokemusta. Uuden ominaisuuden odotetaan parantavan asiakkaiden motivaatiota ja sitoutumista liikuntaan tarjoamalla heille selkeän ja helposti ymmärrettävän kuvan heidän kehityksestään.

Sivu koostuu neljästä päävaiheesta: salitreenien historianäkymä, aikajanan valitseminen, treeniohjelman valinta sekä datan visualisoiminen.
Salitreenien historianäkymän avulla käyttäjät voivat tarkastella aiempia salitreenejä ja niihin liittyviä tietoja, kuten käytetty ohjelma, treenin pituus, suoritetut toistot ja käytetyt painot.
Aikajanan valintaominaisuuden avulla käyttäjät voivat rajata näkyviin tulevat tiedot tiettyyn aikaväliin, esimerkiksi viikkoon tai kuukauteen.
Treeniohjelman valinnalla käyttäjät voivat valita tietyn treeniohjelman, jonka kehitystä haluavat seurata. Tämän osan avulla näytetään ohjelmasta yleiskuva, kehitys prosentuaalisesti sekä muita relevantteja tietoja, kuten keskimääräiset toistomäärät ja sarjat, sekä suurin ja pienin käytetty paino. Katsausta syvennetään myös aktiviteettipohjaisella tarkastelulla, jossa käyttäjällä on mahdollisuus valita liike kerrallaan tuloksia. Tämä mahdollistaa yksityiskohtaisemman analyysin kehityksestä eri lihasryhmissä. Uusi versio tarjoaa visuaalisia esityksiä treenivolyymista ja käytetystä ajasta kuvaajien avulla. Nämä kuvaajat auttavat käyttäjiä hahmottamaan ja tunnistamaan mahdollisia kehityskohteita.

Sovelluksen kehitys keskittyy vahvasti selainohjelmointiin, hyödyntäen Typescript-ohjelmointikieltä ja React-kirjastoa. Nämä teknologiat mahdollistavat tehokkaan ja skaalautuvan käyttöliittymän rakentamisen. Palvelinpuolen ratkaisuna toimii AWS-pilvipalvelu ja Python Django REST rajapinta, datan hakemiseen tietokannasta.

2 PROGRESSIIVINEN VERKKOSOVELLUS

Vuonna 2015 Google Chrome kehittäjä Alex Russell esitteli PWA-konseptin, joka mullisti tavan, jolla verkkosovelluksia kehitetään.
Progressiivinen verkkosovellus (PWA) on verkkoteknologioilla rakennettu sovellus, joka tarjoaa natiivisovelluksen kaltaisen käyttäjäkokemuksen. PWA yhdistää verkkosovellusten saavutettavuuden ja monialustaisen käytön natiivisovellusten edistyksellisiin ominaisuuksiin, kuten offline-käyttöön, taustatoimintoihin ja integraatioihin laitteen muiden sovellusten kanssa. Tämä mahdollistaa saman sovelluksen käytön eri alustoilla – kuten puhelimilla, tableteilla ja tietokoneilla – yhdestä koodipohjasta ilman erillisiä asennuksia (1).

Monien vuosien ajan natiivisovelluksilla oli etulyöntiasema, koska ne pystyivät hyödyntämään natiiviin alustaan kiinnitettyjä API-rajapintoja ja hakemaan tarvittavia resursseja, joita verkko ei ollut alun perin suunniteltu tekemään. Lisäksi vielä kymmenen vuotta sitten responsiivinen suunnittelu ei ollut edes olemassa, mutta nykyään CSS-kirjastot tarjoavat mahdollisuuden skaalautuviin asetteluihin ja oikean kokoisiin kuviin kaikenkokoisille näytöille (1).

2.1 PWA.n tärkeimmät ominaisuudet ja erot natiivisovelluksista

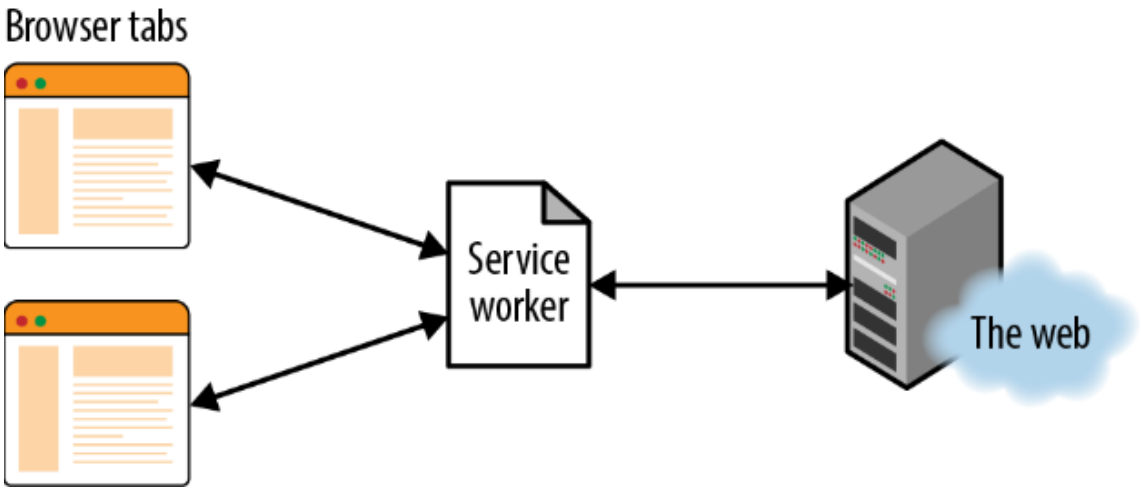
Progressiiviset web-sovellukset (PWA) tarjoavat monia etuja, jotka vastaavat natiivisovelluksille asetettuja odotuksia. Ne eivät ole riippuvaisia käyttäjän verkkoyhteydestä kuten perinteiset verkkosivustot, sillä ne rekisteröivät palvelintyöntekijän, joka havaitsee ja reagoi yhteyden muutoksiin. Tämä mahdollistaa täysipainoisen käyttäjäkokemuksen niin offline-tilassa kuin heikossa tai epävakaassa yhteydessä. Esimerkiksi käyttäjät voivat käyttää PWA:ta lentäessään ja suorittaa toimintoja, kuten viestien lähettämistä, jotka saadaan päätökseen, kun yhteys palautuu (2).

PWA:t tarjoavat myös erittäin nopeat latausajat, sillä service-workerien avulla sivustot voivat latautua lähes välittömästi riippumatta yhteyden nopeudesta. Tämä tekee PWA:sta usein nopeampia kuin perinteiset verkkosivustot ja jopa natiivisovellukset. Lisäksi PWA:t voivat lähettää push-ilmoituksia käyttäjilleen, jotka voivat herättää kiinnostuksen ja sitouttaa käyttäjiä palaamaan sovellukseen pitkänkin tauon jälkeen, samalla tavoin kuin natiivisovelluksista tulevat ilmoitukset (2).

Kun käyttäjä osoittaa kiinnostusta PWA:ta kohtaan, selain ehdottaa sen lisäämistä aloitusnäytölle, jolloin se muistuttaa natiivisovellusta ulkoasultaan. Käyttäjät voivat käynnistää PWA:n aloitusnäytöltä ja nauttia natiivimaisesta kokemuksesta, kuten splash-ruudusta ja koko näytön tilasta ilman selaimen tai puhelimen käyttöliittymiä. Lisäksi PWA:t voivat lukita näyttöorientaation tiettyyn suuntaan, mikä on erityisen tärkeää esimerkiksi peleissä. Tämä kaikki yhdistettynä tekee PWA:sta erittäin houkuttelevan vaihtoehdon natiivisovelluksille (2).

2.2 PWA:n Service Worker

Progressiivisten web-sovellusten ydin on palvelintyöntekijöissä (service workers). Service workerit edustavat merkittävää muutosta siihen, miten tarkastelemme web-kehitystä. Niiden ymmärtäminen on keskeistä, jotta voimme arvioida niiden mahdollisuuksia. Service worker on skripti, joka voidaan rekisteröidä hallitsemaan yhtä tai useampaa sivua verkkosivustolla. Tällaisessa roolissa service worker voi kuunnella ja reagoida kaikkiin hallitsemiensa sivujen tapahtumiin. Esimerkiksi verkkopyynnöt voivat jäädä service workerin käsiteltäväksi. Service worker siis toimii eräänlaisena kerroksena (kuva 1) joka pystyy vastaamaan pyyntöihin riippumatta siitä on verkko kytkettynä tai ei (2).



