jamk

GymDiary-verkkosovelluksen kehitys

Kristian Pekkanen

Opinnäytetyö, AMK
Marraskuu 2024
Insinööri (AMK), tieto- ja viestintätekniikka

jamk | Jyväskylän ammattikorkeakoulu
University of Applied Sciences

jamk

Kuvailulehti

Kristian Pekkanen

GymDiary-verkkosovelluksen kehitys

Jyväskylä: Jyväskylän ammattikorkeakoulu. Marraskuu 2024, 45 sivua.

Tieto- ja viestintätekniikan tutkinto-ohjelma. Opinnäytetyö AMK.

Julkaisun kieli: suomi

Julkaisulupa avoimessa verkossa: Kyllä

Tiivistelmä

Opinnäytetyön tavoitteena oli rakentaa verkkosovellus, joka helpottaisi kuntosaliharjoitteiden tallentamista ja seuraamista. Verkkosovelluksen pääpiirteinä oli helppokäyttöisyys, nopeus ja yhteensopivuus erilaisten päätelaitteiden välillä. Tarpeena oli luoda palvelu, jonka avulla käyttäjät voisivat kirjata kuntosaliharjoitteensa, hallita tehtyjä harjoitteita ja seurata kehitystä graafien avulla.

Ensin analysoitiin markkinoilla olevia sovelluksia ja tarkasteltiin niiden arvosteluja. Tämän jälkeen määriteltiin ominaisuuksia, jotka nousivat keskeisiksi arvosteluissa. Sovelluksen käyttöliittymässä käyttäjille tarjottiin kirjautumisen jälkeen mahdollisuus luoda harjoitepohjia ja muokata niitä, lisätä toteutuneita harjoitteita, selata harjoitteita harjoitepohjien perusteella ja tarkastella progressiota graafeista.

Verkkosovellus toteutettiin ajankohtaisilla ja verkkosovelluskehityksessä relevanteilla teknologioilla. Kehitystyö suoritettiin paikallisessa ympäristössä palvelimen ja verkkosivun osalta. Mobiililaitekehitys suoritettiin Android Studion emulaattorilla. Kehityksessä käytettiin avoimen lähdekoodin JavaScript-kehyskirjastoa Vue.js. Datan visualisoinnissa käytettiin D3.js-kirjastoa. Palvelinpuoli rakennettiin Node.js-ympäristössä käyttäen Express-sovelluskehystä. Palvelin oli yhteydessä MongoDB-tietokantaan. Nämä teknologiat on valittu joustavuuden, skaalautuvuuden ja kehittämiskokemuksen perustein.

Lopputuloksena syntyi selkeä ja toimiva verkkosovellus, joka mahdollisti kuntosaliharjoitteiden luomisen, nopean ja toimivan harjoitteiden merkkaamisen ja progression seuraamisen graafien muodossa. Lopputulos toimi hyvin mobiililaitteessa kuin myös tietokoneella ja täytti asetetut tavoitteet. Valitut teknologiat vastasivat hyvin tavoitteita. Verkkosovelluksen jatkokehitykselle jää myös mahdollisuus laajentaa sen toiminnallisuuksia ja lisätä erilaisia ominaisuuksia, jotka ovat hyödyllisiä kuntosaliharjoitteiden seurannassa.

Avainsanat (asiasanat)

Web-kehitys, Vue, Node.js, Express, MongoDB, D3.js, Kuntosali, Seurantatyökalu, Fullstack-kehitys, JavaScript

Muut tiedot (salassa pidettävät liitteet)

-

jamk

Description

Kristian Pekkanen

Development of the GymDiary web application

Jyväskylä: JAMK University of Applied Sciences, November 2024, 45 pages.

Degree Programme in Information and communication technology Engineering. Bachelor's thesis.

Permission for open access publication: Yes

Language of publication: Finnish

Abstract

The goal of the thesis was to build a web application that would make it easier to track gym workouts. The main features of the web application were ease of use, speed and compatibility across different devices. The aim was to create a service that would allow users to log their gym workouts, manage completed exercises and track progress using graphs.

First, existing applications on the market were analyzed and their reviews were examined. Based on this, key features that were highly rated in the reviews were identified. In the application's interface, users were offered the possibility to create and modify workout templates, add completed exercises, browse exercises based on the templates, and view progress through graphs after logging in.

The web application was built using up-to-date and relevant technologies for web development. Development was done in a local enviroment. Mobile device development was done using the Android Studio emulator. The development was done with open-source JavaScript framework Vue.js. D3.js was used for data visualization. The server-side was built in a Node.js enviroment using the Express framework, and the server was connected to a MongoDB database. These technologies were chosen based on flexibility, scalability and developer experience.

The result was a clear and functional web application that made possible to create gym workouts, quick and efficient logging of exercises and tracking progress through graphs. The final product worked well on both mobile devices and computers and met the set goals. The chosen technologies supported the objectives effectively. There is also potential for future development of web application to expand its functionality and add additional features that would be useful for tracking gym workouts.

Keywords/tags (subjects)

Web development, Vue, Node.js, Express, MongoDB, D3.js, Gym, Tracking tool, Fullstack-development, JavaScript

Miscellaneous (Confidential information)

-

Sisältö

Kuviot

Taulukot

1 Johdanto

1.1 Työn lähtökohta

Ideana oli toteuttaa opinnäytetyönä oma kuntosalitulosten seurantatyökalu. Tarve seurantatyökalusta tuli opinnäytetyöntekijän omasta tarpeesta. Sopivaa valmista sovellusta ei löytynyt, sillä osa tärkeistä ominaisuuksista oli maksumuurin takana. Tiettyyn pisteeseen asti puhelimen muistiinpanot-sovellukseen on hyvä lisätä esimerkiksi päivämäärä, tehty liike, sarjapaino, toistot ja sarjat. Kun kuntosalilla käyntikertoja kertyy esimerkiksi puolenvuoden ajalta, muistio alkaa olla pitkä. Tiedon hakeminen ja lisääminen ei ole enää nopeaa ja yksinkertaista.

Tiedonhaku puhelimen muistiosta oli epäkäytännöllistä. Tiedosta ei myöskään pystynyt rakentamaan minkäänlaisia graafeja progression vahvistamiseksi. Koska kuntosaliharrastus oli vasta alkuvaiheessa, olisi hyvä seurata, että kehitystä tapahtuu. Tämä motivoi kehittämään oman seurantatyökalun, joka mahdollistaa tehokaan tiedon tallentamisen ja analysoinnin.

1.2 Kuntosaliharrastus

Kuntosaliharrastus on yksi suosituimmista liikuntamuodoista ja se tarjoaa monia fyysisiä ja henkisiä terveyshyötyjä. Säännöllinen kuntosaliharjoittelu kehittää voimaa, kestävyyttä ja yleistä hyvinvointia (Semeco, 2017). Kuntosaliharrastajien on tärkeää seurata kehitystä, jotta he voivat asettaa uusia tavoitteita ja nähdä edistymisensä konkreettisessa muodossa. Kehityksen seuraaminen on erityisen tärkeää, kun harjoittelua tehdään tavoitteellisesti.

1.3 Tavoite

Tarkoituksena oli luoda selainpohjainen sovellus. Kirjautumisen jälkeen on mahdollista lisätä kuntosaliharjoitteita, tarkastella edellisiä harjoitteita ja nähdä graafeja potentiaalisesta kehityksestä. Tämä käyttöliittymällinen tietojen tallennuspalvelu helpottaa oman datan käsittelyä, tutkimista ja analysointia.

Opinnäytetyössä on ensin tutkittu minkälaisia sovelluksia markkinoilla tarjolla. Tästä sai hyvää pohjaa sovelluksen suunnitteluun. Esimerkiksi helposti lisättävät hyvät ominaisuudet on hyvä ottaa oman sovelluksen kehityksessä harkintaan. Lisäksi tämä esittää kompastuskivet, joita muiden luomat sovellukset omaavat.

1.4 Tutkimusasetelma

Työssä keskityttiin kehittämään selainpohjainen työkalu kuntosalitulosten seurantaan. Tavoitteena oli selvittää, millaisia ominaisuuksia ja toiminnallisuuksia käyttäjät arvostavat, sekä kuinka nämä voidaan toteuttaa tehokkaasti selainpohjaisessa sovelluksessa. Lisäksi työssä perehdyttiin valittuihin teknologioihin, jotka mahdollistavat tehokkaan fullstack-sovelluksen kehittämisen. Tärkeimpiä kysymyksiä tällaisen sovelluksen suunnittelussa ovat:

1. Miten luodaan toimiva verkossa toimiva sovellus?
2. Millaisia tarpeita ja odotuksia käyttäjältä on seurantatyökalulle?
3. Mitkä ominaisuudet ovat tärkeitä kuntosaliharjoitteiden seuraamisessa?

Nämä kysymykset auttavat luomaan toimivan ja käytännöllisen sovelluksen. Sovelluksen suunnittelun lähtökohtana oli täyttää tärkeimmät kriteerit, kuten käyttäjien mahdollisuus seurata kuntosaliprogressiota, esimerkiksi graafien avulla.

Keskeinen haaste on nykyaikaisten ilmaisversioiden puutteelliset ominaisuudet. Vaikka monia ominaisuuksia on kehitetty, useimmat sovellukset on tehty taloudellinen hyöty mielessä ja ne ovat usein liian monimutkaisia. Tämä sovellus ei ole tarkoitettu toimimaan henkilökohtaisena valmentajana, vaan yksinkertaisena ja selkeänä työkaluna harjoittelun seuraamiseen ja kehityksen tarkasteluun.

2 Kuntosalisovellusten hyöty

Tschabitcher (2024) mukaan ominaisuudet, joita hän pitää tärkeinä arvostelukriteereiden perusteella ovat sovelluksen yksinkertainen käyttö, kuntosaliharjoitteiden yksinkertainen muokkaami-

nen, yksinkertainen tulosten lisääminen, toimiva sovelluksen ulkoasu, oman datan oman laitteeseen vieminen, uuden tuloksen lisäämisen aikana näkee edelliset tulokset samasta liikkeestä, valmiita treenipohjia ja toimivat graafit edellisistä tuloksista.

Tschabitcher (2024) mainitsee myös, että asiat, joita ei pidä hyvinä puolina ovat puuttuvat ominaisuudet kuten painojen laskuri. Eli esimerkiksi tarvittavat levypainot haluttuun painoon, kehon mittojen seuraaminen, rajoittunut kuntosaliharjoitteiden muokkaaminen ja ei ole mahdollista viedä dataasi omalle laitteelle.

2.1 Kuntosalisovellukset

2.1.1 Tarkempi tutkinta

Opinnäytetyössä etsittiin valmiita sovelluksia Google Play -sovelluskaupasta, josta löytyi muutama tutkittava vaihtoehto. Hakusanana käytettiin ”Workout Tracker”. Taulukkossa 1 on näkyvissä tietoa sovelluksista. Tarkastelun kohteeksi otettiin eniten ladatuimpia sovelluksia. Sovelluksia testattiin Samsung-puhelimella, jossa oli Android-versio 14.

Taulukko 1. Tutkittujen sovelluksien tiedot

Sovelluksen nimi	**Valmistaja**	**Versio**	**Lataus kerrat**
Workout Planner Muscle Booster	Welltech Apps Limited	3.19.1	+10 miljoonaa
Fitify: Fitness, Home Workouts	Fitify Workouts s.r.o.	1.68.1	+10 miljoonaa
RepCount Gym Workout Tracker	Siper Apps	4.5.6	+100 tuhatta
Workout Planner Gym&Home:FitAi	Social Tech Inc	1.2.8	+1 miljoona

2.1.2 Yhteisiä ominaisuuksia

Yhtenäisenä ominaisuutena huomattiin, että käyttäjän luotua sovellus kysyy paljon perustietoja käyttäjistä. Näihin kesti keskimäärin 2-3 minuuttiin vastata. Tämä on vaadittua esimerkiksi personoitujen kuntosaliohjelmien luomiseen. Yleisimmät kysymykset olivat:

1. Mikä on tavoite?
2. Paino?
3. Pituus?
4. Käyttäjän kropan tyyppi?
5. Mikä on käyttäjän toivottu kehon tyyppi?
6. Mitkä kehon osat haluat priorisoida?
7. Onko edellistä kokemusta kuntoilusta?

