Tampereen ammattikorkeakoulu

GraphQL:n avulla rajapinnan toteuttaminen verkkosovellukseen

Tuomo Jouppi

OPINNÄYTETYÖ
Joulukuu 2022

Tieto- ja viestintätekniikan tutkinto-ohjelma
Ohjelmistotekniikka

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tieto- ja viestintätekniikan tutkinto-ohjelma
Ohjelmistotekniikka

JOUPPI, TUOMO:
GraphQL:n avulla rajapinnan toteuttaminen verkkosovellukseen

Opinnäytetyö 39 sivua
Joulukuu 2022

Opinnäytetyössä rakennettiin moderneilla teknologioilla Projektori-nimisen verkkosovelluksen backend-puolta. Backend-puoli käsitti työssä tietokannan ja rajapinnan. Verkkosovelluksen rakentamisessa käytettyihin teknologioihin perehdyttiin työssä tarkasti, koska ne sisälsivät moderneja, vaihtoehtoisia paradigmoja backend-puolen kehittämiseen. Keskeisimpänä esiteltynä ja käytettynä teknologiana työssä oli GraphQL, johon nojaten toteutettiin rajapinta.

Projektori on Tampereen ammattikorkeakoulun tietotekniikan koulutusohjelman käyttötarpeisiin tuleva verkkosovellus, jolla helpotetaan sitä, että projekteille saataisiin opiskelijoista tekijöitä. Sovelluksen suunnitteluun ja kehittämiseen saatiin suuntaviivoja ammattikorkeakoulujen avoimelta TKI-toimintahankkeelta ja TAMKin tietotekniikan koulutusohjelmalta.

Projektori-verkkosovelluksen backend-puolelle valmistui työssä toimintakuntoinen sovellus. GraphQL osoittautui hyväksi ja joustavaksi teknologiaksi, jolla toteuttaa rajapinta verkkosovellukseen. Projektorin rajapintaan valmistui merkittävä osa toivotuista, perustavanlaatuisista ominaisuuksista. Monet nyt tehdyt ominaisuudet vaativat vielä hienosäätöä ennen kuin sovellus voidaan julkaista ensimmäisten sovelluskäyttäjien saataville. Lisäksi sovelluksen testaamiseen tulisi kiinnittää huomiota jatkokehityksessä, koska tämän työn puitteissa siihen ei ehditty panostaa.

Asiasanat: GraphQL, backend, verkkosovellus, Node.js

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in ICT Engineering
Software Engineering

JOUPPI, TUOMO:
Implementing an API for a web application with GraphQL

Bachelor's thesis 39 pages
December 2022

The objective of this thesis was to develop a backend application for the Projektori web application. The backend application was built using modern technologies, which included alternative paradigms for a backend development. Therefore, these technologies were also thoroughly introduced in the thesis. The most important technology described and used was GraphQL, which was used to implement the application's API.

The Projektori was developed for the degree programme in ICT engineering at Tampere University of Applied Sciences. The Projektori web application will be used as a place to manage projects for students. Preliminary guidelines for the development and design of the Projektori web application were received from the degree programme in ICT engineering and open RDI and innovation ecosystem project.

As a result of the thesis, an appropriate backend application for Projektori was developed. GraphQL proved to be a flexible and developer-friendly technology to implement API. A significant part of the desired fundamental features was developed to the Projektori's API. Further study and development are needed for tweaking features and testing the application.