Kuva 1. Service workerin toiminta sivujen kerroksena (2)

Tämä tarkoittaa, että vaikka käyttäjä sulkisi kaikki sovelluksen välilehdet, service worker voi silti kommunikoida palvelimen kanssa. Se voi vastaanottaa ja näyttää push-ilmoituksia tai varmistaa että käyttäjän suorittamat toimenpiteet tulevat toimitetuksi palvelimelle (2).

3 KÄYTETTYJEN TEKNOLOGIOIDEN TEORIAA

3.1 Typescriptin historia ja erot Javascriptiin

Typescript ohjelmointikieli on kehitelty ja paranneltu seuraaja Javascript–ohjelmointikielestä, joka on saavuttamassa suurta suosiota nykyajan selainohjelmoinnissa. Typescriptin kehitti Anders Hejlberg, vuonna 2010 ja julkaistiin myöhemmin 2012. Typescript on syntaksiltaan sama kuin Javascript, muutamia lisäominaisuuksia lukuun ottamatta. Typescriptin tärkein ominaisuus on staattinen tyypitys, mikä tarkoittaa, että muuttujien ja funktioiden tyypit määritellään koodin kirjoitusvaiheessa, eikä tyyppejä voi muuttaa lennossa (kuten Javascriptissä) (3).

3.2 React.js kirjaston syntyperä ja toimintaperiaatteet

React kirjaston kehittäjänä oli Facebookin ohjelmistokehittäjä Jordan Walke. Vuoden 2013 jälkeen React julkaistiin avoimeksi lähekoodiksi ja täten liittyi muiden käyttöliittymäkirjastojen joukkoon. React laajeni nopeasti, ja esimerkiksi jo 2015 Netflix ilmoitti käyttävänsä Reactia käyttöliittymänsä kehityksessä. Nopeaa nousua tuki myös vahva kehitystiimi, jotka jo useamman vuoden ajan olivat kehittäneet ja optimoineet projektia. Samana vuonna 2015 julkaistiin React Native mobiilisovellusten kehittämiseen ja muita Reactin kanssa yhteensopivia työkaluja kuten React Router ja Redux julkaistiin samaan aikaan (4, luku 1).

Reactin käyttö selainohjelmoinnissa edellyttää kahden kirjaston, Reactin ja ReactDOM:n, sisällyttämistä. React on kirjasto, joka vastaa näkymien luomisesta. ReactDOM puolestaan on kirjasto, joka huolehtii käyttöliittymän varsinaisesta renderöinnistä selaimessa. React on kirjasto, joka automatisoi selain-DOM:n päivittämisen. Tämä helpottaa suorituskykyisten yksisivusovellusten (SPA) rakentamista, sillä React hoitaa monimutkaiset päivitystoimenpiteet puolestamme.

Reactia käytettäessä meidän ei tarvitse suoraan käsitellä DOM-rajapintaa (API). Sen sijaan annamme Reactille ohjeet, mitä haluamme rakentaa, ja React vastaa elementtien renderöinnistä ja niiden sovittamisesta (4, luku 4).

Selain-DOM koostuu DOM-elementeistä, kun taas React-DOM rakentuu React-elementeistä. Vaikka nämä elementit näyttävät samankaltaisilta, ne ovat periaatteessa erilaisia. React-elementti kuvailee, miltä varsinainen DOM-elementti pitäisi näyttää. Käytännössä React-elementit toimivat ohjeina siitä, miten selain-DOM rakennetaan (4, luku 4).

3.2.1 Komponentit ja JSX

JSX yhdistää JavaScriptin (JS) ja XML:n (X). Se on Javascript-laajennus, jonka avulla React-elementtejä voidaan määritellä suoraan JavaScript-koodissa käyttäen tunnistepohjaista syntaksia (kuva 2). JSX muistuttaa ulkoisesti HTML:ää, mikä voi aiheuttaa sekaannusta, mutta kyseessä on yksinkertaisesti toinen tapa luoda React-elementtejä (4, luku 5).

```
const message = 'Tervetuloa React-sovellukseen!';

return (
  <div>
    <h1>{message}</h1>
    <p>Tämä on esimerkki JSX:stä.</p>
    <button onClick={() => alert('Klikkasit nappia!')}>Klikkaa minua!</button>
  </div>
);
}
```

Kuva 2. JSX-syntaksin kanssa kirjoitettua HTML:ää

JSX-syntaksia käytetään kuten tavallista HTML:n kaltaista rakennetta. Javascriptin mukana voi olla muuttujia tai funktioita, jotka liitetään HTML:n osaksi, tässä tapauksessa {message} näyttää muuttujan arvon ja onClick tapahtumassa on JavaScript-funktio, joka suoritetaan, kun nappia klikataan.

JSX on tiivisti yhteydessä React-komponentteihin, sillä JSX on tapa kirjoittaa React-komponenttien ulkoasu ja rakenne. React-sovellusten käyttöliittymät rakentuvat komponenteista riippumatta niiden koosta, sisällöstä tai käytetyistä teknologioista. Käyttöliittymän osia voivat olla esimerkiksi painikkeet, listat ja otsikot. Yhdessä nämä osat muodostavat käyttöliittymän. (4, luku 4) React-komponentit ovat itsenäisiä, uudelleenkäytettäviä rakennuspalikoita. Ne vastaanottavat syötteitä, joita kutsutaan propseiksi, ja palauttavat React-elementtejä, jotka sitten kuvaavat käyttöliittymää (kuva 3). Komponentteja voidaan käyttää uudelleen sovelluksen eri osissa, mikä auttaa ylläpitämään johdonmukaisuutta ja vähentämään koodin toistamista (5).

```jsx
// Komponentti, joka vastaanottaa propsin ja käyttää sitä renderöinnissä
function Greeting({ name }) {
    return <h1>Hello, {name}!</h1>;
  }

  // Pääkomponentti, joka käyttää Greeting-komponenttia ja siirtää propsin
  function App() {
    return (
      <div>
        <Greeting name="John" />  {/* Siirretään 'name' prop Greeting-komponentille */}
        <Greeting name="Alice" />
      </div>
    );
  }

  export default App;
```

Kuva 3. Tiedonvälitys komponenttien välillä propsien avulla

3.3 Tilanhallinta Reactissa ja ylesimmin käytetyt koukut

Reactin tila on tapa tallentaa ja hallita tietoja tai dataa. Tila on JavaScript-objekti, joka sisältää reaaliaikaisia tietoja tai informaatiota verkkosivulla. Reactin tila on React-komponentin instanssi, joka voidaan määritellä havaittavien ominaisuuksien joukkona, jotka hallitsevat komponentin käyttäytymistä. Toisin sanoen, komponentin tila on objekti, joka sisältää tietoa, joka voi muuttua komponentin elinkaaren aikana. (6) Tilan päivittäminen on erityisen tärkeää Reactissa, joka on suunniteltu tehokkaasti reagoimaan käyttöliittymissä tapahtuviin muutoksiin. Kun sovelluksen tila muuttuu, React päivittää DOM:in reaktiivisen ohjelmointimallin avulla. Tilan päivittämisen kyky on olennaista modernien ja responsiivisten verkkosovellusten luomisessa (7).

Reactissa tilan lisääminen komponentteihin tapahtuu käyttämällä koukku (hooks) ominaisuutta. Koukut sisältää uudelleenkäytettävää logiikkaa, joka on erillään komponenttipuusta. Ne mahdollistavat toiminnallisuuden liittämisen komponentteihin. React tarjoaa useita valmiita koukkuominaisuuksia, joita voidaan käyttää suoraan ilman erillistä määrittelyä. (4, luku 6.)