Sovellukset tarjosivat kysymysten jälkeen maksumuurin takana olevia kuntosaliohjelmia. Jos ei halua maksaa sovelluksen käytöstä, oli vaihtoehto jatkaa ilman maksamista. Kuitenkin sovellukset tarjosivat usein ”popup”-tyylisiä alennuksia erityisinä tarjouksina erilaisista ominaisuuksista.

”RepCount Gym Workout Tracker” on näistä sovelluksista lähimpänä sovellusta, jota opinnäytetyössä luodaan. Sovelluksessa pystyi luomaan yksinkertaisella käyttöliittymällä omia ohjelmia. Ohjelmia pystyi toteuttamaan erillisinä harjoittelusessioina. Tämä sovellus on toimiva vaihtoehto korvaamaan puhelimen muistiosovelluksen. Valitettavasti statistiikat olivat kuitenkin maksumuurin takana tässäkin sovelluksessa.

2.1.3 Hyvät ominaisuudet

Aloittaville kuntosaliharrastajille sovelluksen luomat kuntosaliohjelmat ovat varmasti toimiva ratkaisu perusasioiden perehtymiseen, mutta usein kokeneemmalla harrastajalla on joko hänelle henkilökohtaisesti suunniteltu kuntosaliohjelma tai hän on valmennuksessa, jossa hänelle muutetaan tarvittaessa ohjelmaa. Valmiin kuntosaliohjelman luomisesta voisi kysyä käyttäjältä, ennen tämän luomista.

Sovelluksista löytyi myös sarjojen taukoajastin. Tietynlaisissa kuntosaliohjelmissa on määritetty tarkkaan taukojen pituudet, joten tämä ominaisuus on hyödyllinen ja kohtuullisen helppo kehittää.

Osa sovelluksista tarjosi myös maksumuurin takana olevia ravinto-ohjelmia. Näissä yleensä löytyi uuden asiakkaan alennus, jotta asiakkaalta saadaan vain pienellä summalla aloittamaan sovelluksesta maksamisen.

Sovelluksista pääsääntöisesti löytyi graafeja kuntosaliharjoittelun aikana tehtävistä liikkeistä. Tämä on hyödyllinen ominaisuus uusille kuntosaliharrastajille uusien liikkeiden oppimiseen.

2.1.4 Parannus ideat

Usein kriittiset ominaisuudet olivat maksumuurien takana. Tietysti kaupallistettujen sovelluksien täytyy olla tuottavia. Vaihtoehtoisesti sovelluksesta voisi tehdä ilmaisia käyttää, mutta tuotto täytyisi sitten saada joko mainoksilla tai myymällä ihmisten tietoa. Näistä kumpikaan ei ole hyvä vaihtoehto. Oman sovelluksen luonnissa, ei ole tavoite tehdä rahaa. Tavoite on tehdä toimiva oma versio tarvittavilla ominaisuuksilla.

Myös omien kuntosaliohjelmien tuominen sovelluksiin oli tehty vaikeaksi. Joko täytyi maksaa ohjelman muuttamisesta tai ei ollut vaihtoehtoa kirjoittaa omia liikkeitä sovellukseen.

Olisi hyvä, että sovelluksessa olisi myös mahdollisuus viedä kaikki aikaisemmat treenit omaan laitteeseen esimerkiksi csv-tiedostomuodossa.

2.1.5 Miksi luoda uusi sovellus?

Kun kehitetään tämän tyyppinen työkalun, kehittäjä tietää miten tietoja käytetään. Uusia ominaisuuksia pystytään lisäämään, kun niitä keksii ja maksumuureja ei tietenkään ole. Kaikkeen pystyy siis vaikuttamaan.

Jatkokehittäminen on varmasti osa tätä projektia. Aikaa on vain rajoitettu aika tämän projektin tekemisessä.

2.2 Miksi nämä sovellukset ovat hyödyllisiä?

Sovelluksien isot latauskerrat kertovat, että sovelluksille on kysyntää, mutta miksi? Tutkimus ”Vapaa-ajan liikunta ja fyysinen aktiivisuus lisääntyvät Suomessa WHO:n tavoitteiden mukaisesti” osoittaa, että riittämättömästi liikkuvia vapaa-ajan harrastajien osuus on ollut laskussa vuodesta 1982-2017 vuosien välisenä aikana (Wennman, Borodulin & Jousilahti 2019, 2). Tämän pohjalta voi päätellä, että kokonaisuudessaan liikunnan harrastaminen on kasvanut ihmisten kesken.

Kuntosaliharrastuksen terveyshyödyt heijastuvat myös arkielämään. Harrastuksen voi ottaa mukaan omaan arkeen tukemaan arjessa jo olevia aktiviteetteja, kuten raskaampien asioiden nostaminen maasta, painavien ostoskassien kantaminen tai muuttokuorman kantaminen. Kuntosaliharrastus antaa eväitä oikeanlaiseen nostamistekniikkaan, voimaan ja kestävyyteen.

Kuntosaliharjoittelemisen kulmakiviä ovat säännöllisyys, spesifisyys, nousujohteisuus ja yksilöllisyys. Suurin apu kuntosaliohjelmista tulee nousujohteisuuteen. Nousujohteisuus kuntosalilla tarkoittaa, että painoharjoittelun täytyy olla progressiivista. Käytännössä painoharjoitteen täytyy olla jollain tavalla raskaampi, kuin edellisellä kerralla. Raskaampi voi tarkoittaa esimerkiksi nostamalla isompia painoja, lisäämällä raskaita työsarjoja, harjoittelemalla useammin tai vaihtamalla liikettä raskaampaan (Mäennä, Olli, Puputti, Parkkinen, Roisinen, Kuukasjärvi & Haverinen 2019, 25). Tämän vahvistaa usea eri julkaisu, kuten myös julkaisussa ”the ultimate guide to building muscle mass” mainitaan, että ydintekijöitä lihaskasvuun ovat progressiivinen ylikuormittaminen, harjoittelu frekvenssi ja oikeanlainen harjoittelutekniikka (Heath N.d). Keskiverto kuntosaliharjoittelijakin on saattanut huomata, että ei voi tulla vahvemmaksi, jos ei haasta itseään ja työnnä itseään pois mukavuusalueelta.

Kun on tarkoitus harjoitella kuntosalilla nousujohteisesti, harjoittelijan täytyy muistaa, kuinka painavilla painoilla viimeksi teki, kuinka monta työsarjaa teki ja kuinka monta toistoa missäkin sarjassa tuli. Kun päivän kuntosaliharjoitteessa on useampi erilainen liike, jota harjoittelija tekee erilaisilla painoilla ja toistomäärillä, hänen täytyisi muistaa nämä painot ja määrät. Tätä ongelmaa varten on luotu sovelluksia, jotka muistavat nämä painot ja toistomäärät harjoittelijan puolesta.

3 Valitut teknologiat

3.1 Vue.js

Vue.js on avoimen lähdekoodin JavaScript-kehyskirjasto. Se on suunniteltu käyttöliittymien kehittämiseen web-sovelluksissa. Vuea käyttävät tuotannossa esimerkiksi Nasa, Apple, Google, Microsoft, Gitlab, Zoom, Tencent, Weibo ja monet muut. Teknologia on siis todettu isoissa teknologia yrityksissä toimivaksi ratkaisuksi. Vuea voi käyttää JavaScript- tai TypeScript-ohjelmointikielillä (Frequetly asked questions N.d).

Vuen laajan käyttäjäkunnan ansiosta sillä on helppo aloittaa haasteellisempikin projekti, koska materiaalia Vuen käyttämisestä ja eri komponenttien hyödyntämisestä löytyy paljon.

Vue.js kehittäjä on Evan You, joka julkaisi projektinsa vuonna 2014. Tästä eteenpäin Vue.js on ollut joukkueellisen kokoaika- ja vapaaehtoisienkehittäjien ylläpitämä, jota Evan You johtaa. Vue.js-projektia rahoitetaan sponsoroinnilla (Frequetly asked questions N.d).

Vue tarjoaa laajan valikoiman työkaluja näyttävän ja toimivan verkkosivun luomiseen pienellä oppimiskynnyksellä. Tämän lisäksi Vue on kilpailukykyinen tehokkuudessa muiden isojen teknologioiden, eli esimerkiksi angular ja react (Emadamerho-Atori 2023).

3.2 D3.js

D3.js on avoimen lähdekoodin JavaScript-kirjasto. Se on suunniteltu datan visualisointien luomiseen web-sovelluksissa. Sen on kehittänyt Mike Bostock vuonna 2011 (What is D3? N.d).

D3.js mahdollistaa monipuolisen ja interaktiivisen datagrafiikoiden luomisen verkkosivulle sille syötetystä datasta. Tämä on huomattava apu progression visualisointiin kuntosaliharjoitteidenseuranta-ohjelmassa. Kropan ulkoiset muutokset ei välttämättä kerro harjoittelijalle koko totuutta, vaan vaaditaan jo luotuja statistiikkoja painojen nostelusta osoittamaan kehitystä graafisessa muodossa.

D3.js vahvuuksista on joustava ja laaja yhteensopivuus erilaisien datalähteiden kanssa. Tämän vuoksi kehittäjän on helppo ottaa D3.js osaksi omaan projektiin. D3.js omaa myös kattavan verkkosivun, jossa on listattu ominaisuudet, koodi esimerkit ja muut tärkeät ominaisuudet kehittäjälle luettavaksi.

3.3 Node.js

Palvelimen ohjelmointi voi kuulostaa pelottavalle, mutta se ei kuitenkaan eroa muusta ohjelmoinnista juurikaan. Palvelinohjelmointiin kehitetty erilaisia hyviä työkaluja, joista yksi on Node.js. Node.js on avoimen lähdekoodin JavaScript-kirjasto. Se toimii monella eri alustalla eli ei ole sidottu käyttöjärjestelmään. Node.js pohjautuu Google Chromen ytimeen selaimen ulkopuolella (Node.js tutorial in Visual Studio Code 2024). Tämä ei kuitenkaan tarkoita sitä, että Node.js liittyy selaimiin, vaan Node.js-kirjastoa käytetään palvelinpuolen ohjelmoinnissa sen laajan ja monipuolisten ominaisuuksien vuoksi. Se mahdollistaa tehokkaat ja skaalautuvat palvelinominaisuudet, joissa tarvitaan reaaliaikaista tietojenkäsittelyä. Näitä ominaisuuksia kutsutaan lyhenteellä ”CRUD”, joka tarkoittaa create, read, update ja delete. Eli luo, lue, päivitä ja poista. Näillä ominaisuuksilla pystytään toteuttaa erilaisia palvelinpuolen ratkaisuja.

Nodea varten on myös kehitetty sovelluskehyksiä. Yksi sovelluskehys on nimeltään Express. Express tarjoaa palvelinpuolen kehittämiseen hyödyllisiä työkaluja. Swarnabha mainitsee julkaisussaan avainominaisuuksiksi reitityksen, välikäsittelyn, mallien, virheiden käsittelyn ja helppokäyttöisyyden (Swarnabha 2023). Näiden lisäksi geeksforgeeks-verkkosivun julkaisussa mainitaan lisäksi pyyntö- ja vastausobjekti ja tiedon jäsentäminen uniikeiksi ominaisuuksiksi (Unique features of Express JS 2023). Nämä ominaisuudet helpottavat ominaisuuksien luomista palvelinpuolelle. Palvelinpuoli on vastuussa verkkosivun datan käsittelystä, joka on yksi ydin ominaisuus verkkosivussa. Palvelimen pitää osata suodattaa mahdolliset hyökkäykset, selvitä vikatilanteista ja vastata niihin oikein, käsitellä verkkosivulta saatu data. Palvelinpuoli toimii käytännössä suodattimena ja datan käsittelijänä verkkosivun ja tietokannan välissä.