Key words: GraphQL, backend, web application, Node.js

SISÄLLYS

1 JOHDANTO ... 6
2 TYÖN TAUSTA ... 7
3 VERKKOSOVELLUKSEN TEKNOLOGINEN RAKENNE ... 8
3.1 Full stack -verkkosovellus ... 8
3.2 Teknologiapino ... 9
3.3 Kontitus ... 10
4 TEKNOLOGIAT ... 12
4.1 Apollo ... 12
4.1.1 Apollo-alusta ... 12
4.1.2 GraphQL ... 13
4.1.3 Apollo Server -palvelinsovellus ... 14
4.2 PostgreSQL ... 15
4.3 Prisma ... 15
4.4 Typescript ... 17
4.5 Node.js ... 18
4.5.1 Arkkitehtuuri ... 18
4.5.2 NPM ja YARN ... 19
5 PROJEKTORIN ESITTELY ... 21
5.1 Sovelluksen suunnittelu ... 21
5.2 Sovellukseen toteutettu toiminnallisuus ... 21
6 PROJEKTORIN BACKEND-PUOLEN SOVELLUSKEHITYSTYÖ ... 25
6.1 Tietokanta ... 25
6.2 Apollo Server -palvelinsovelluksen konfigurointi ... 29
6.3 GraphQL-palvelin ... 30
6.3.1 Skeema ... 30
6.3.2 Ratkaisijat ... 32
6.3.3 Rajapintaoperaatiot ... 34
7 POHDINTA ... 37
LÄHTEET ... 38

LYHENTEET JA TERMIT

API	*Application programming interface* Ohjelmointirajapinta
CRUD	*Create, Read, Update, Delete* Tiedon luominen, lukeminen päivittäminen ja poistaminen, mitkä ovat neljä tärkeintä tallennettun tiedon operaatiota.
enumeraatio	Tietotyyppi, jonka kaikki sallitut arvot ovat tiedossa sitä määriteltäessä
HTTP	*Hypertext Transfer Protocol* Protokolla, jota laitteet ja ohjelmistot voivat käyttää tiedonsiirtoon.
JSON	*JavaScript Object Notation* JavaScript-ohjelmointikielistandardiin perustuva tiedonsiirtokieli.
middleware	Väliohjelma. Esimerkiksi ohjelmakomponentti, joka toimii ohjelmistokerrosten välissä palveluna.
REST	*Representational State Transfer* Arkkitehtuurillinen tapa standardien mukaiseen tiedonsiirtoon verkossa olevien järjestelmien välillä.
skeema	Määrämuotoinen tapa esittää tiedon jäsentymistä ja mahdollista sisältöä.
SQL	*Structured Query Language* Kyselykieli, jolla tehdään useimpiin relaatiotietokantoihin haut ja muutokset.
TAMK	Tampereen ammattikorkeakoulu
TCP-portti	*Transmission Control Protocol* -portti Protokollan tarvitsema palvelupiste tietoliikenneyhteyksien erotteluun ja tunnistamiseen.
TKI	Tutkimus-, kehittämis- ja innovaatiotoiminta

1 JOHDANTO

Opinnäytetyössä toteutetaan moderneilla teknologioilla backend-puolta Projektori-nimiseen full stack -verkkosovellukseen. Projektorilla pyritään helpottamaan sitä, että projektit löytäisivät opiskelijoista tekijöitä. Projektoria kehitetään alkuun Tampereen ammattikorkeakoulun tietotekniikan koulutusohjelman käyttötarpeisiin, mutta myöhemmin Projektori tuodaan Tampereen ammattikorkeakoulun muiden koulutusohjelmien saataville.

Projektorin suunnitteluun ja kehittämiseen saatiin suuntaviivat tietotekniikan koulutusohjelmalta ja AMKien avoimelta TKI-toimintahankkeelta. AMKien avoimelta TKI-toimintahankkeelta saatiin lisäksi rahoitusta sovelluksen kehittämiseen. Projektoria kehitetään pääosin opiskelijavetoisesti.

Teknologiat, jotka tässä työssä esitellään, ovat osaksi vaihtoehtoisia paradigmoja sisältäviä teknologioita sille, miten backend-puoli perinteisemmin rakennetaan verkkosovellukseen. Näihin teknologioihin perehdytään tässä työssä. Keskeisimpänä teknologiana työssä on GraphQL:n avulla toteutettu rajapinta. Työssä käydään lisäksi pääpiirteittäin läpi, millaisista palasista nykypäivän full stack -verkkosovellus voi rakentua.