3.3.1 useState

Reactissa useState on erityinen funktio, joka mahdollistaa tilan lisäämisen komponentteihin. Se tarjoaa tavan määritellä ja hallita tilamuuttujia suoraan komponenteissa (8). UseState voi tallentaa minkä tahansa tyyppisiä arvoja. (kuva 4).

```
// Tilamuuttuja "count" ja sen päivittäjä "setCount"
const [count, setCount] = useState(0);

return (
  <div>
    <h1>Napsautusten laskuri</h1>
    <p>Olet napsauttanut {count} kertaa</p>
    <button
      onClick={() => setCount(count + 1)}
    >
      Napsauta minua
    </button>
    <br />
    <button
      onClick={() => setCount(0)}
    >
      Nollaa laskuri
    </button>
  </div>
);
```

Kuva 4. Tilamuuttujan määrittely komponentille

Kuvan 4 esimerkissä käytetään useState-funktiota, laskurin alkuarvo asetetaan nollaksi. Tilamuuttuja count tallentaa laskurin nykyisen arvon. Tilapäivitykset tehdään setCount-funktion avulla, joka mahdollistaa uuden arvon asettamisen laskurin tilamuuttujaan. Laskurin päivittämiseen on käytössä kaksi nappia. “Napsauta minua” -painikkeen painaminen kutsuu setCount funktiota, joka lisää laskurin arvoon yhden. “Nollaa laskuri” -painike palauttaa laskurin alkuarvoon kutsumalla setCount(0) (9).

3.3.2 useEffect

Kun komponentti renderöi, se yksinkertaisesti palauttaa käyttöliittymän. Renderöinti tapahtuu, kun sovellus latautuu ensimmäisen kerran tai kun propsien tai tilan arvot muuttuvat.
UseEffect-koukkua mahdollistaa sivuvaikutusten, kuten datan hakemisen, DOM-päivitysten ja ajastimien toteuttamiseen komponentin sisällä. Sivuvaikutuksilla tarkoitetaan toimintoja, jotka eivät ole osa renderöintiprosessia, kuten konsoliloki, tai vuorovaikutukset selaimen tai API:en kanssa (kuva 5). Näitä voidaan totetuttaa React-sovelluksessa renderöinnin jälkeen useEffect-koukun avulla (4, luku 7). Oletuksena useEffectia suoritetaan jokaisella renderöinnillä, mutta

riippuvuuslistan avulla voidaan hallita, milloin sivuvaikutus suoritetaan. Tämä tekee useEffectista olennaisen työkalun sivuvaikutusten hallinnassa (10).

```
const [data, setData] = useState(null);

// Haetaan data API:sta
useEffect(() => {
  // Async funktio API-kutsua varten
  const fetchData = async () => {

    const response = await fetch("https://jsonplaceholder.typicode.com/posts");
    const result = await response.json();
    setData(result); // Tallennetaan haettu data tilaan
  };

  fetchData(); // Kutsutaan async funktio
}, []); // Tyhjä riippuvuuslista, joten tämä suoritetaan vain kerran
```

Kuva 5. UseEffectin käyttö datan haussa

Tässä esimerkissä useEffect-koukkua käytetään datan hakemiseen API:sta komponentin latauksen yhteydessä. Koukun avulla voidaan hallita tilan päivityksiä ja varmistaa, että komponentti reagoi dynaamisesti saadun datan perusteella.

3.4 Tietokanta

Tietokannan konsepti juontaa juurensa 1960-luvulle, jolloin kehitettiin ensimmäisiä tapoja tallentaa ja järjestää suuria tietomääriä. IBM loi yhden ensimmäisistä tietokannoista Yhdysvaltain väestölaskentaa varten. 1970-luvulla esiteltiin relaatiotietokantamalli, joka muodosti perustan nykyisille tietokannan hallintajärjestelmille.
Tietokanta on organisoitu kokoelma dataa, joka on tallennettu ja käytettävissä sähköisesti. Niitä käytetään suurten määrien rakenteellista ja rakenteetonta dataa hallintaan, ja ne tukevat monenlaisia toimintoja, kuten datan tallentamista, analysointia ja hallintaa. Tietokantoja hyödynnetään laajasti eri ympäristöissä, kuten liiketoiminnassa, tieteessä ja hallinnoissa (11).

Kaikilla tietokannoilla on yksi perusajatus: Maailmassa syntyy tilanteita, jolloin asioista täytyy tallentaa tietoa, ja nämä asiat ovat yhteydessä toisiinsa monin eri tavoin. Jotta datan tallennuspaikka voitaisiin määritellä tietokannaksi, sen tulee sisältää sekä tiedot että tiedot näiden tietojen välisistä suhteista. Esimerkiksi asiakkaat voidaan yhdistää heidän tekemiinsä tilauksiin, ja varastotuotteen voidaan yhdistää kyseisiä tuotteita koskeviin tilauksiin. Tietokannan idea on, että

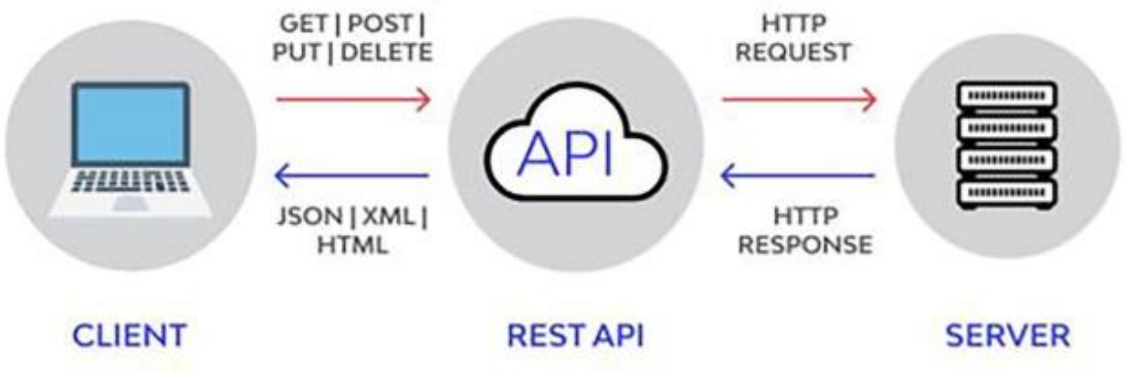
käyttäjän ei tarvitse huolehtia siitä, miten tiedot ovat fyysisesti tallennettuina levylle. Käyttäjä esittää tietojen käsittelypyynnöt tietojen välisiin suhteisiin perustuen, ja tietokannan hallintajärjestelmä (DBMS) kääntää nämä pyynnöt fyysisen tallennuksen mukaisiksi (12).

Tietokannat ovat välttämättömiä tehokkaaseen tiedonhallintaan ja päätöksenteon tukemiseen dataohjautuvassa maailmassa. Niiden avulla voidaan luotettavasti tallentaa, hakea ja käsitellä tietoa, mikä hyödyttää niin pieniä kuin suuria organisaatioita. Tietokantojen eri tyyppien ja rakenteiden ymmärtäminen on keskeistä kaikille, jotka työskentelevät datan parissa. Teknologian kehitys kasvattaa jatkuvasti tietokantojen merkitystä ja niiden tarjoamia mahdollisuuksia (13).

3.5 Rest Api

Verkkokehityksen maailmassa rajapinnat (API:t, Application Programming Interface) ovat keskeisessä asemassa mahdollistaen saumattoman viestinnän eri ohjelmistojen ja palveluiden välillä. API:t muodostavat modernien verkkosovellusten selkärangan yhdistämällä monimuotoisia järjestelmiä. Rajapinnat ovat mullistaneet tapaa, jolla sovelluksia rakennetaan, sillä ne antavat kehittäjille mahdollisuuden hyödyntää laajaa valikoimaa kolmansien osapuolien tarjoamia toimintoja. Tämä ominaisuus mahdollistaa monipuolisten sovellusten kehittämisen ilman, että kaikkea tarvitsee rakentaa alusta asti. Tämä on erityisen tärkeää nykyisessä nopeatahtisessa ja kilpailullisessa kehitysympäristössä (14).

Rest (Representtational State Transfer) on keskeinen arkkitehtuurityyli, joka on monien päivittäisessä käytössä oleva rajapintojen perusta, joita käytetään yleisesti hajautettujen ja skaalautuvien verkkosovellusten rakentamisessa. Restful -rajapinta on verkkorajapinnan tyyppi, joka noudattaa tiettyjä periaatteita resurssien suunnittelussa ja käsittelyssä internetin yli (Kuva 6) (14).