3.4 MongoDB

Tiedon tallentamiseen tarvitaan jonkinlainen tietokanta. Paljon erilaisia vaihtoehtoja ja ratkaisuja on tähänkin, mutta valittu tietokantapalvelun tarjoaja on MongoDB. MongoDB on dokumenttitietokanta, joka perustuu horisontaaliseen skaalautuvuusarkkitehtuuriin. MongoDB perustettiin vuonna 2007 ja sillä on laaja kannattajakunta kehittäjäyhteisössä (Why use MongoDB and When to Use It? N.d). MongoDB tietokantaa ei voi verrata perinteiseen SQL-tietokantaan, joka tallentaa tiedot tauluina eli riveinä ja sarakkeina (What is SQL? 2024). Jokainen MongoDB tietue on BSON-muodossa. MongoDB luovuttaa tiedot kuitenkin JSON-muodossa.

Dokumenttitietokannat ovat joustavia, joten ne sopivat hyvin erilaisiin projekteihin. Joustavuus syntyy siitä, että MongoDB sallii vaihtelut dokumenttien rakenteissa ja tallentaa osittain täydellisiä dokumentteja. Yksi dokumentti voi myös sisältää toisia dokumentteja upotettuna, mikä helpottaa monimutkaisten tietorakenteiden hallintaa. Tämä on pidetty ominaisuus kehittäjien kesken, koska se antaa vapaat kädet kehittäjän päättää siitä, minkälainen rakenne tallennetulla datalla on.

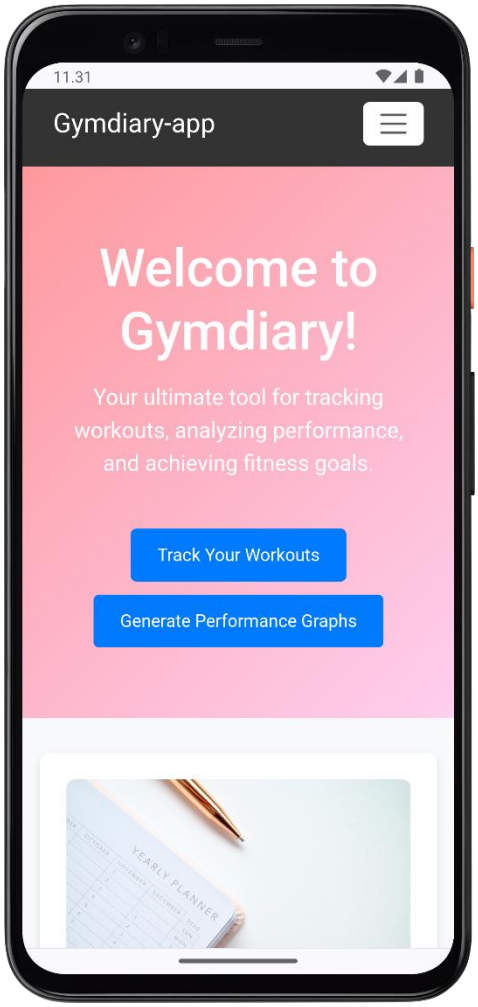
MondoDB:n käyttö on erityisen hyödyllistä sovelluksissa, joissa on paljon muutoksia tietojen rakenteessa tai kun tarvitaan nopeaa iterointia kehitystyössä. Hyvänä esimerkkinä toimii web-sovellukset, jotka ovat tyypillisiä käyttökohteita MongoDB:lle sen joustavuuden ja skaalautuvuuden ansiosta. MongoDB skaalautuu horisontaalisesti eli tietokantaa voidaan laajentaa lisäämällä uusia palvelimia, mikä mahdollistaa suuren tietomäärän käsittelyn ja järjestelmän suorituskyvyn parantamisen ilman merkittäviä haittoja. Reddemin mukaan tämä ei ole niin helppo vaihtoehto SQL-tietokannan kanssa, koska tietueiden hallinta on vaikeaa, kun käytössä on useita palvelimia hallitsemassa samaa dataa (Reddem 2019). Tämä ei kuitenkaan tarkoita sitä, että se on mahdotonta, mutta huomattavasti vaikeampaa kuin esimerkiksi MongoDB:n tapauksessa. MongoDB on suosittu vaihtoehto No-SQL tietokannoista, erityisesti aloittavien yritysten keskuudessa.

4 Sovelluskehitys

4.1 Kotisivu

Sovelluskehitys toteutettiin pääosin mobiililaitteet mielessä, koska palvelua käytetään pääsääntöisesti mobiililaitteilla. Käyttöliittymän tärkeimmät ominaisuudet ovat sen yksinkertaisuus, helppokäyttöisyys ja toimivuus. Kun tärkeimmät ominaisuudet ovat mielessä, on hyvä lähteä rakentamaan sitä. Suunnitteluvaihe ohitettiin täysin huomaamatta. Toteutus lähti liikkeelle kotisivusta, koska projekti alkoi omana sivuprojektina.

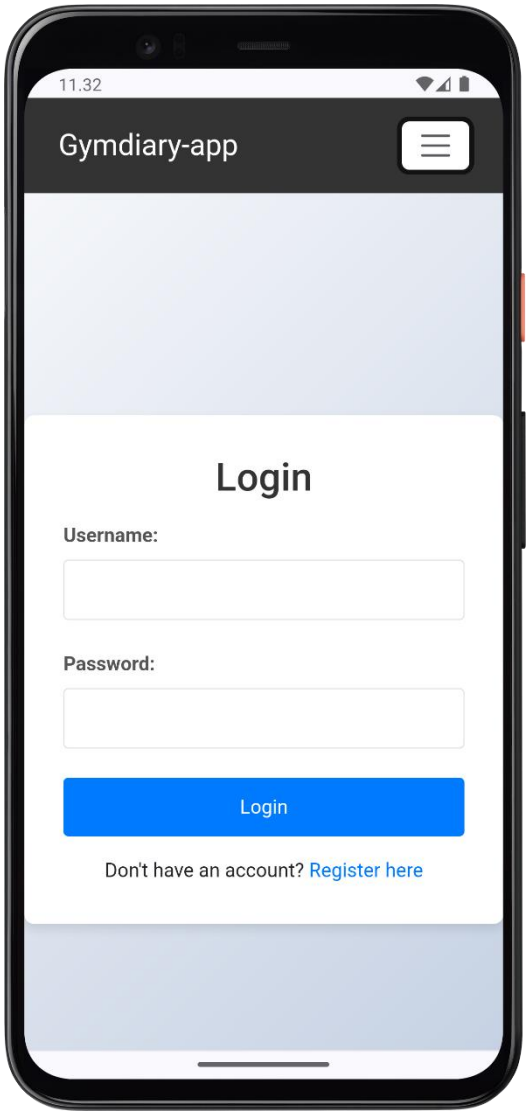
Kotisivu on yksinkertainen ja helposti käytettävä. Alaspäin selatessa listautuu alustan ominaisuuksia. Navigointipalkissa olevasta napista aukeaa käytettävissä olevat painikkeet ennen kirjautumista (ks. kuvio 1).



Kuvio 1. Kotisivun näkymä

4.2 Kirjautuminen

Kirjautuminen on tehty yksinkertaiseksi isoilla napeilla ja minimaalisella näkymällä. Kirjautumissivu on simppeli. Napit on suunniteltu isoiksi, joka helpottaa huomattavasti käytettävyyttä mobiililaitteilla. Kirjautumisen jälkeen aukeaa pudotusvalikkoon napit, joiden avulla pystyy käyttämään sovellusta. Kirjautumisen jälkeen käyttäjä ohjataan takaisin etusivulle (ks Kuvio 2).



Kuvio 2. Kirjautumissivun näkymä

API-kutsut on toteutettu Axios-nimisellä npm-paketilla perinteisen fetch-funktion sijaan. Axios tarjoaa yksinkertaisen, yhteensopivan ja varmatoimisen tavan keskustella palvelimen kanssa. Sen

etuihin kuuluu sisäänrakennettu tuki JSON-datan käsittelylle ja automaattinen virheiden käsittely. Axios myös mahdollistaa samanaikaiset pyynnöt. Tämä tekee Axiosta monipuolisen työkalun monenlaisiin projekteihin, jossa tarvitaan lutettavaa ja tehoksta HTTP-käsittelyä (Kamau 2024). Lähdekoodi tallentaa käyttäjän käyttäjänimen ja palvelimen lähettämän JSON Web Tokenin selaimen sessionStorage-muistiin käyttäjän varmentamista varten myöhemmäksi (ks Kuvio 3).

```vue
<template>
  <div class="login-page">
    <div class="login-container">
      <h1>Login</h1>
      <form @submit.prevent="login">
        <div class="form-group">
          <label for="username">Username:</label>
          <input type="text" id="username" v-model="username" required>
        </div>
        <div class="form-group">
          <label for="password">Password:</label>
          <input type="password" id="password" v-model="password" required>
        </div>
        <button type="submit">Login</button>
      </form>
      <div class="register-link">
        <p>Don't have an account? <RouterLink to="/register">Register here</RouterLink></p>
      </div>
    </div>
  </div>
</template>
<script setup>
import { ref } from 'vue';
import { useRouter } from 'vue-router';
import axios from 'axios';
import { useAuthStore } from '@/stores/auth';
import config from '../config';

const username = ref('');
const password = ref('');
const router = useRouter();
const authStore = useAuthStore();

const login = async () => {
  try {
    const response = await axios.post(`http://${config.apiHost}:${config.backendPort}/user/login`, {
      name: username.value,
      password: password.value
    });

    const { token } = response.data;
    authStore.login(username.value, token);
    router.push('/');
  } catch (error) {
    console.error('Login failed:', error);
  }
};
</script>
```

Kuvio 3. Kirjautumisen lähdekoodi

4.3 Autentikaatio

Autentikaatio tapahtuu jokaisen sivun vaihdoksen välissä. Kuviossa 4 on näkyvissä lähdekoodi sivunvaihdoksien välillä tapahtuvasta tarkistuksesta. Ennen jokaista sivunvaihdosta suoritetaan checkLoginStatus-funktio, joka palauttaa joko true- tai false-arvon. Jos kirjautumisen arvo on false,

käyttäjä ohjataan kirjautumissivulle. Jos kirjautumisen arvo on true, käyttäjä ohjataan sivulle, johon käyttäjä oli navigoimassa (ks Kuvio 4).

```
router.beforeEach(async (to, from, next) => {
  console.log('Navigating to:', to.path)

  const authStore = useAuthStore()

  const status = await authStore.checkLoginStatus()

  if (to.meta.requiresAuth && !status.isLoggedIn) {
    console.log('Redirecting to login due to no token')
    next('/login')
  } else {
    next()
  }
})
```

Kuvio 4. Sivun vaihdon välissä tapahtuva tarkistus

Kuviossa 5 on näkyvissä edellä mainittu checkLoginStatus-funktio. Funktio tarkistaa, onko sessionStoragessa authToken-niminen token. Jos ei, niin käyttäjä kirjataan ulos. Jos se löytyy, niin se lähetetään palvelimelle tarkistettavaksi. Jos tokeni on vanhentunut tai virheellinen, käyttäjä kirjataan ulos. Jos tokeni on validi, käyttäjä jatkaa sisään kirjautuneena (ks. Kuvio 5).