Luvussa 2 käydään läpi työn taustaa. Luvut 3 ja 4 käsittelevät sovelluksen teknologista rakennetta ja käytettäviä teknologioita. Luku 5 esittelee lyhyesti Projektorin toiminnot, jotka toteutettiin. Toimintojen esittely helpottaa ymmärtämään lukua 6, jossa käydään läpi Projektorin backend-puolen kehitystyötä ja siellä käsiteltävää GraphQL:n avulla toteutettua rajapintaa.

2 TYÖN TAUSTA

TAMKin tietotekniikan koulutusohjelmassa on ollut vuosia haasteena yhdistää eri kontaktien kautta löytyvät työmahdollisuudet ja työmahdollisuuksia hakevat opiskelijat. Nämä työmahdollisuudet ovat realisoituneet opiskelijoille esimerkiksi opinnäytetyö- tai työharjoittelupaikkoina, projektiopintoina suoraan yritykseen tai AMKin kautta, tai työtehtävinä erilaisissa hankkeissa, joissa TAMK on osapuolena.

Työmahdollisuudet tulevat useasti nopealla aikataululla, joka on aiheuttanut aikataulupaineita tavoittaa opiskelijoista sopivat tekijät ajoissa. Luonnollinen opiskelijoiden kierto opintojen alusta valmistumiseen myös osaltaan johtaa haasteisiin tiedon välityksen osalta. Ratkaisuksi oli mietitty alustaa, jossa opetushenkilökunta voisi kätevästi ilmoittaa työmahdollisuuksista. Tällaiselta alustalta opiskelijat voisivat helposti silmäillä heitä itseään kiinnostavia työmahdollisuuksia.

Alustaidea konkretisoitui lopulta tietotekniikan koulutusohjelman henkilöstön ohjauksessa verkkosovellukseksi TAMKin järjestämän kesäharjoittelujakson aikana kahden opiskelijan työpanoksella itseni mukaan lukien. Verkkosovelluksen työnimeksi tuli Projektori. Tähän asti tehdystä sovelluskehitystyöstä yhtenä dokumentaatiotuotoksena on tämä opinnäytetyö, joka keskittyy Projektori-verkkosovelluksen backend-puolen kehittämisosuuteen.

3 VERKKOSOVELLUKSEN TEKNOLOGINEN RAKENNE

3.1 Full stack -verkkosovellus

Arkkitehtuurisesti Projektori on verkkosovelluskokonaisuus, josta voidaan erotella suurempina teknologisina sovelluskerroksina asiakassovellus (frontend) ja palvelinpuolen sovellus (backend). Projektorin frontend-puoli on verkkoselaimissa, niin tietokoneella kuin mobiililla, sovelluskäyttäjillä sovelluksen enimmäkseen näkyvä osuus. Backend-puoli puolestaan käsittää enimmäkseen sovelluksen taustalla tapahtuvaa toiminnallisuutta, rajapinnan ja tietokannan. Frontend- ja backend-puoli ovat toisiinsa liitoksissa ja yhdessä muodostavat full stack -verkkosovelluksen.

Full stack -termille ei ole yksiselitteistä määritelmää, koska eri konteksteissa sillä on eri merkityksiä. Yleisesti ottaen full stack -termillä kuitenkin tarkoitetaan sovelluksen rakenteellista pinomaisuutta eli kerroksisuutta. Tässä opinnäytetyössä lähestymistapa on lähinnä teknisestä näkökulmasta sovelluksen kerroksisuus, jollei muuten mainita.

Full stack -sovelluksen frontend on tyypillisimmillään sitä, minkä sovelluskäyttäjä voi nähdä ja johon hän voi olla vuorovaikutuksessa. Verkkosovelluksissa se on internetselaimessa näkyvä käyttöliittymä. Teknisesti internetselain tulkitsee esimerkiksi HTML:n, CSS:n, Javascriptin ja mediatiedostot selainmoottorilla käyttäjälle näkyviksi visuaalisiksi elementeiksi. (The Software Guild 2015.)