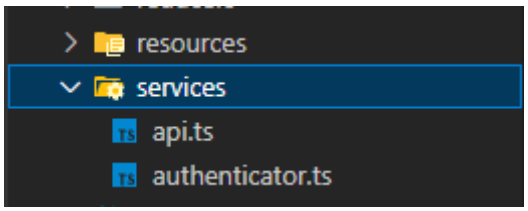
Kuva 6. Restful API (4)

4 TOTEUTUS

Toteutusosuudessa käydään läpi projektin keskeiset vaiheet ja sisällöt. Opinnäytetyö alkoi toimeksiantajan kanssa pidetyllä aloituspalaverilla, jossa määriteltiin työn laajuus ja tavoitteet. Tehtävänäni oli kehittää uusi ominaisuus jo olemassa olevaan sovellukseen. Projektin etenemistä seurattiin säännöllisillä viikkopalavereilla, joissa olivat mukana ohjaava henkilö sekä edellisen version kehittäjä. Näissä sprinttipalavereissa esittelin työni etenemistä, ratkaisin esiin nousseita haasteita ja kerroin seuraavista askelista. Projektia auttamaan sain UI-designerin, joka suunnitteli käyttöliittymän graafisen puolen, ja jonka kanssa yhteistyössä projektia tehtiin.

4.1 Historianäkymä

Treenihistorianäkymä tulisi pitää sisällään listan asiakkaan treeneistä, ja lisäksi tiedot kuten; käytetty ohjelma, päivämäärä sekä treenin kesto. Tietojen haku sekä kommunikointi palvelimen kanssa toteutettiin Axios -kirjastoa käyttäen sekä Django REST API rajapintaa hyödyntäen. Työtä helpotti suuresti se, että tietokantakyselyt olivat valmiina rajapinnassa ja jäljellä oli käyttöliittymältä tulleet pyynnöt. Palvelimelle tehtävät pyynnöt pidetään näille tarkoitetussa Services-projektikansiossa, joihin niihin on helppo viitata eri projektin komponenteissa. (Kuva 7).



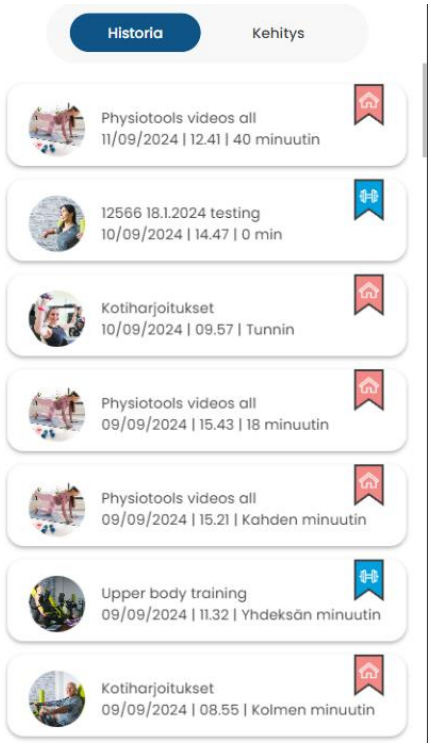
Kuva 7. Services-kansio, jossa pyynnöt API:lle sijaitsevat

Tässä tapauksessa luodaan funktio, joka suorittaa asynkronisen GET pyynnön käyttäen Axios-kirjastoa, jossa palvelimelta haetaan asiakkaan käynnit salilla, jotka Django REST API palauttaa JSON-formaatissa. (Kuva 8).

```
const getVisits = async (userId: number, page = 1, perPage: number) => {
    const serverUrl = localStorage.getItem("server_url");
    const response = await Axios.get(
        `${serverUrl}/visits/?page=${page}&page_size=${perPage}&user__id=${userId}&o=-login`
    );
    return response.data;
};
```

Kuva 8. getVisit funktio asiakkaan käyntien hakuun.

Komponentissa luodaan useEffect koukku, joka käyttää getVisits funktiota hakeakseen palvelimelta vierailut käyttäen parametreja, kuten käyttäjän id jotta tiedetään kenen vierailuja haetaan. Koska vierailuja voi asiakkaalla olla runsaasti täytyy tämä ottaa huomioon listan latauksessa, senpä takia funktio rakennettiin niin että se lataa yhden sivun mitan kerrallaan ja asiakkaan skrollatessa alaspäin lista päivittyy näin lisäten vierailuja. Listan esityksessä käytettiin react-virtualizedin autosizer komponenttikirjastoa, joka tarjoaa erinomaisia joustavia ominaisuuksia React elementeille. Komponentti säätää automaattisesti lapsikomponentin leveyttä ja korkeutta niin että se täyttää kaiken käytettävissä olevan tilan, ja näin eliminoi tarpeen manuaalisesti määrittää komponentin mittoja (15). Kun kyseessä on niin kauan päivittävä lista, kun vierailuja on, tämän ominaisuuden hyödyntäminen osoittautui hyväksi ratkaisuksi. (Kuva 9.)



Kuva 9. Lista asiakkaan treeneistä

4.2 Aikajanan valitseminen

Aikajana tulee määrittämään millä aikavälillä kehitystä tultaisiin näyttämään, ja käyttäjille annettaisiin mahdollisuus sen säätämiseen. Tämä on erittäin keskeinen osa sivun toiminnallisuutta.

Suunnitelman mukaan komponentti sisältää kolme valittavissa olevaa aikaintervallia: 7 päivää, 14 päivää ja kuukausi. Käyttäjä voi valita näistä sopivimman aikavälin. Ratkaisuna on luoda funktiot jokaiselle intervallille, jotka vastaavat kyseisten aikavälien painikkeiden renderöinnistä. Nämä funktiot ovat keskeisessä roolissa intervallin valitsemisessa (kuva 10).

Valinnan suorittamiseen luodaan lisäksi erillinen funktio, joka vastaanottaa tiedon siitä, mikä intervalli on valittu. Ongelmaa monimutkaisti se, että muualla projektissa muut komponentit olivat riippuvaisia tiedosta mikä aikaväli on valittu. Tämän takia asetetaan napin klikkaukset tilamuuttujiin ja myöhemmin tieto välitetään komponentille, joka vastaa data esittämisestä. Näin saadaan oikeat aikavälit kohtaamaan muiden tietojen kanssa.

```
const renderDates = () => {
  if(showMonths || !selectedInterval) return null;

    if(selectedInterval === '7') {
        return renderSevenDays();
    } else if(selectedInterval === '14') {
        return renderFourteenDays();
    } else if(selectedInterval === 'monthly') {
        return renderMonths();
    }
}
```

Kuva 10. Tämä funktio tarkastaa mikä aikaväli valitaan ja tieto välitetään komponenttipuuhun.

Tämän lisäksi varmistetaan, että dataa on näytillä aina kun sivulle tullaan, niin valintojen tilamuuttujien oletusarvoksi asetetaan se kuukausi, milloin viimeinen treeni on tehty. Lopputulokseksi saatiin toimiva kokonaisuus, josta asiakas voi valita intervallin ja aikavälin. (Kuva 11.)

Kuva 11. Esimerkki. Asiakas valitsee intervallin kuukausittain ja aikavälin elokuu.