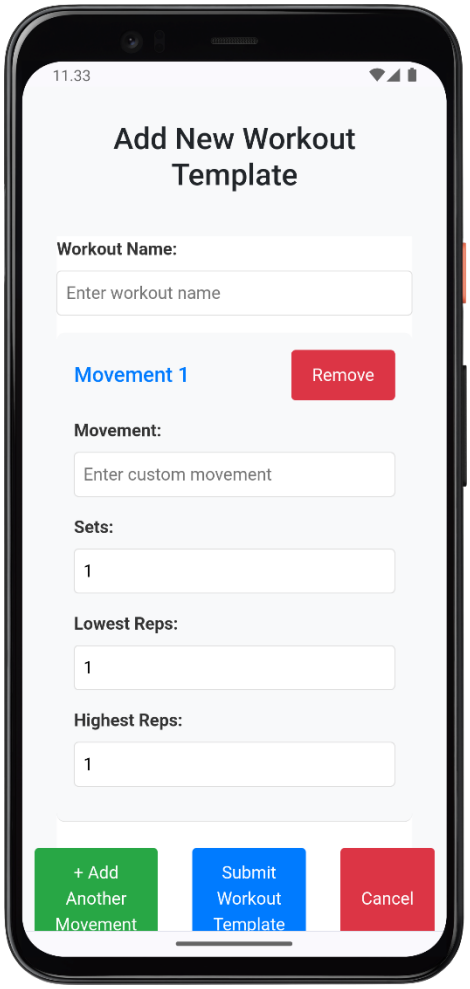
```
async checkLoginStatus() {
  const token = sessionStorage.getItem('authToken')
  const username = sessionStorage.getItem('username')
  console.log('Checking login status with token:', token)

  if (!token) {
    console.log('No token found in sessionStorage')
    this.logout()
    return { isLoggedIn: false }
  }

  try {
    const response = await axios.post(
      `http://${config.apiHost}:${config.backendPort}/user/checkToken`,
      { token }
    )
    console.log('Token validation response:', response.data)

    if (response.data.response === 'OK!' && response.data.access) {
      this.isLoggedIn = true
      this.username = username || 'User'
      return { isLoggedIn: true }
    } else {
      console.log('Token invalid, logging out')
      this.logout()
      return { isLoggedIn: false }
    }
  } catch (error) {
    console.error('Token validation failed:', error)
    this.logout()
    return { isLoggedIn: false }
  }
}
```

Kuvio 5. Käyttäjän autentikaation validointi palvelimelta

4.4 Harjoitepohja

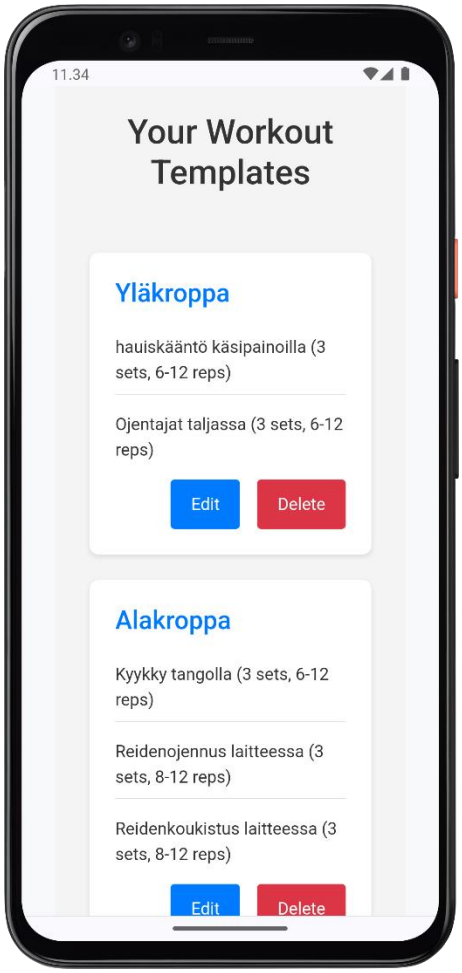
Ideana oli mahdollistaa, että käyttäjä pystyy luomaan omia harjoitepohjia. Harjoitepohja sisältää liikkeet. Liikkeet sisältävät liikkeen nimen, sarjamäärä, alhaisimman toistomäärän ja korkeimman toistomäärän. Tämä mahdollistaa nopean valinnan erilaisten harjoitteiden välillä kuntosalilla. Käyttäjä pystyy esimerkiksi luomaan ylä-, ja alakroppaharjoitteen, joten kuntosalilla ei tarvitse kuin valita tehtävä harjoite.

Kuviossa 6 on näkymä, jossa on ajateltu käyttäjän päätelaitetta eli mobiililaitetta. Napit ovat isoja ja helposti nähtävissä. Käyttäjä pystyy luomaan monta eri liikettä harjoitteeseensa. Tämän jälkeen harjoitepohja tallennetaan, jonka jälkeen pohjat näkyvät näkymässä, jossa valitaan tehtävä harjoite.



Kuvio 6. Harjoitepohjan luomisen näkymä

Harjoituspohjia pystyy muokkaamaan ja poistamaan käyttäjän tahdon mukaisesti. Harjoitepohjien muokkausnäkymä aukaisee samanlaisen näkymän kun harjoitepohjien lisäysnäkymä, mutta täyttää kaikki tiedot muokattavan pohjan tiedoilla. Tämä pitää harjoitepohjien lisäämis - ja muokkausnäkymät yhdenlaisena. Harjoitepohjien määrää ei ole rajoitettu, joten käyttäjä voi luoda niin monta harjoitepohjaa kun haluaa (ks. Kuvio 7).

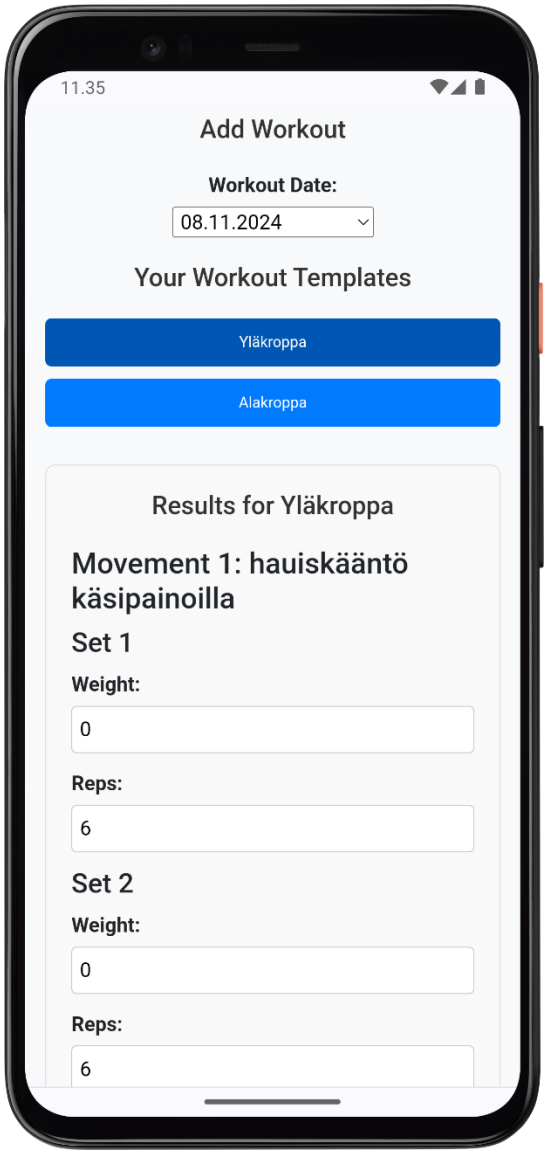


Kuvio 7. Näkymä luoduista harjoitepohjista

4.5 Harjoitteen lisääminen

Kuntosalilla pääprioriteetti on useimmilla olla tehokas nostojen suhteen, myös isona tekijänä on olla tehokas ajan kanssa. Tämän vuoksi nopeasti valittava harjoite ja nopea lukemien syöttö on tärkeää.

Näkymä pohjustaa kaavakkeen niin monella sarjalla, kun käyttäjä on määritellyt harjoitepohjissa. Jos käyttäjä ei ole lisännyt harjoitetta valitulla pohjalla, toistomääränä on käyttäjän määrittämä minimitoistomäärä. Jos käyttäjä on tehnyt harjoitteen kyseisellä harjoitepohjalla, kaavake täyttyy edellisen harjoitekerran tiedoilla. Tämä mahdollistaa helpon progressiivisen ylikuormittamisen (ks. Kuvio 8).



Kuvio 8. Harjoitteen lisäysnäkymä

4.6 Harjoitteiden seuraaminen

Harjoitteiden seuranta oli yksi tärkeimmistä ominaisuuksista palvelussa. Käyttäjät voivat tarkastella eri liikkeiden ja harjoitteiden suorituksia visuaalisten kaavioiden avulla. Kaaviot on laskettu kaavalla: paino * toistomäärä * sarjamäärä.

Testikäyttäjälle on syötetty kahden yläkroppa-harjoitteen tulokset kahdelle eri päivälle. Graafeihin piirtyy jokaisen suoritetun harjoitteen tulokset eri päiviltä. Tämä mahdollistaa helpon progressiivisen ylikuormituksen seurannan. Graafien nousujohteisuus myös toimivat motivaattoreina käyttäjille (ks. Kuvio 9).

Kuvio 9. Graafinäkymä käyttäjän tekemistä harjoitteista

4.7 Kokonaisuus

Kokonaisuudessaan sovelluksen kehittämisessä oli panostettu käyttäjäystävälliseen käyttöliittymään, joka mahdollisti sujuvan ja tehokkaan tavan tallentaa kuntosaliharjoitteita.

Sovelluksen keskeiset ominaisuudet, kuten harjoitepohjien luominen, yksinkertainen harjoitteiden tallentaminen ja seuraaminen visuaalisten kaavioiden avulla, tukevat käyttäjien tavoitteita ja auttavat heitä kuntosaliharrastuksessa. Erityisesti harjoitteiden seurantatoiminto oli suunniteltu tarjoamaan käyttäjille arvokasta tietoa heidän edistymisestään, mikä voi toimia motivoivana tekijänä käyttäjälle.

Käyttäjät voivat nopeasti valita ja syöttää harjoitetietojaan, mikä tekee treenihetkistä tehokkaita ja vähemmän ajatustavieviä.

5 Palvelinohjelmointi

5.1 Käytetyt kirjastot

Palvelinohjelmointi on olennainen osa nykypäiväistä verkkosivua. Se mahdollistaa tiedon käsittelyn ja tallentamisen tietokantaan. Palvelin myös mahdollistaa rajapinnan kaiken tiedon vastaanottamiseen ja luovuttamiseen verkkosivulle. Kuvio 10 havainnollistaa kirjastojen käyttöönottoa.

```
const express = require('express');
const logger = require('./middleware/logger');
const cors = require('cors');
const app = express();
```

Kuvio 10. Sovelluksen käyttämät palvelinpuolen kirjastot

Kuviossa 11 on näkyvissä middlewaret. Käytännössä nämä ovat funktioita, joita suoritetaan ennen varsinaisen pyynnön kohteen olevaa reittiä. Tässä tapauksessa nämä funktiot lisäävät tietoturvaa, muuttavat palvelimelle tulevat string-tyyppistä tietoa objektiksi ja kirjaavat talteen pyyntöjen tietoja.

```
// middleware
app.use(cors({ origin: '*' }));
app.use(express.json());
app.use(logger);

// logging middleware
app.use((req, res, next) => {
    console.log(`[${new Date().toISOString()}] ${req.method} ${req.url}`);
    next();
});
```

Kuvio 11. Sovelluksen käytössä olevat middlewaret

5.2 CORS

CORS on lyhenne Cross-Origin Resource Sharing. Se tarkoittaa verkkosovellusten tietoturvamekanismia, joka sallii tai estää eri lähteistä tulevat pyynnöt. CORS on tärkeä osa palvelinohjelmointia, kun API-rajapinnat ovat julkisia. Ilman CORS:sia, kuka tahansa voi yrittää saada tietoja palvelimelta, koska palvelin vastaisi ja käsittelisi kaikkien lähteiden pyynnöt. Kun palvelin on ohjelmoitu oikein, palvelin käsittelee vain aidosta lähteestä tulevat pyynnöt. Tämä on tietoturvan kannalta erittäin tärkeä osa (Cross-Origin Resource Sharing (CORS) N.d).