Frontend-puoli toimii asiakassovelluksena palvelinpuolen sovellukselle eli backend-puolelle. Backend-puoli tarjoaa resursseja ja toimintoja frontend-puolelle. Backend-puolella voi olla frontend-tyyppisten asiakassovellusten lisäksi myös backend-tyyppisiä asiakassovelluksia. (The Software Guild 2015.)

Backend-puolella tarkoitetaan tyypillisesti palvelinta ja siellä tapahtuvaa sovellustoimintaa. Myös tietokanta usein lasketaan kuuluvan osaksi backend-puolta. Backend-puolella tapahtuu kaikki se toiminta, mikä ei ole välttämättä sovelluskäyttäjän visuaalisesti havainnoitavissa. Backend-puoli voi olla esimerkiksi

vastuussa tietokannasta ja rajapinnasta asiakassovelluksille päin. (The Software Guild 2015.)

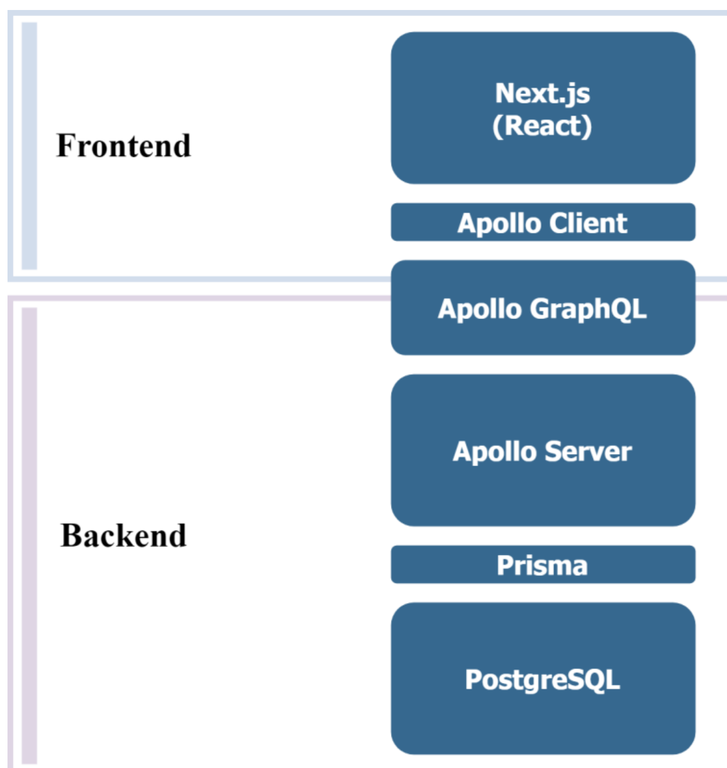
Raskaat laskentatehoa vaativat toimenpiteet usein suoritetaan backend-puolella, jos suinkin mahdollista. Verkkosovellusten backend-puolen merkitys koko sovelluksen toiminnallisuuden toteuttamisessa yleensä korostuu, mikäli tarvitaan monimutkaisempaa tai raskaampaa taustalla tapahtuvaa toiminnallisuutta tai tiedon säilömistä. (The Software Guild 2015.)

3.2 Teknologiapino

Projektori-verkkosovellus haluttiin toteuttaa moderneilla sovellustekniikoilla ja -teknologioilla. Nämä vaatimukset otettiin huomioon sovelluksen suunnittelutyön aikana. Tärkeä kriteeri teknologiavalinnoille oli myös se, että teknologioilla olisi vahva tuki tulevaisuudessa, koska sovelluksen elinkaaresta haluttiin mahdollisimman pitkä. Sovelluksen elinkaarena pidetään sitä aikaa, joka kuluu sovelluksen kehitystyön aloituksesta sen käytöstä poistoon (Haikala & Märijärvi 2004, 36). Projektorin sovelluskehitystyöstä saattaa tulla pitkäkestoista, koska siihen ei ole kiinnitettynä jatkuvasti kokopäiväisiä tekijöitä.