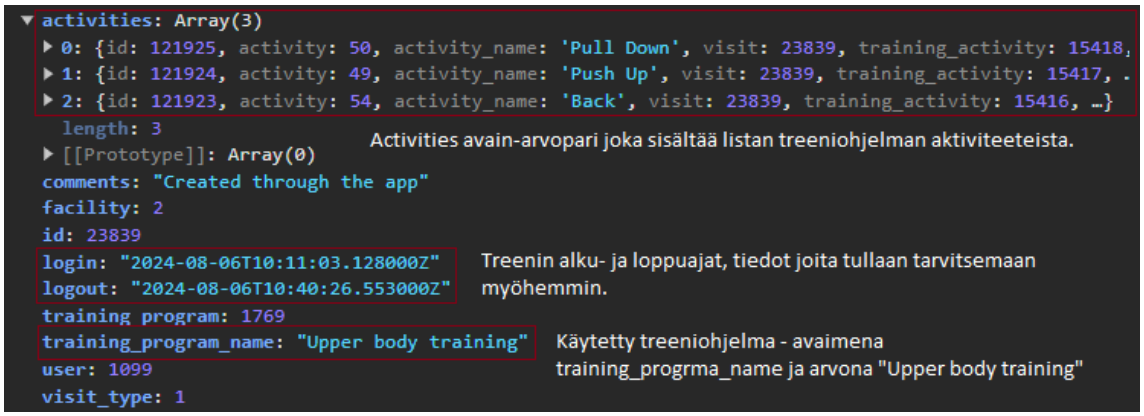
4.3 Treeniohjelman ja aktiviteettien valinta

Seuraavaksi haasteena oli minkä treeniohjelman tietoja halutaan näyttää. Tässäkin käyttäjällä on mahdollisuus itse valita minkä ohjelman tietoja haluaa seurata. Jotta tarvittavat tiedot käyttäjältä saadaan, luodaan useEffect-koukku (kuva 12), jossa suoritetaan API pyyntö tietokannalle käyttäjän treenien hakuun. Tämän avulla saadaan paljon hyödyllistä tietoa kuten treeniohjelmien nimet niiden sisältämät aktiviteetit, treenien päivämäärät, joita voimme jo verrata tätä ennen luotuun aikajanaan.

```
useEffect(() => {
  async function initData() {
      setLoading(true);
      const nextPage = loadedPage + 1;
      try {
          const data = await getVisits(currentUser.id, nextPage, perPage );
          console.log("Fetched Data", data);
          setTrainings(data.results)
      } catch(err) {
          console.error(err);
      }
      setLoading(false);
  }
  initData();
}, [])
```

Kuva 12. UseEffect koukku, jossa API:lta pyydetään käyttäjän treenit ja asetetaan tilamuuttujaan setTrainings

Apilta tuleva tieto saadaan JSON objekteina. JSON on ihmisten luettavissa oleva tiedostomuoto, jota käytetään erityisesti verkkosovelluksissa tiedonvälityksessä, sillä se on kevyt ja helposti käsiteltävissä. Perusrakenne koostuu avain-arvo pareista joita voimme helposti muokata Reactin avulla siihen muotoon minkä parhaaksi näemme (16). Kuvassa 13 esimerkkinä haettu JSON objekti käyttäjän yhdestä treenikerrasta. (Kuva 13).



Kuva 13. Esimerkki haetusta JSON objektista.

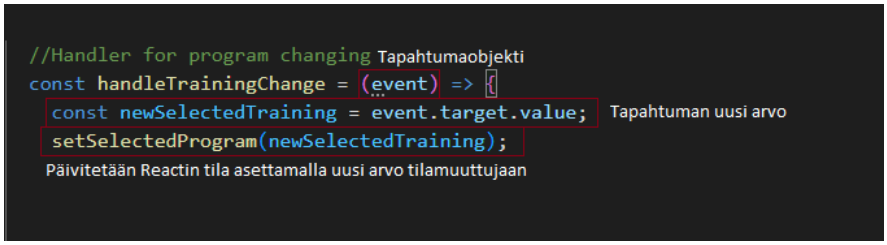
Kuten kuvasta näemme JSON objektin avain-arvopari voi sisältää erilaisia arvoja. Aktiviteetti avain-arvoparin arvona on lista treenin aktiviteeteista, jotka tarjoavat kattavat tiedot myöhemmin, kun tarkastellaan treenin kulkua.

Käyttäjän haetuista treeneistä voimme nyt helposti Javascriptin ja Reactin metodeilla poistaa uniikit treeniohjelmien nimet JSON listasta, ja asettaa ne muuttujaan. Tämän jälkeen muuttuja välitetään komponenttipuuhun, jossa treeniohjelmien nimet renderöidään Select komponenttiin (kuva 14), josta käyttäjä voi valita minkä treeniohjelman tietoja halutaan nähdä.

```
<Select
 className={classes.selectButton}
 value={selectedProgram || ''}
 displayEmpty
 onChange={handleTrainingChange}
 MenuProps={{
    PaperProps: {
        className: classes.menuPaper
    }
 }}
 >
    {uniqueTrainingPrograms && uniqueTrainingPrograms.length > 0 ? (
    uniqueTrainingPrograms
    .filter((programName): programName is string => !!programName)
    .map((programName, index) => (
    <MenuItem
    key={index}
    value={programName}
    className={`${classes.menuItem} ${selectedProgram === programName ?
        classes.selectedMenuItem : ''}`}
    >
        {programName}
    </MenuItem>
    ))
    ) : (
        <MenuItem value="" disabled>
            {t('progress_page.no_programs')}
        </MenuItem>
    )}
</Select>
```

Kuva 14. Select komponentti, jossa treeniohjelmien nimet renderöidään dropdown listaan

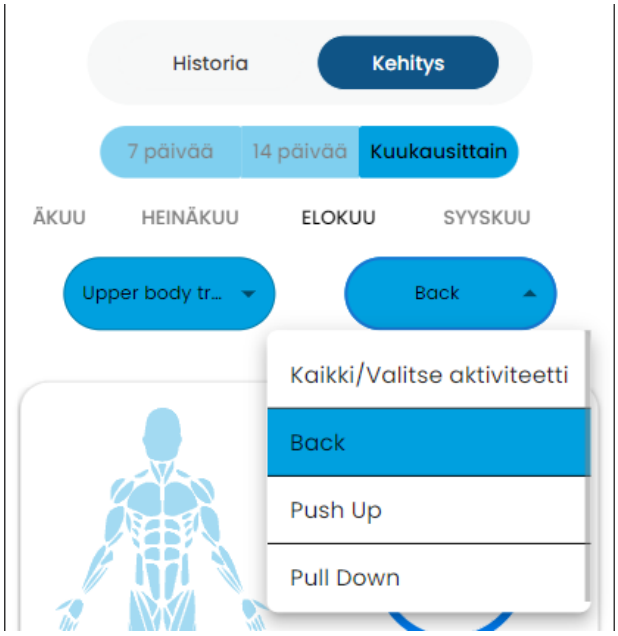
Jotta sujuva treeniohjelmien vaihto listassa olisi mahdollista täytyy luoda vielä funktio handleTrainingChange, jota kutsutaan jokaisessa välissä, kun ohjelmaa yritetään vaihtaa. HandleTrainingChange funktio pitää sisällään tapahtumaobjektin, joka aktivoituu, kun listassa arvo muuttuu. Uusi arvo talletetaan muuttujaan, ja jota käyttämällä päivitetään Reactin uusi tila. (Kuva 15).



Kuva 15. drop-down listan päivitys

Drop-down select komponentti saatiin valmiina Material-UI komponenttikirjastoa hyödyntäen. Projektin yksi keskeisimpiä kirjastoja – Material UI, on avoimen lähdekoodin ilmainen React komponenttikirjastoa, joka tarjoaa useita valmiiksi rakennettuja ja helposti muokattavissa olevia käyttöliittymätyökaluja (17).

Samanlainen käsittely tehtiin myös aktiviteettien valinnalle, ja lopputulokseksi saatiin kaksi drop-down listaa, joista käyttäjät voivat valita mitä treeniohjelmaa haluavat tarkastella. (Kuva 16).