CORS-asetuksiin voi määrittää pyyntöjen sallitun alkuperän, sallitut HTTP-metodit, kuten GET, POST, PUT ja DELETE, sallitut pyyntöjen otsikot ja tiettyjen käyttöoikeuksien käytön, esimerkiksi evästeiden (Cross-Origin Resource Sharing (CORS) N.d).

Kuviossa 12 on näkyvissä määritetty CORS-asetukset. Erilaiset asetukset annetaan CORS:in käyttöönottovaiheessa. Tässä tapauksessa palvelinkehitys on toteutettu paikallisesti, joten ainoa määritetty asetus oli, että CORS hyväksyy kaikista osoitteista pyynnöt.

```
app.use(cors({ origin: '*' }));
```

Kuvio 12. CORS-asetukset

5.3 Reittien määrittely

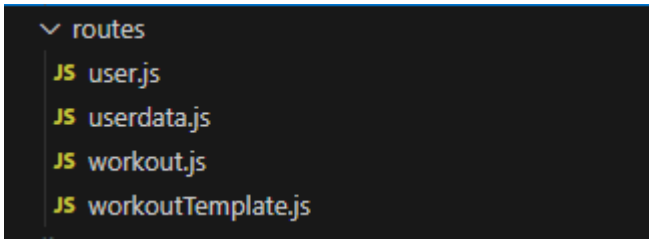
Seuraavaksi on määritelty erilaisten pyyntöjen reitit. Reitit ohjaavat miten palvelin käsittelee eri osoitteisiin tulevat pyynnöt. Paikallisessa palvelimen kehittämisessä palvelimen osoite on tietokoneen paikallisen verkon osoite ja palvelimen lähdekoodissa määritetty portti. Eli esimerkiksi http://192.168.10.10:5000.

Reitit ohjataan ensin yleisesti oikeisiin reitti-middlewareihin, jotka tässä tapauksessa olivat /userdata, /user, /workoutTemplate ja /workout. Eli pyyntö saapuu esimerkiksi http://192.168.10.10:5000/userdata (ks. Kuvio 13). Pyynnön tyyppi ja parametri vaikuttavat seuraavassa vaiheessa mihin funktioon pyyntö ohjataan.

```
// routes
const userdataRouter = require('./routes/userdata');
app.use('/userdata', userdataRouter);

const userRouter = require('./routes/user');
app.use('/user', userRouter);

const workoutTemplateRouter = require('./routes/workoutTemplate.js');
app.use('/workoutTemplate', workoutTemplateRouter);

const workoutRouter = require('./routes/workout');
app.use('/workout', workoutRouter);
```

Kuvio 13. Sovelluksen käytössä olevat reitit

Seuraava vaihe reiteissä on itse reitit. Jokaiselle eri osoitteelle on oma lähdekoodi, joka ohjaa pyynnön halutulle controller-funktiolle. Kuviossa 14 on näkyvissä gymdiary-app:in reitit.



Kuvio 14. Kaikki routes-tiedostot palvelimella

Kuviossa 15 on näkyvissä workoutTemplate.js-tiedosto, joka sisältää lähdekoodin harjoitepohja-pyyntöjen ohjaukseen. Tähän reittiin tuleva pyyntö tulee osoitteeseen http://192.168.10.10:5000/workoutTemplate/byUser ja on GET-pyyntö. GET-pyyntö ohjautuu ensin authMiddlewarelle, joka tarkistaa, että pyynnön mukana tullut token on validi. Tämän jälkeen pyyntö ohjataan halutulle funktiolle, joka tässä tapauksessa on getWorkoutsByUserId-funktio.

Sama kaava toistuu erilaisten pyyntöjen kanssa, riippuen mihin osoitteeseen pyyntö tulee.

```
// routes/workoutTemplate.js
const express = require('express');
const router = express.Router();
const { getWorkoutsByUserId, createWorkoutTemplate, deleteWorkoutTemplate, updateWorkoutTemplate } = require('../controller/workoutTemplate');

const authMiddleware = require('../middleware/authentication');

// Route to get workout templates by user ID
router.get('/byUser', authMiddleware, getWorkoutsByUserId);

// Route to create a new workout template
router.post('/create', authMiddleware, createWorkoutTemplate);

// Route to delete a workout template by ID
router.delete('/delete', authMiddleware, deleteWorkoutTemplate);

// Route to update a workout template by ID
router.put('/update', authMiddleware, updateWorkoutTemplate);

module.exports = router;
```

Kuvio 15. Harjoitepohjan reitit palvelimella

authMiddleware tarkistaa käyttäjän tokenin ennen kuin funktioita suoritetaan. Tokeniin on sisällytetty käyttäjän nimi ja id yksinkertaisempien pyyntöjen luomiseksi. Hallinta-funktioille ei siis tarvitse lähettää erikseen käyttäjän id:tä. Autentikaatio-middleware purkaa tokenin ja syöttää pyyntöön käyttäjän tiedot seuraavalle funktiolle käytettäväksi (ks. Kuvio 16). Jos token ei kuitenkaan ole validi, pyyntöön vastataan virheilmoituksella.

middleware > JS authentication.js > authentication

```
const jwt = require('jsonwebtoken');
require('dotenv').config();

const authentication = async (req, res, next) => {
    const authHeader = req.headers.authorization;
    const token = authHeader && authHeader.split(' ')[1];

    try {
        if (!token) {
            return res.status(401).json({ error: 'No token delivered' });
        }

        const decodedToken = jwt.verify(token, process.env.ACCESS_TOKEN_SECRET);

        if (decodedToken) {
            req.user = decodedToken;
            next();
        } else {
            return res.status(401).json({ error: 'Invalid token, please login' });
        }
    } catch (error) {
        console.error('Token verification error:', error);
        return res.status(500).json({ error: error.message });
    }
};

module.exports = authentication;
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS

```
[nodemon] starting `node app.js`
Server is running on port 5001
[2024-10-31T15:42:57.752Z] POST /user/checkToken
[2024-10-31T15:42:57.768Z] GET /workoutTemplate/byUser
Request User: {
  name: 'test',
  id: '66b269d973339304744b73b2',
  iat: 1730389333,
  exp: 1730403733
}
```

Kuvio 16. Autentikaatio-middlewaren logiikka

5.4 Hallinta-funktiot

Pyyntö voidaan käsitellä halutulla tavalla, kun pyyntö on saatu ohjattua oikealle funktiolle. Kuvissa 15 on näkyvissä harjoitepohja-funktiot:

1. getWorkoutsByUserId
2. createWorkoutTemplate
3. deleteWorkoutTemplate
4. updateWorkoutTemplate

Käyttäjän harjoitteiden hakufunktio getWorkoutsByUserId toimii yksinkertaisesti niin, että WorkoutTemplate-mallin kannasta etsitään kaikki, joiden tiedoista löytyy käyttäjän id. Palvelin lähettää pyynnön vastauksessa kaikki harjoitepohjat, jotka vastaavat kriteerejä. Jos kuitenkin tapahtuu jokin virhe, palvelin vastaa virheilmoituksella (ks. Kuvio 17).

```
const getWorkoutsByUserId = async (req, res) => {
    console.log('Request User:', req.user);
    try {
        const userId = req.user.id;
        const workoutsModels = await WorkoutTemplate.find({ userId });
        console.log('Workout Templates:', workoutsModels);
        res.status(200).json(workoutsModels);
    } catch (error) {
        console.error('Error fetching workouts:', error);
        res.status(500).json({ error: 'Internal Server Error' });
    }
};
```

Kuvio 17. Harjoitepohjien hakemislogiikka id-parametrilla

Harjoitepohjien luomisfunktio createWorkoutTemplate on hieman sisällökkäämpi. Funktio yrittää ensin ottaa authMiddlewaren syöttämän käyttäjän id:n pyynnöstä. Tämän jälkeen pyynnöstä otetaan harjoitepohjan nimi ja liikkeet. Funktio tarkistaa, että kaikki vaadittavat elementit löytyvät eli nimi ja liikkeet. Jos vaadittavia elementtejä ei löydy, palvelin vastaa pyyntöön virheilmoituksella. Jos kuitenkin kaikki vaadittavat elementit löytyvät, funktio luo uuden harjoitepohjan määritetyn mallin mukaan ja tallentaa sen tietokantaan, vastaten pyyntöön viestillä ja luodulla harjoitepohjalla (ks. Kuvio 18).

```
// Create new workout template
const createWorkoutTemplate = async (req, res) => {
    console.log('Request Body:', req.body);
    try {
        const userId = req.user.id; // userId from the request object set by authMiddleware
        const { name, movements } = req.body;

        // Validate input
        if (!name || !Array.isArray(movements) || movements.some(movement =>
            !movement.movement || movement.sets == null || movement.lowestReps == null || movement.highestReps == null)) {
            return res.status(400).json({ error: 'Missing required fields' });
        }

        // Create and save the workout template with all movements
        const newWorkoutTemplate = new WorkoutTemplate({
            userId,
            name,
            movements
        });

        const savedWorkoutTemplate = await newWorkoutTemplate.save();

        console.log('Saved Workout Template:', savedWorkoutTemplate);
        res.status(201).json({ message: 'Workout template created successfully', workoutTemplate: savedWorkoutTemplate });
    } catch (error) {
        // Debugging: Log error details
        console.error('Error creating workout template:', error);
        res.status(500).json({ error: 'Internal Server Error' });
    }
};
```

Kuvio 18. Harjoitepohjan lisäämislogiikka

Harjoitepohjien poistamisfunktio deleteWorkoutTemplate poistaa tietokannasta halutun harjoitepohjan. Funktio odottaa pyynnön mukana tulevaa id-parametria, joka toimii poistettavan harjoitepohjan tunnisteena. Jos id-parametri puuttuu, palvelin vastaa virheilmoituksella. Funktio hakee tietokannasta harjoitepohjamallia saadun id:n perusteella. Harjoitepohja poistetaan sen löydyttyä ja palvelin vastaa onnistumisviestillä, joka ilmoittaa harjoitepohjan poistetuksi. Jos harjoitepohjaa ei löydy, palvelin vastaa virheilmoituksella (ks. Kuvio 19).

```
const deleteWorkoutTemplate = async (req, res) => {
    try {
        const { id } = req.query; // expect id as query parameter

        if (!id) {
            return res.status(400).json({ error: 'Missing workout template ID' });
        }

        const result = await WorkoutTemplate.findByIdAndDelete(id);

        if (!result) {
            return res.status(404).json({ error: 'Workout template not found' });
        }

        res.status(200).json({ message: 'Workout template deleted successfully' });
    } catch (error) {
        console.error('Error deleting workout:', error);
        res.status(500).json({ error: 'Internal Server Error' });
    }
};
```

Kuvio 19. Harjoitepohjan poistamislogiikka id-parametrilla

Harjoitepohjan päivitysfunktio updateWorkoutTemplate vastaa olmeassa olevan harjoitepohjan päivittämisestä tietokannassa. Funktio ottaa pyynnön mukana saapuvat id, name ja movements-parametrit, jotka sisältävät päivitettävän harjoitepohjan tiedot. Ensin funktio tarkistaa, että kaikki tarvittavat tiedot ovat pyynnössä mukana. Jos jokin tieto puuttuu tai ei ole validi, palvelin vastaa virheilmoituksella. Seuraavaksi funktio etsii tietokannasta harjoitepohjaa saadun id:n perusteella. Jos harjoitepohjaa ei löydy, palvelin vastaa virheilmoituksella. Jos harjoitepohja löytyy, funktio päivittää harjoitepohjan nimen ja liikkeet saaduilla parametreillä. Harjoite pohja tallennetaan tietokantaan päivitettynä ja palvelin vastaa onnistuneella viestillä (ks. Kuvio 20).