Projektorille lopulta päätettyä teknologista rakennetta voidaan tarkastella full stack -verkkosovelluksille tyypillisen pinomaisen rakenteen kautta, koska pinon eri kerroksia voidaan käsitellä omina kokonaisuuden muodostavina sovelluskerroksina. Pinolla viitataan ohjelmointikieliin, teknologioihin ja työkaluihin, mitkä muodostavat digitaalisen lopputuotteen (The Software Guild 2015). Pinon sovelluskerroksia lähemmin tarkastelemalla nähdään, mistä tarkalleen ottaen digitaalinen lopputuote muodostuu.

Kuviossa 1 on Projektori-verkkosovelluksen pino. Pinossa päällimmäisenä on teknologioita, joita käytetään sovelluksen frontend-puolella. Puolestaan näiden alapuolella on teknologioita, joita käytetään sovelluksen backend-puolella. Näistä backend-puolella käytetyistä teknologioista kerrotaan tarkemmin tässä opinnäytetyössä.



KUVIO 1. Projektori-verkkosovelluksen pino

3.3 Kontitus

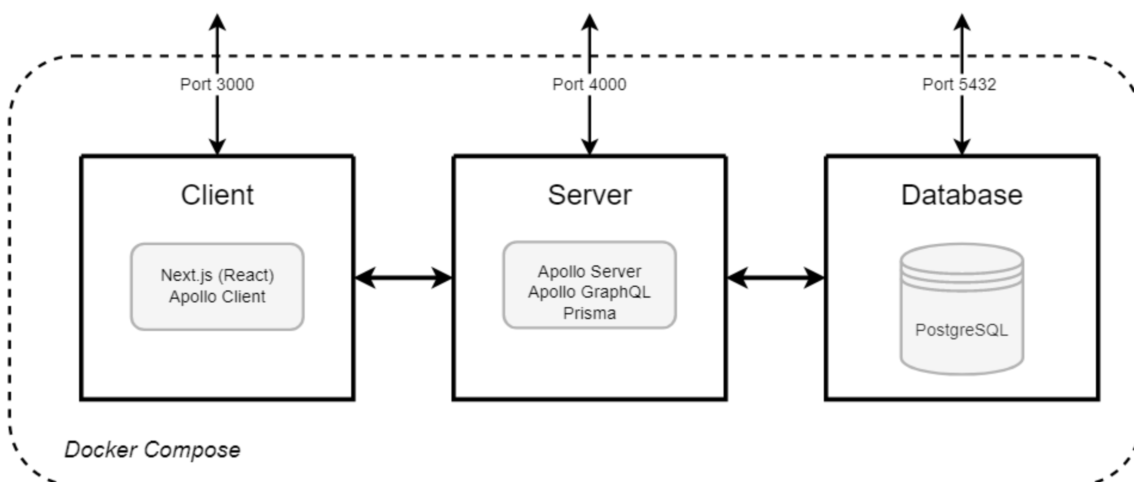
Backend- ja frontend-puoli ovat Projektorissa sovelluskerroksia, joita pystyttään kehittämään omina sovelluskokonaisuuksinaan. Vaikka sovelluskerrosten sovelluskehitystyö tapahtuisi välillä erillään, on ne tarkoitus lopulta liittää toisiinsa. Tällaista koko sovelluksen pilkkomista pienemmiksi sovelluskokonaisuuksiksi kutsutaan nykypäivän sovelluskehityksessä kontittamiseksi (Kotilainen 2020).

Kontittaminen on tapa, jolla sovelluskokonaisuuksia voidaan paketoida omiksi sovellusyksiköikseen. Kontti (container) toimii sovellusyksikölle eristettynä ajoympäristönä, johon ympärillä olevan järjestelmän vaikutukset on minimoitu. Tämä tuo lisävarmuutta siihen, että sovellusyksikkö toimii sekä kehitys- että tuotantoympäristöissä suunnitellulla tavalla. (Kotilainen 2020.)

Docker on yksi suosituimmista sovellusalustoista konttien hallintaan niin Windows kuin Linux-järjestelmissä. Docker tarjoaa monipuolisesti työkaluja ja keinoja erilaisiin konttien hallinnan tarpeisiin. Esimerkiksi Docker Compose -työkalulla

mahdollistetaan useammasta kontista muodostuvien sovellusten suorittaminen omissa ajoympäristöissään samanaikaisesti. (Docker 2022.)