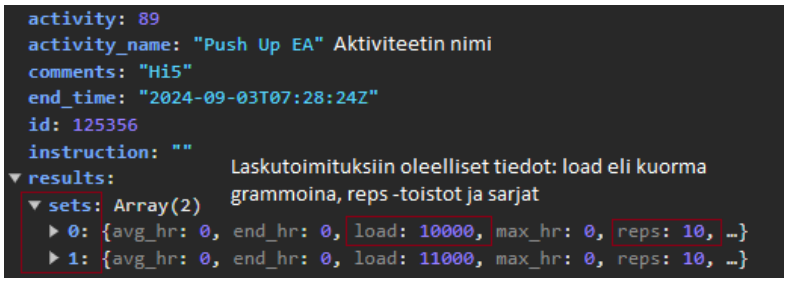


Kuva 16. Valmiit drop-down listat treeniohjelmien ja aktiviteettien valinnoille

4.4 Datan visualisointi

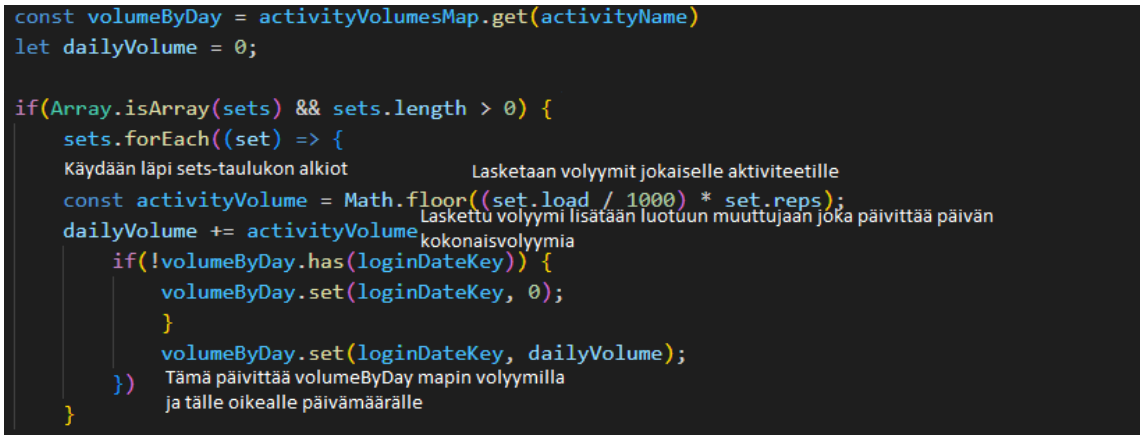
Seuraava haastavin ja kompleksisin vaihe oli datan visualisointi. Ensimmäiseksi täytyi päättää mitä tietoa haluaisimme näyttää, koska dataa oli paljon – mahdollisuuksia oli monia. Selkeä valinta oli käyttäjän kokonaisvaltainen edistyminen, tämä tulisi antamaan suuntaa siihen, miten käyttäjän treenit ovat kehittyneet treenien aloittamisesta nykyaikaan. Ja koska halusimme kaavioita kuvastamaan käyttäjien treenien kulkua, treenin kokonaiskuorma vaikutti erinomaiselta ratkaisulta tähän tilanteeseen. Kokonaiskuorma toiselta nimeltään treenivolyymi, joka on yksinkertaistettuna treenin aikana nostettu kilomäärä. Lyhykäisyydessään volyymi lasketaan: Sarjat x Toistot x Käytetty vastus. Esimerkkinä jos penkkipunnerrusta tehdään viidelläkymmenellä kilolla, 3 sarjaa ja 5 toistoa saadaan treenivolyymiksi: 3 x 50 x 5 = 750kg (18). Yksi hyvä mittari oli myös käyttäjien treeneihin käytetty aika, jota voitaisiin kuvata kuvaajan avulla.

Nyt kun metodit ja suunnitelma oli valmiina, seuraava tavoite oli päättää ensimmäisen osion eli kokonaisvaltaisen kehityksen laskeminen ja visualisointi. Jo edellä mainitut haetut käyttäjän treenit tulivat tärkeään rooliin tässä vaiheessa. Kuvassa esitän mitä tietoja tulimme seuraavassa osiossa käyttämään. (Kuva 17).



Kuva 17. Haettu treeni, ja sen sisällä oleva aktiviteetti objekti

Kuvassa näemme haetun treenin JSON objektin, joka pitää sisällään tärkeää tietoa kuten tässä tapauksessa oleellisimmat olivat aktiviteetit ja tämän objektin sisällä olevat parametrit: Sets, load ja reps. Näitä arvoja käyttämällä voimme laskea käyttäjälle treenivolyymin. (Kuva 18).



Kuva 18. Treenivolyymin lasku päivälle

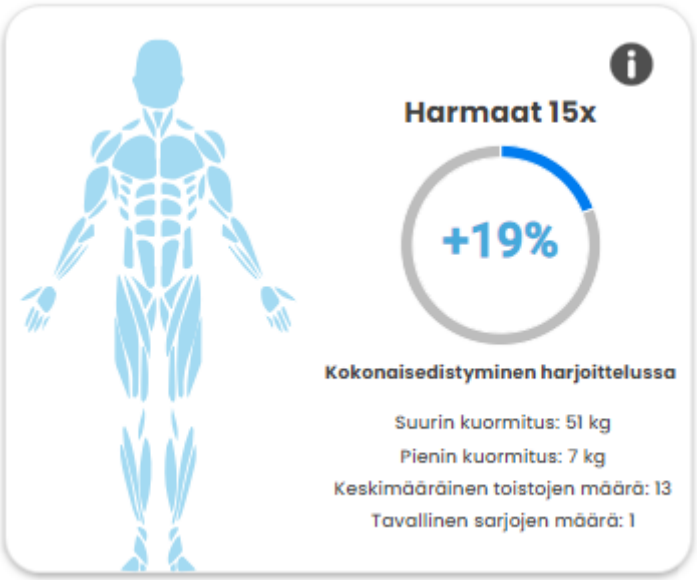
Tämä komento käydään läpi jokaiselle aktiviteetille, eli lyhykäisyydessään lopputulokseksi saadaan lista päivämääristä ja näinä päivinä tehdyistä treenien volyymeista. Tämän jälkeen luodaan funktio, joka vastaanottaa parametrina taulukon, joka sisältää haluamamme päivämäärien volyymit. Näistä arvoista lasketaan prosentuaalinen muutos aloitusvolyymista lopetusvolyymiin: $\frac{lopetusvolyymi - aloitusvolyymi}{aloitusvolyymi} * 100$

Saatu arvo talletetaan tilamuuttujaan ja välitetään uudelle komponentille, joka vastaa tämän arvon esittämisestä.

Samaan instanssiin haluttiin lisää tietoa treenien kulusta, joten jo saamastamme datasta otettiin aktiviteetit ja suodatettiin niitä tarkemmin. Tavoitteena oli näyttää treenien aktiviteeteista; suurin käytetty kuorma, pienin käytetty kuorma, keskimääräinen toistojen määrä sekä tavallinen sarjojen määrä. Tämän onnistuakseen olisi käytävä kaikki aktiviteetit läpi ja poimia sieltä etsimämme arvot. (Kuva 19).

Kuva 19. Käydään aktiviteettien sarjat läpi ja matkalta poimitaan tarvittavat tiedot

Lisäksi vielä kerätyistä toistoista ja sarjojen kokonaismäärästä lasketaan keskiarvot ja näin saadaan kaikki tarvittavat arvot esitystä varten. Lopputulokseksi saatiin näkymä, josta asiakas näkee prosentuaalisen kehityksensä sekä yleiskuvaa aktiviteeteista. (Kuva 20).



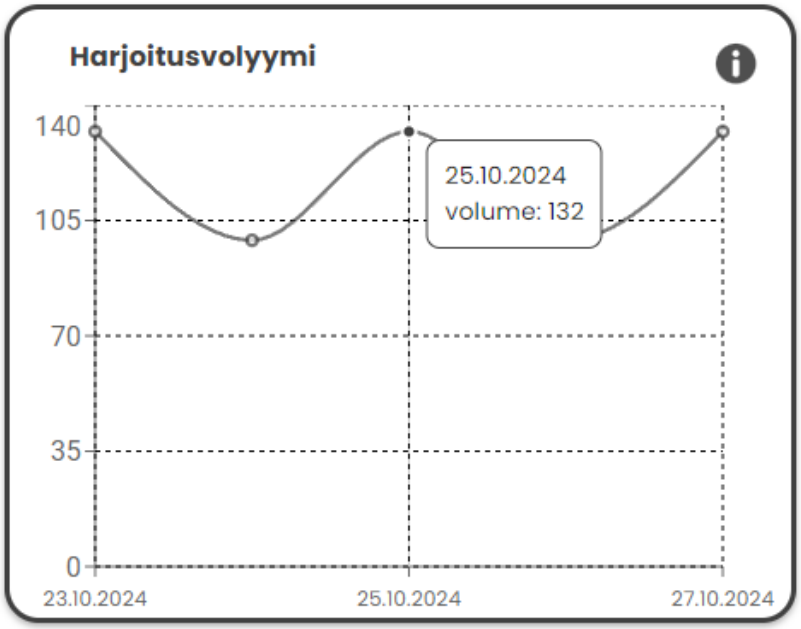
Kuva 20. Näkymä prosentuaalisesta kehityksestä

Kuvaajien näyttäminen antaa asiakkaalle suoraa osviittaa siitä, miten treenien kulku menee. Prosentuaaliset arvot voivat olla joskus hieman hämmentäviä sen takia päätettiin sen rinnalle näyttää myös kuvaaja harjoitusvolyymista eli harjoituksen kokonaiskuormasta. Tämän lisäksi myös aika-aspekti voi olla kiinnostava tietää, joten kuvaaja harjoituksen kestosta päätettiin lisätä.