```
const updateWorkoutTemplate = async (req, res) => {
    try {
        const { id, name, movements } = req.body;

        // Validate the required fields
        if (!id || !name || !Array.isArray(movements)) {
            console.error('Validation failed:', { id, name, movements });
            return res.status(400).json({ error: 'Missing required fields' });
        }

        // Find the existing workout template
        const existingWorkout = await WorkoutTemplate.findById(id);

        if (!existingWorkout) {
            console.error('Workout template not found with ID:', id);
            return res.status(404).json({ error: 'Workout template not found' });
        }

        // Update the existing workout template
        existingWorkout.name = name;
        existingWorkout.movements = movements;

        const updatedWorkout = await existingWorkout.save();

        res.status(200).json(updatedWorkout);
    } catch (error) {
        res.status(500).json({ error: 'Internal Server Error' });
    }
};
```

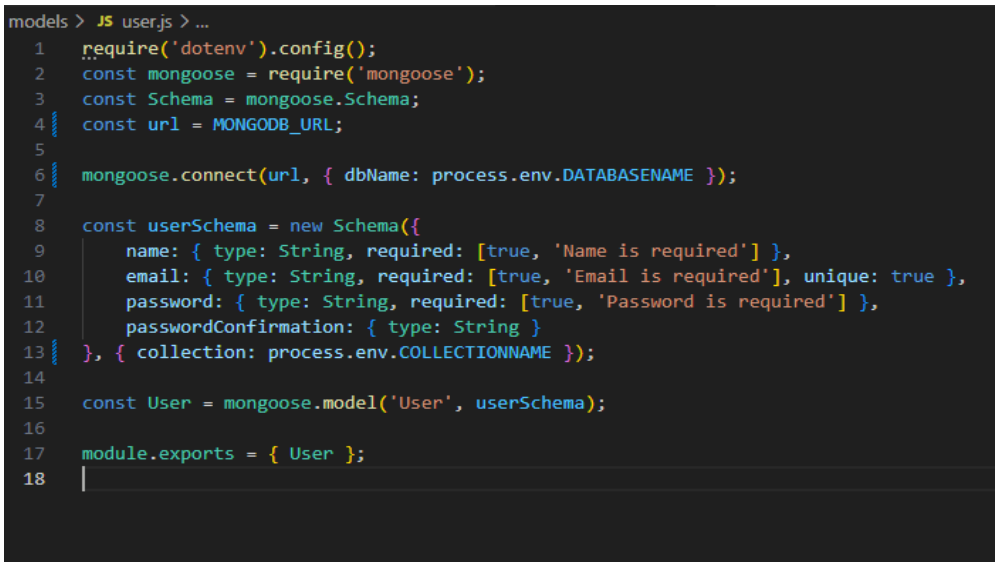
Kuvio 20. Harjoitepohjien päivitys logiikka saaduilla parametreilla

5.5 Mallit

Mallit määrittelevät, kuinka sovelluksen tiedot tallennetaan tietokantaan. Ne toimivat pohjina uusien objektien luomisessa tiettyyn rakenteeseen ja mahdollistavat tietojen validoinnin sekä tietokannan yhdenmukaisuuden. Tässä sovelluksessa mallit on määritelty käyttäen Mongoose-kirjastoa, joka on MongoDB-tietokantaan kehitetty objektimallinnustyökalu Node.js-kirjastolle. Sovelluksessa on kolme mallia: user, workoutTemplate ja workout. Jokainen malli tallentaa tiettyjä tietoja, mutta jokaisella on oma tarkoituksensa.

Kuviossa 21 on näkyvissä user-malli. Tämä malli sisältää seuraavat pakolliset kentät:

1. name – Käyttäjän nimi, joka tallennetaan tekstimuotoisena.
2. email – Käyttäjän sähköposti, joka tallennetaan tekstimuotoisena.
3. password – Käyttäjän salasana, joka tallennetaan hajautettuna tekstinä.
4. passwordConfirmation – Salasanan vahvistus, jota käytetään rekisteröitymisessä, mutta ei tallenneta erikseen.



Kuvio 21. User-malli palvelimella

Kuviossa 22 on näkyvissä workoutTemplate-malli, joka koostuu kahdesta eri mallista. Ensimmäinen on WorkoutMovementSchema, joka kuvaa yksittäistä liikettä. Se sisältää seuraavat pakolliset kentät:

1. movevent – Liikkeen nimi, joka tallennetaan tekstimuotoisena
2. sets – Sarjojen määrä, tallennetaan numeroarvona
3. lowestReps – Vähimmäistoistojen määrä liikkeelle, tallennetaan numeroarvona
4. highestReps – Enimmäistoistojen määrä liikkeelle, tallennetaan numeroarvona

WorkoutMovementSchemaa käytetään osana itse workoutTemplateSchemaa. Kun luodaan uusi workoutTemplate, se sisältää todennäköisesti useita workoutMovementSchemoja listassa. Kokonainen workoutTemplateSchema sisältää pakolliset kentät:

1. userId – Käyttäjän uniikki id, joka tallennetaan Mongoosen ObjectId-arvona
2. name – Harjoitteen nimi, joka tallennetaan tekstimuotoisena
3. movements – Harjoitteen liikkeet, jotka tallennetaan workoutMovementSchema-muotoisina objekteina listassa

```
const mongoose = require('mongoose');

const workoutMovementSchema = new mongoose.Schema({
    movement: { type: String, required: true },
    sets: { type: Number, required: true },
    lowestReps: { type: Number, required: true },
    highestReps: { type: Number, required: true }
}, { _id: false });

const workoutTemplateSchema = new mongoose.Schema({
    userId: { type: mongoose.Schema.Types.ObjectId, required: true, ref: 'User' },
    name: { type: String, required: true },
    movements: [workoutMovementSchema],
}, { timestamps: true });

module.exports = mongoose.model('WorkoutTemplate', workoutTemplateSchema);
```

Kuvio 22. WorkoutTemplate-malli palvelimella

Kuviossa 23 on näkyvissä workoutSchema. Se sisältää:

1. workoutId – harjoitteen uniikki id, joka tallennetaan Mongoose ObjectId-muodossa.
2. userId – Käyttäjän uniikki id, joka tallennetaan Mongoose ObjectId-muodossa.
3. workoutTemplateId – Harjoitepohjan uniikki id, joka tallennetaan Mongoose ObjectId-muodossa.
4. date – Harjoitteen päivämäärä, joka tallennetaan Date-muodossa.
5. movements – Harjoitelista, joka sisältää liikkeen nimen tekstimuodossa, painot listana numeromuodossa, sarjamäärän listana numeromuodossa ja toistot listana numeromuodossa.

```
const workoutSchema = new mongoose.Schema({
    workoutId: { type: mongoose.Schema.Types.ObjectId, required: false, ref: 'Workout' },
    userId: { type: mongoose.Schema.Types.ObjectId, required: true, ref: 'User' },
    workoutTemplateId: { type: mongoose.Schema.Types.ObjectId, required: true, ref:
    'WorkoutTemplate' },
    date: { type: Date, required: true },
    movements: [{
        movement: { type: String, required: true },
        weight: [{ type: Number, required: true }],
        sets: { type: Number, required: true },
        reps: [{ type: Number, required: true }]
    }]

}, { timestamps: true });

module.exports = mongoose.model('Workout', workoutSchema);
```

Kuvio 23. Workout-malli palvelimella

5.6 Käyttäjän luominen

Tietoja käsitellään käyttäjäkohtaisesti palvelussa, joten käyttäjiä pitää pystyä luomaan ja käyttämään. Tietoturvasyistä salasanoja ei säilytetä selkolukuisena tietokannassa, vaan rekisteröinnin yhteydessä salasana hajautetaan. Salasanan hajautus tarkoittaa salasanan muuttamista tietyn mittaiseksi tekstiksi, joka sisältää erilaisia merkkejä. Salasanan hajautusfunktiot ovat suunniteltu niin, että ei olisi mahdollista kääntää hajautusfunktion tuotosta enää takaisin alkuperäiseksi tekstiksi (Temoye 2024).

Kuviossa 24 on näkyvissä funktio, joka luo uuden käyttäjän palvelimella. Funktio ottaa nimen, sähköpostin, salasanan ja salasanan vahvistuksen. Näitä perustietoja tarvitaan käyttäjän luomiseen.

Ensin funktio tarkistaa, että onko saadulla sähköpostilla jo käyttäjä tietokannassa. Tämän jälkeen tulee kolme erilaista tarkistusta. Ensimmäinen tarkistetaan pyynnöstä kaikki tarvittavat tiedot. Jos jokin näistä puuttuu, palvelin vastaa virheilmoituksella. Seuraavana tarkistetaan, onko sähköposti jo tietokannassa. Jos sähköpostilla löytyy käyttäjä, palvelin vastaa virheilmoituksella. Kolmantena tarkistuksena varmistetaan, että saatu salasana ja salasanavahvistus on sama. Jos ei, niin palvelin vastaa virheilmoituksella.

Jos kaikki tarkistukset menevät läpi, lähdekoodi luo salasanan hajautuksen. Hajautuksen jälkeen luodaan käyttäjä saadulla nimellä, sähköpostilla ja hajautetulla salasanalla. Kun käyttäjä on luotu, se tallennetaan tietokantaan ja palvelin vastaa pyyntöön, että käyttäjä luotu onnistuneesti. Jos jokin kohta epäonnistuu tarkistusten jälkeen, palvelin vastaa virheilmoituksella.

```
const { User } = require('../models/user');
const bcrypt = require('bcryptjs');

const createUser = async (req, res) => {
    console.log('register request received');
    const { name, email, password, passwordConfirmation } = req.body;

    try {
        const userExists = await User.findOne({ email });
        if (!name || !email || !password || !passwordConfirmation) {
            return res.status(400).send({ success: false, msg: 'All fields are required.' });
        }
        if (userExists) {
            return res.status(400).send({ success: false, msg: 'User already exists with this email: ' + email + '.' });
        }
        if (password !== passwordConfirmation) {
            return res.status(400).send({ success: false, msg: 'Password and password confirmation do not match.' });
        }

        const saltRounds = 10;
        const passwordHash = await bcrypt.hash(password, saltRounds);

        const user = new User({
            name,
            email,
            password: passwordHash
        });

        await user.save();
        res.status(200).json({ success: true, msg: 'User created successfully.' });
    } catch (error) {
        console.error('Error creating user:', error.message);
        res.status(500).json({ success: false, msg: 'Internal server error.' });
    }
};

module.exports = { createUser };
```

Kuvio 24. createUser-funktio palvelimella

5.7 Kirjautuminen ja JSON Web Token

Kun käyttäjä kirjautuu verkkosivulle, sivusto lähettää kirjautumispyynnön palvelimelle. Pyyntö sisältää käyttäjän nimen ja salasanan. Onnistuneen kirjautumisen yhteydessä palvelin lähettää verkkosivulle JSON Web Tokenin ja käyttäjän id:n. JSON Web Tokenia käytetään käyttäjän varmentamiseen palvelun käytön aikana. JSON Web Tokenin käyttötarkoitus ei ole tiedon salaaminen, vaan

aitouden varmistaminen. Tokenit ovat allekirjoitettuja ja koodattuja, mutta eivät salattuja (Suresh 2022).

Kuviossa 25 on näkyvissä login-funktio, joka käsittelee sisäänkirjautumisen. Funktio ottaa vastraan käyttäjänimen ja salasanan. Ensimmäinen funktio tarkistaa, että käyttäjänimi ja salasana on annettu. toinen tai molemmat puuttuvat, palvelin vastaa virheilmoituksella. Tämän jälkeen funktio hakee tietokannasta käyttäjää annetulla käyttäjänimellä. Jos käyttäjää ei löydy, palvelin vastaa virheilmoituksella. Kun käyttäjä on löydetty, palvelin tarkistaa bcrypt-kirjaston avulla, vastaako annettu salasana tietokantaan tallennettua hajautettua salasanaa. Jos sanasanat eivät täsmää, palvelin vastaa virheilmoituksella.

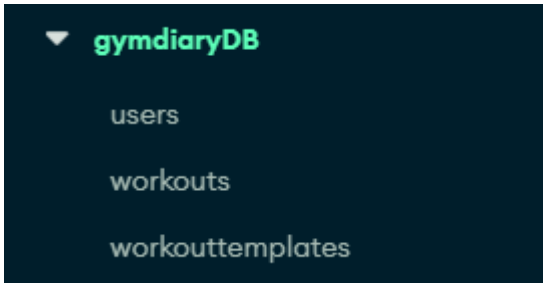
Jos kaikki tarkastukset menevät läpi, funktio luo JSON Web Tokenin käyttäjän tiedoista käyttäjän nimestä ja id:stä. Token luodaan käyttämällä palvelimen ympäristömuuttujassa määritettyä salaista kirjain- ja numeroyhdistelmää sekä tokenin vanhentumisaikaa. Tämä Token on allekirjoitettu ja koodattu, jotta sen aitous voidaan tarkistaa myöhemmin. Lopuksi palvelin vastaa onnistuneen kirjautumisen onnistuneesti ja palauttaa Tokenin ja käyttäjä id:n.