Kuviossa 2 on havainnollistettu Projektorin konttien orkestrointia Docker Compose -työkalun avulla. Client-kontti vastaa frontend-sovelluskerroksen toiminnasta ja puolestaan Server- ja Database-kontit yhdessä backend-sovelluskerroksesta. Kaikki kontit ovat Dockerin kytkeminä toisiinsa siten, että ne voivat keskustella toistensa kanssa. Lisäksi jokaiselle kontille on määritetty kehitysympäristössä ulkoinen TCP-portti, jonka kautta konttiin tai kontin ohjelmasuoritteeseen päästään käsiksi.



KUVIO 2. Konttien orkestrointi Projektori-verkkosovelluksessa

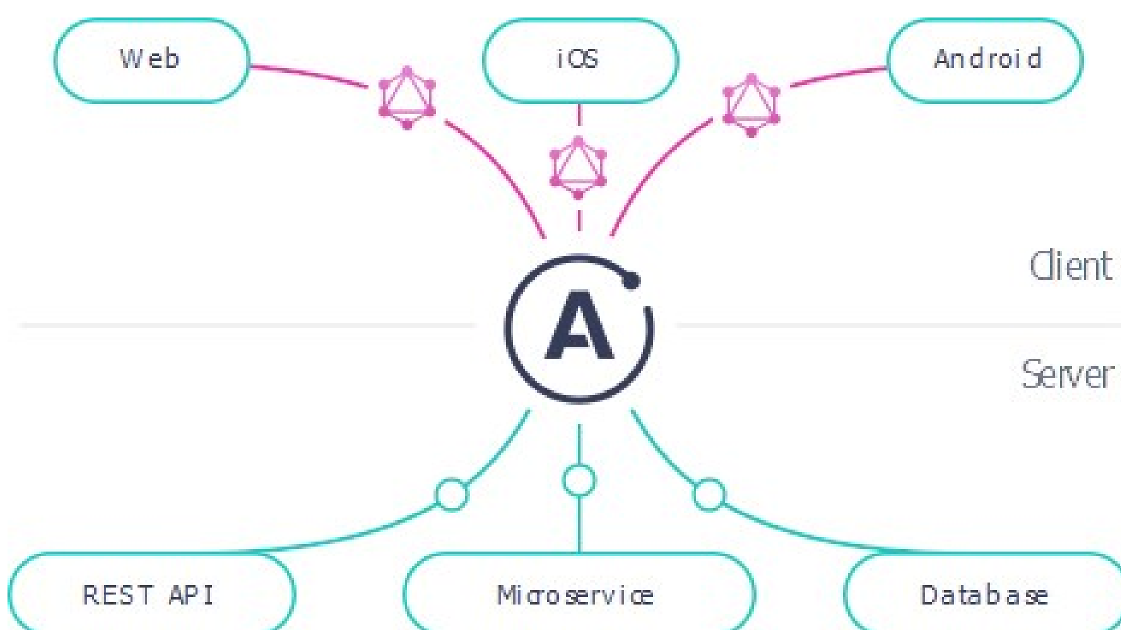
4 TEKNOLOGIAT

4.1 Apollo

4.1.1 Apollo-alusta

Apollo on kokonaisvaltainen alusta tiedonhallintaan sovelluksissa. Sillä helpotetaan rakentamaan, hallitsemaan ja skaalaamaan yhtenäisemmällä tavalla sovelluksessa liikkuvia datavirtoja. Apollon hallitsemat datavirrat voivat olla vuorovaikutuksessa minkä tahansa yhdistetyn tietovaraston kanssa ja liitettynä voi olla myös ulkoisia rajapintoja. Asiakassovelluksille päin datavirrat määritellään ja toteutetaan GraphQL:n tarjoamilla rakenteilla. (The Apollo Graph Platform n.d.)