Ensimmäiseksi lähestyin kokonaiskuorman kuvaajaa. Tämä oli mahdollista toteuttaa helposti, sillä minulla oli jo valmis taulukko, joka sisälsi päivämäärät sekä näinä päivinä tehdyt volyymimäärät. Jo aiemmin tehdyssä piirakkadiagrammissa törmäsin uuteen kirjastoon nimeltä recharts, joka osoittautui erinomaiseksi työkaluksi myös tähän tarkoitukseen.

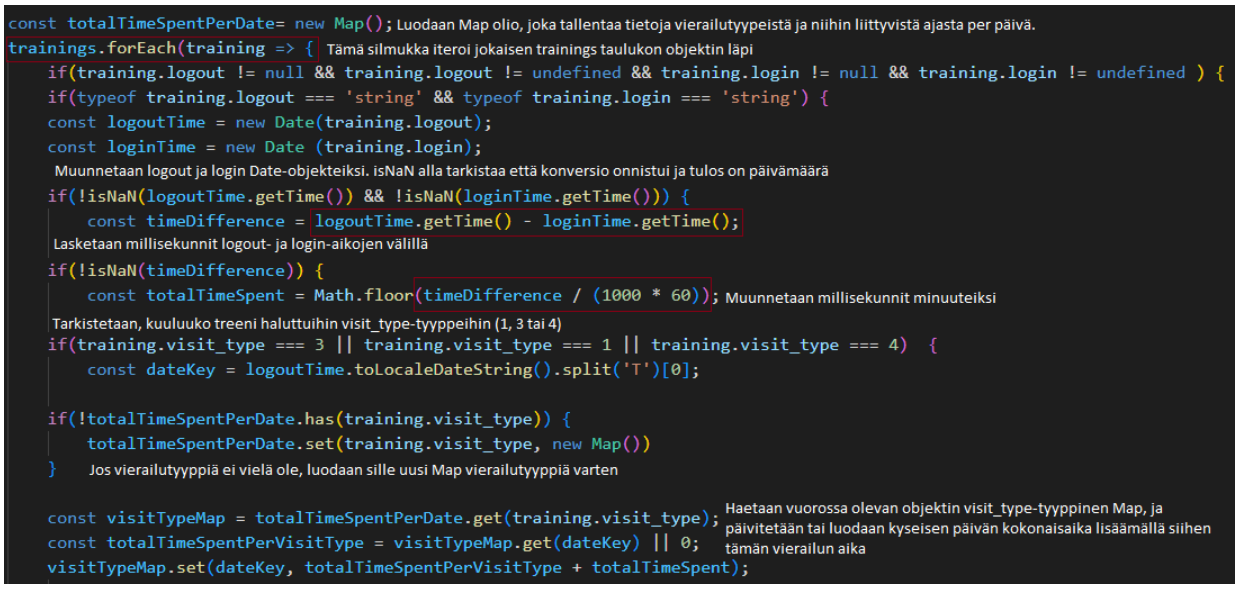
Recharts on suosittu MIT-lisensioitu kirjasto, joka on suunniteltu erityisesti kaavioiden luomiseen React sovelluksissa. Sen vahvuus on helppokäyttöisyydessä ja komponenttien selkeässä rakenteessa. Kirjasto hyödyntää SVG-tekniikkaa kaavioiden piirtämiseen ja käyttää taustalla kevyitä osia D3-kirjastosta, joka on tunnettu tehokkaista tietojen visualisointiominaisuuksista. Kirjastosta löytyy useita sisäänrakennettuja komponentteja kuten BarChart, LineChart, ja esimerkiksi aiemmin käyttämäni pieChart, joita yhdistelemällä voidaan luoda monipuolisia ja interaktiivisia kaavioita. Tässä tapauksessa haluttiin viivadiagrammi, ja yhdessä muiden

tukikomponenttien kanssa sain luotua ideaalin responsiivisen kuvaajan esittämään kokonaiskuormaa. (Kuva 21) (19).



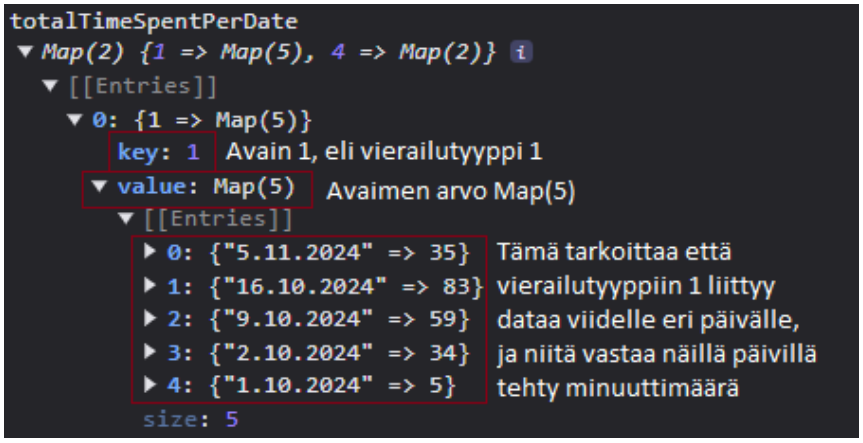
Kuva 21. Viivadiagrammi kuvastaa asiakkaan harjoitusvolyymia

Lopuksi päätettiin toteuttaa kuvaaja, joka havainnollistaa asiakkaan ajankäyttöä. Tähän valitsin aiemmin mainitun Recharts-kirjaston pylväsdiagrammin. Datan valmistelu tätä kuvaajaa varten vaati kuitenkin hieman enemmän työtä, sillä arvot eivät olleet suoraan käytettävissä. Ajankäytön arvot johdettiin treenien login- ja logout-arvojen erotuksesta, joka määritteli kunkin harjoituksen pituuden minuuteissa. Tämä tarkoitti, että jokainen treeni piti erikseen käsitellä, poimia tarvittavat arvot ja laskea niiden aikaväli (kuva 22). Koska järjestelmä tukee myös kotiharjoituksia, oli tärkeää tarkistaa myös jokaisen vierailutyyppi. Näin saliharjoitukset ja kotiharjoitukset voitiin erotella toisistaan ja esittää graafissa oikein omissa kategorioissaan. Tämä erottelu varmistaa, että asiakkaan ajankäytön visualisointi on tarkka ja helposti tulkittava.



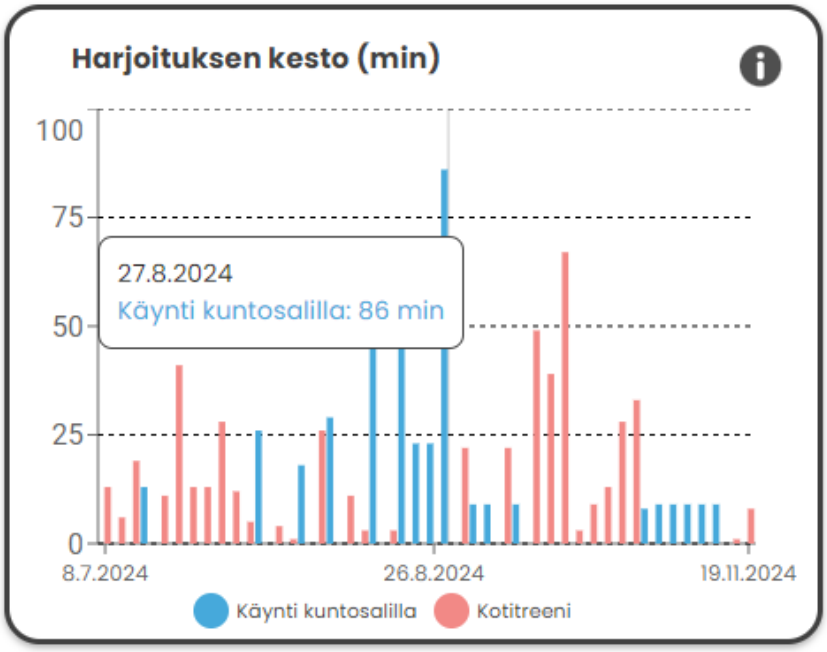
Kuva 22. Kokonaisuus, joka ottaa trainings-taulukon sisältä objektin ja laskee treenin ajan