```
const jwt = require('jsonwebtoken');
const bcrypt = require('bcryptjs');
require('dotenv').config();
const { User } = require('../models/user');

const login = async (req, res) => {
    try {
        const { name, password } = req.body;

        if (!name || !password) {
            return res.status(401).json({ error: 'No name or password delivered' });
        }

        const user = await User.findOne({ name });
        if (!user) {
            return res.status(401).json({ error: 'Invalid username' });
        }

        const passwordCorrect = await bcrypt.compare(password, user.password);
        if (!passwordCorrect) {
            return res.status(401).json({ error: 'Invalid username or password' });
        }

        const userForToken = {
            name: user.name,
            id: user._id
        };

        const token = jwt.sign(userForToken, process.env.ACCESS_TOKEN_SECRET, { expiresIn: process.env.JWT_EXPIRES_IN });
        res.status(200).json({ token, user: user._id });
    } catch (error) {
        console.error('Login failed:', error.message);
        res.status(500).json({ error: 'Internal Server Error' });
    }
};

module.exports = login;
```

Kuvio 25. login-funktio palvelimella

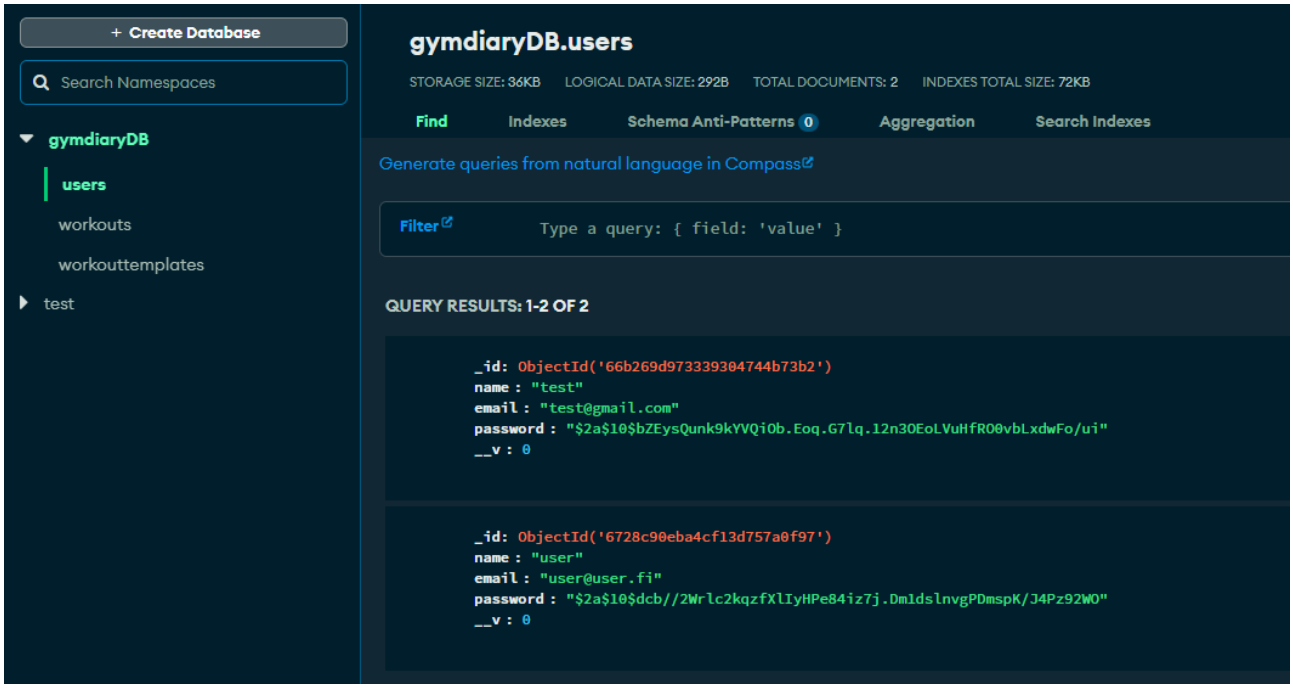
6 Tietokanta

MongoDB tarjoaa verkkosivuillaan täyden hallinnan ja tarkastelun. Kuviossa 26 on näkyvissä tietokannan rakenne, joka näyttää palvelimella määritettyjen mallien pohjalta rakennetut kokoelmat. Kokoelmat sisältävät mallein mukaisen rakenteen, joka helpottaa tiedon hallintaa tietokannassa.



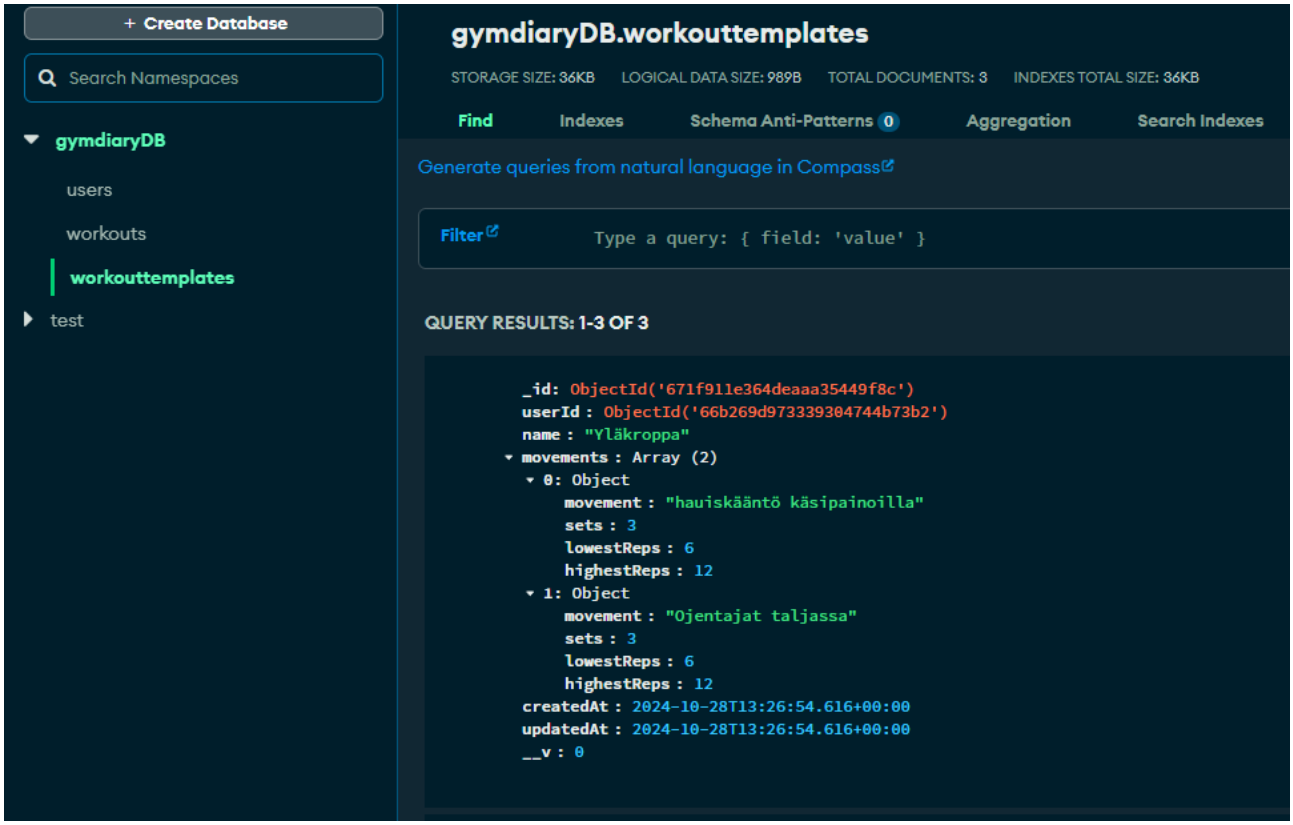
Kuvio 26. Tietokannan kokoelmat

Kuviossa 27 on näkyvissä käyttäjätiedot tietokannassa. Tiedot ovat user-mallin mukaiset ja salasana on hajautettuna createUser-funktion toimesta. Käyttäjätietoja on helpompi käsitellä _id:n avulla, koska muut tiedot saattavat muuttua. Käyttäjän id ei koskaan muutu luomisen jälkeen.



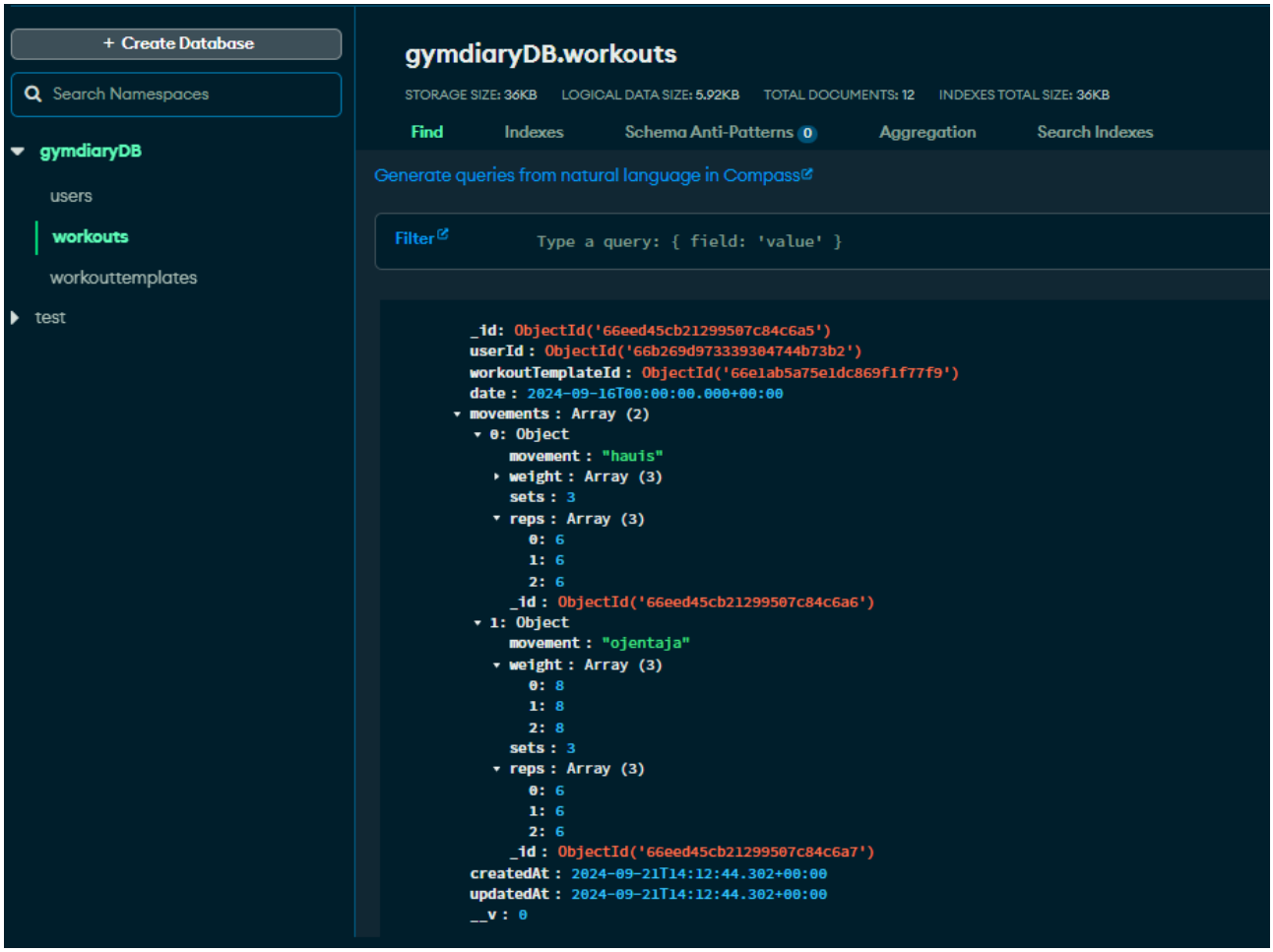
Kuvio 27. Käyttäjätiedot tietokannassa

Kuviossa 28 on näkyvissä workouttemplate-esimerkki. Esimerkistä näkee miten workouttemplate muodostuu useasta workoutMovement-malleista. Harjoitepohjan luoneen käyttäjän id myös tallentuu helpottamaan käyttäjän kaikkien harjoitepohjien hakua.



Kuvio 28. Harjoitepohjatiedot tietokannassa

Kuviossa 29 on näkyvissä esimerkki harjoitteiden tallentamisesta. Jokainen harjoite tallentuu tietokantaan harjoitemallin perusteella. Jokaiselle harjoitteelle tulee uniikki id, käyttäjän id, harjoitepohjan id, toteutuneet sarjat ja toistot painoilla, päiväys ja milloin päivitetty. Harjoitteet tallentuvat listana.



Kuvio 29. Harjoitetiedot tietokannassa

7 Sovelluksen testaus

7.1 Toteutus

Sovelluksen testaus suoritettiin manuaalisesti erilaisilla laitteilla ja selaimilla varmistaen, että sovellus toimii odotetusti kaikissa käyttötilanteissa. Mobiililaitteiden osalta testaus suoritettiin Android Studion emulaattorilla, jossa voitiin simuloida erilaisia Android-laitteita ja tarkistaa sovelluksen toimivuus mobiililaitteilla. Tämä testaus keskittyi erityisesti sovelluksen responsiivisuuteen, navigointiin ja ulkoasun sujuvuuteen mobiililaitteilla.

Tietokoneella sovelluksen toimivuutta testattiin Windows 11 käyttöjärjestelmällä käyttäen eri selaimia, kuten Microsoft Edgejä, Mozilla Firefoxia ja Google Chromea. Testaus varmisti, että sovellus toimii oikein näillä yleisillä selaimilla ja että kaikki käyttöliittymän elementit skaalautuvat oikein eri näyttökokoihin.

7.2 Tulokset

Testauksen aikana huomattiin muutamia pieniä visuaalisia virheitä, kuten graafien tekstit eivät olleet täysin näkyvissä, graafin otsikko oli diagrammien päällä ja muita pieniä visuaalisia virheitä. Nämä ongelmat kuitenkin korjattiin. Kokonaisuudessaan testausprosessi varmisti, että sovellus toimii sujuvasti ja responsiivisesti niin mobiililaitteilla kun tietokoneilla. Käyttäjäkokemus pysyi tasalaatuisen hyvänä päätelaitteesta riippumatta.

8 Pohdinta

Odotukset käytettävälle kuntosaliharjoitteiden tallentamissovellukselle olivat helppokäyttöisyys, selkeys ja toimivuus. Näihin ominaisuuksiin panostettiin palvelussa. Palvelun tärkeimmät ominaisuudet olivat harjoitteiden yksinkertainen ja nopea merkitseminen käyttäen harjoitepohjia. Nämä ydinominaisuudet saatiin toimimaan luotettavasti ja yksinkertaisesti. Palvelun ulkonäköön ja helppokäyttöisyyteen keskityttiin kehitysvaiheessa. Napit ovat isot ja selkeät, käyttöliittymä on responsiivinen ja ulkoasu yksinkertainen. Hyvänä lisänä toimii progression seurantaan rakennetut graafit D3.js-kirjastoa käyttäen.

Sovelluksen eri osat, kuten käyttöliittymä, palvelin ja tietokanta muodostavat toimivan kokonaisuuden, joka mahdollistaa käyttäjälle sujuvan ja tehokkaan tavan seurata kuntosaliharjoitteitaan. Sovelluksen käyttäjäystävällinen käyttöliittymä yhdistyy palvelimen rajapintaan, jossa tiedot käsitellään ja tallennetaan MongoDB-tietokantaan. D3.js-kirjastoa käytettiin datan visualisointiin, jolloin käyttäjä voi helposti seurata edistymistään graafien avulla. Vue.js, Node.js ja Express.js tukevat sovelluksen skaalautuvuutta ja joustavuutta, mahdollistaen laajennusten ja päivitysten kehittämisen.