Yllä olevan koodin tarkoituksena on laskea asiakkaiden käyttämä kokonaisaika minuutteina jokaiselle päivälle ja vierailutyypille. Tulokset tallennetaan muuttujaan totalTimeSpentPerDate, joka on Map-rakenteeltaan. (Kuva 23.) Tämä rakenne sisältää avain-arvo-pareja, joissa: Avain(key) edustaa treenin visit_type arvoa ja arvo(value) toinen Map joka säilöö tietoja päivämääräkohtaisesti. Sisäinen Map sisältää: Avain(dateKey) mikä edustaa päivämäärää, ja arvo on kyseisen päivän aikana kertyneiden minuuttien summa.



Kuva 23. totalTimeSpentPerDate Map:in rakenne

Tämä rakenne on hyödyllinen, kun halutaan analysoida tietoa vierailutyypeittäin ja päivämäärittäin. Nyt voidaan selkeästi erottaa salitreenit ja kotitreenit toisistaan ja antaa niille graafinen kuvaaja (kuva 24).



Kuva 24. harjoituksen kesto pylväsdiagrammi

5 POHDINTA

Tämän opinnäytetyön tavoitteena oli kehittää olemassa olevaa sovellusta HUR Oy:n ohjelmistoyksikölle. Sovellus oli saanut asiakaslähtöistä palautetta uusista ominaisuuksista. Tämä nosti projektin keskiöön todellisten käyttäjien tarpeet ja odotukset, mikä toi työhön käytännönläheisen ja asiakaslähtöisen näkökulman. Kehitystyö saatiin toteutettua määrittelyjen mukaisesti aikataulussa.

Aihe oli hyvin mielenkiintoinen ja samalla riittävän laaja itsensä haastamiselle. Sovelluksen kehitys tarjosi monipuolisen katsauksen web-sovelluskehityksen keskeisiin haasteisiin ja mahdollisuuksiin. Projektissa hyödynnettiin ennestään tuttuja teknologioita, kuten React ja TypeScript, mikä mahdollisti syvemmän keskittymisen ohjelmistokehityksen toteutukseen ja osaamisen laajentamiseen. Lisäksi projektiin kuului useiden uusien teknologioiden, kuten AWS-pilvipalveluiden, Django-Frameworkin ja uusien React-komponenttikirjastojen hyödyntämisen. Näiden uusien työkalujen käyttöönotto kehitti merkittävästi teknistä osaamista ja tarjosi samalla arvokasta kokemusta uusien teknologioiden omaksumista osaksi laajempaa kehitysprosessia. Opinnäytetyön tekeminen kokonaisuudessaan antoi merkittäviä työkaluja tulevaisuuteen kokonaisuuksien hahmottamisesta ja kehityksestä.

Projektin aikana korostuivat myös työelämässä keskeiset taidot, kuten palaverikäytännöt ja säännöllinen viestintä. Edistymistä raportoitiin päivittäin toimeksiantajan ohjelmisto-osaston vetäjälle, sekä viikottaisissa sprinttipalavereissa edellisen version kehittäjän kanssa. Tämä yhteistyö edesauttoi projektin sujuvaa etenemistä, eikä suurempia ongelmia esiintynyt missään vaiheessa.

Projektin lopputulosta voidaan pitää onnistuneena niin aikataulun, toiminnallisten vaatimusten kuin laadun osalta. Kehitetty ominaisuus saatiin testattavaksi asiakkaille, ja palaute on tähän mennessä ollut pääosin positiivista. Sovelluksen jatkokehitys on luonteva askel tämä opinnäytetyön tulosten pohjalta. Kehitystyö loi hyvän perustan sovelluksen tuleville parannuksille. Yksi merkittävimmistä tulevaisuuden kehityssuunnista on käyttöliittymän kokonaisvaltainen uudistaminen.

LÄHTEET

1. Love, Chris 2018. Progressive Web Application Development by Example. Yhdysvallat: O'Reilly Media. Hakupäivä 22.11.2024. O'Reilly for Higher Education. Vaatii käyttöoikeuden.

2. Ater Tal 2017. Building Progressive Web Apps. Yhdysvallat: O'Reilly Media. Hakupäivä 16.12.2024. O'Reilly for Higher Education. Vaatii käyttöoikeuden.

3. Ritika, 2023. What is TypeScript? Definition, history, features and uses. Hakupäivä 14.9.2024. https://invedus.com/blog/what-is-typescript-definition-history-features-and-uses-of-typescript/

4. Banks, Alex & Porcello, Eve 2020, Learning React. Modern Patterns for Developing React Apps. 2. Painos. Yhdysvallat: O'Reilly Media. Hakupäivä 22.11.2024. O'Reilly for Higher Education. Vaatii käyttöoikeuden.

5. Geeksforgeeks 2024. React Components. Hakupäivä 13.12.2024. https://www.geeksforgeeks.org/reactjs-components/?ref=header_outind

6. Geeksforgeeks 2023. State Management in React – Hooks, Context API and Redux. Hakupäivä 16.12.2024. https://www.geeksforgeeks.org/state-management-in-react-hooks-context-api-and-redux/

7. Adeoye Tito 5.2.2024, React Hooks: useState (With Practical Examples). Hakupäivä 16.12.2024. https://medium.com/@titoadeoye/react-hooks-usestate-with-practical-examples-64abd6df6471

8. Geeksforgeeks 2024. React useState Hook. Hakupäivä 14.12.2024. https://www.geeksforgeeks.org/reactjs-usestate-hook/

9. Herrera Esteban 2024. useState in React: A complete guide. Hakupäivä 16.12.2024. https://blog.logrocket.com/guide-usestate-react/

10. Geeksforgeeks 2024c. ReactJS useEffect Hook. Hakupäivä 14.12.2024. https://www.geeksforgeeks.org/reactjs-useeffect-hook/

11. Jain Pulkit 2024. What is a Database? Everything You Need to Know. Hakupäivä 13.12.2024. https://www.simplilearn.com/tutorials/dbms-tutorial/what-is-a-database

12. Jan L. Harrington April, 2016, Relational Database Design and Implementation, 4th Edition. Yhdysvallat: O'Reilly Media. Hakupäivä 13.12.2024. O'Reilly for Higher Education. Vaatii käyttöoikeuden.

13. Geegksforgeeks 2024. What is Database? Hakupäivä 13.12.2024. https://www.geeksforgeeks.org/what-is-database/

14. Selvaraj Sivaraj 2024. Mastering Rest APIs: Boosting Your Web Development Journey with Advanced API Techniques. Yhdysvallat: O'Reilly Media. Hakupäivä 16.12.2024. O'Reilly for Higher Education. Vaatii käyttöoikeuden.

15. wuweiweiwu 2019. Luettu 15.9.2024. https://github.com/bvaughn/react-virtualized/blob/master/docs/usingAutoSizer.md

16. Sam Robbins 2022 A beginners guide to JSON,the data format for the internet. Hakupäivä 21.9.2024. https://stackoverflow.blog/2022/06/02/a-beginners-guide-to-json-the-data-format-for-the-internet/

17. Material-UI. React components for faster and easier web-development. Hakupäivä 17.12.2024. https://mui.com/material-ui/getting-started/

18. Martikkala Teresa 2016. Treenivolyymi ja -intesiteetti. Hakupäivä 21.9.2024. https://healthisamatterofchoice.blogspot.com/2016/09/treenivolyymi-ja-intensiteetti.html

19. Refine 2024. Create charts using Recharts Hakupäivä 8.11.2024.
https://refine.dev/blog/recharts/#what-is-recharts