Kehitysalueina on muutamien ominaisuuksien lisääminen. Yhtenä kehitysalueena olisi harjoituspohjiin jokaiseen liikkeeseen kuvaus. Kuvaus mahdollistaisi esimerkiksi vinopenkkipunneruksen kuvauksessa penkin kulman määrittämisen.

Progression toteuttaminen on tärkeä osa kuntosaliharrastusta ja yksi progressiomuoto on sarjataukojen pituus. Jos palvelussa olisi mahdollisuus käyttää sisäänrakennettua ajastinta, joka tallentaisi sarjataukojen pituuden, tämä lisäisi jälleen yhden mitattavan statistiikan.

Projektissa oli tarkoitus saada julkaistua testiympäristö valituille testihenkilöille, mutta palvelun julkaisu verkkoon vaatii vielä kehitystyötä. Palvelun runko on kuitenkin kasassa, joten kohtuullisella työmäärällä palvelu voitaisiin julkaista käyttäjien saataville verkkoon.

Lähteet

Cross-Origin Resource Sharing (CORS). N.d. Blogi julkaisu verkkosivuilla mdn web docs. Viitattu 31.10.2024. https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS.

Emadamerho-Atori, N. 2023. Angular vs. React vs. Vue.js: Comparing performance. LogRocket 31.8.2023. Viitattu 3.5.2024. https://blog.logrocket.com/angular-vs-react-vs-vue-js-comparing-performance/.

2016/679/EU. 2016. Euroopan parlamentin ja neuvoston asetus (EU) 2016/679. Viitattu 23.4.2024. https://eur-lex.europa.eu/legal-content/FI/TXT/HTML/?uri=CELEX:32016R0679.

Frequently asked questions. N.d. Virallinen vuejs verkkosivusto. Viitattu 2.5.2024. https://vuejs.org/about/faq.html.

Heath, S. N.d. The ultimate guide to building muscle mass. Everyoneactive. Viitattu 3.5.2024. https://www.everyoneactive.com/content-hub/gym/muscle-mass/.

Introduction to Node.js. N.d. Virallinen Node.js verkkosivusto. Viitattu 15.5.2024. https://nodejs.org/en/learn/getting-started/introduction-to-nodejs.

Kamau, J. 2024. Axios vs. Fetch API: Selecting the Right Tool for http Requests. Blogi julkaisu Medium verkkosivulla 22.2.2024. Viitattu 28.10.2024. https://medium.com/@johnnyJK/axios-vs-fetch-api-selecting-the-right-tool-for-http-requests-ecb14e39e285.

Kumar, S. 2022. How JWT (JSON Web Token) authentication work? Julkaisu dev.to nettisivuilla 25.4.2022. Viitattu 4.11.2024. https://dev.to/kcdchennai/how-jwt-json-web-token-authentication-works-21e7.

Mäennenä, J., Olli, J., Puputti, J., Parkkinen, J., Roisinen, T., Kuukasjärvi, K. & Haverinen, M. 2019. Voimaharjoittelu: Teoriasta Parhaisiin Käytäntöihin. Ensimmäinen painos. Lahti: VK-Kustannus Oy. Viitattu 25.4.2024. https://janet.finna.fi, Ellibslibrary.

Node.js tutorial in Visual Studio Code. 2024. Virallinen Visual studio code verkkosivusto. Päivitetty 5.2.2024. Viitattu 15.5.2024. https://code.visualstudio.com/docs/nodejs/nodejs-tutorial.

Saikrishna, R. 2019. When to Use MongoDB Rather than MySQL. Blogi julkaisu Medium verkkosivulla 21.2.2019. Viitattu 16.5.2024. https://medium.com/@rsk.saikrishna/when-to-use-mongodb-rather-than-mysql-d03ceff2e922.

Semeco, A. 2017. The Top 10 Benefits of Regular Exercise. Blogi julkaisu verkkosivulla Healthline. 10.2.2017. Päivitetty 13.11.2023. Viitattu 24.5.2024. https://www.healthline.com/nutrition/10-benefits-of-exercise#takeaway.

Swarnabha, S. 2023. What is Express.js?. Blogi julkaisu Scaler verkkosivulla 26.5.2023. Päivitetty 2.2.2024. Viitattu 15.5.2024. https://www.scaler.com/topics/expressjs-tutorial/what-is-express-js/.

Temoye, D. 2024. How to Hash Passwords with bcrypt in Node.js. Julkaisu freecodecamp nettisivuilla 3.4.2024. Viitattu 4.11.2024. https://www.freecodecamp.org/news/how-to-hash-passwords-with-bcrypt-in-nodejs/.

Tschabitscher, H. 2024. The 10 Best Workout Log Apps of 2024. Lifewire 1.1.2024. Viitattu 5.4.2024. https://www.lifewire.com/best-workout-log-apps-4140222.

Unique features of Express JS. 2023. Geeksforgeeks verkkosivun julkaisu sivulla geeksforgeeks. Päivitetty 23.11.2023. Viitattu 15.5.2024. https://www.geeksforgeeks.org/unique-features-of-express-js/.

Wennman, H., Borodulin, K & Jousilahti, P. 2019. Vapaa-ajan liikunta ja fyysinen aktiivisuus lisääntyvät Suomessa WHO:n tavoitteen mukaisesti. Tutkimus tiiviisti. 30/2019. Helsinki: Terveyden ja hyvinvoinnin laitos. Viitattu 2.5.2024. https://www.julkari.fi/bitstream/handle/10024/138483/TUTI2019_30_Liikunta%20vapaa-ajalla%20WHO_tark_.pdf?sequence=1&isAllowed=y.

What is D3?. N.d. Virallinen d3.js verkkosivusto. Viitattu 3.5.2024. https://d3js.org/what-is-d3.

What is SQL?. 2024. Geeksforgeeks verkkosivun julkaisu sivulla geeksforgeeks. Päivitetty 8.5.2024. Viitattu 16.5.2024. https://www.geeksforgeeks.org/what-is-sql/.

Why use MongoDB and When to Use It?. N.d. MongoDB virallinen verkkosivu. Viitattu 16.5.2024. https://www.mongodb.com/resources/products/fundamentals/why-use-mongodb.