Apollo-alustasta erotellaan palvelinsovellus (Apollo Server) ja asiakassovellus (Apollo Client) omiksi teknisiksi kokonaisuuksiksi. Kuviossa 3 havainnollistetaan Apollo-alustan hallitsemia datavirtoja näiden teknisten kokonaisuuksien välillä. Näiden kokonaisuuksien välissä toimii Apollon GraphQL-palvelin toteutus. (The Apollo Graph Platform n.d.)



KUVIO 3. Apollo-alustan hallitsemat datavirrat palvelin- ja asiakassovelluksen välillä (The Apollo Graph Platform n.d.)

4.1.2 GraphQL

GraphQL on Facebookin kehittämä datan esitysstandardi rajapintaan. GraphQL tarjoaa standardoidun kielen objektityyppien, datatyyppien ja rajapintaoperaatioiden määrittelemiseen. Standardipohjaisuutensa takia sitä voidaan käyttää useiden eri ohjelmistoalustojen ja -kehyksien välillä sujuvasti. Johdonmukainen ja luotettava datan esitystapa, niin palvelin- että asiakassovelluksessa, on tehnyt GraphQL:stä varteenotettavan vaihtoehdon rajapinnan toteutustapana. (Choi 2020, luku 9.)

GraphQL-rajapintaa toteutettaessa käytettään sen sisältämää tyypitysjärjestelmää datan tyypityksille. Rajapinnan objektityypit kenttien datatyyppeineen tai objektityyppeineen määritellään GraphQL-palvelimella olevassa GraphQL-skeemassa, joka toteutetaan GraphQL:n skeemakielellä (Schema Definition Language). Kyseistä skeemaa käytetään lisäksi mahdollisten CRUD-operaatioiden esittelyyn kuin myös rajapintaan kohdistuvien GraphQL-rajapintakyselyiden validointiin. (Choi 2020, luku 9.)

GraphQL:n etuina on joustavuus ja sen tarjoama kehittäjäkokemus. GraphQL-rajapintaan on sovelluskehittäjän helppo lisätä uusia kenttiä datatyyppeineen, joita asiakassovelluksesta voidaan sopivan hetken tullen alkaa hyödyntämään. Tämä sujuvoittaa rajapinnan kehittämistä ja ominaisuuksien nopeampaa käyttöönottoa. Periaatteena GraphQL:ssä on, että asiakassovellus saa päättää, mitä kenttiä GraphQL-rajapinnasta palautetaan. Perinteisemmissä rajapinnoissa, kuten esimerkiksi REST API:ssa, tämä ei ole ollut mahdollista, koska rajapinnan palauttamiin kenttiin asiakassovelluksessa ei enää ole voitu vaikuttaa. (Why adopt GraphQL? n.d.)

REST API:iin verrattuna GraphQL:llä voidaan vähentää kaistanleveyden käyttöä ja sillä tavoin laskea palvelinkustannuksia, koska GraphQL-rajapintakyselykielellä (query language) voidaan toteuttaa täsmällisiä rajapintakyselyitä. GraphQL-rajapinnat taipuvat hyvin uudelleen käytettäviksi eri käyttötarkoituksiin, kun GraphQL-rajapintakyselykielistä rajapintakyselyä vain muutetaan. (Choi 2020, luku 9.)

GraphQL-palvelimelle on ainoastaan yksi päätepiste (endpoint). GraphQL-rajapintakyselykielellä toimitaan tämän yhden päätepisteen kautta. Sen kautta hoidetaan kaikki rajapintaoperaatiot. Päätepistettä käyttävät asiakassovellukset rajapintapäätepisteenä ja muut datapalvelut päätepisteenä GraphQL-palvelimelle. (Choi 2020, luku 9.)

Apollo GraphQL on yksi monista GraphQL-spesifikaation toteuttavista GraphQL-palvelimista. Apollo alustan keskeisenä valttina väitetään olevan se, että toimintojen keskiössä on GraphQL ja sen rajapintatoteutus ilman muita rajapintatoteutustapoja. Apollo Server ja Apollo Client tuovat mukanaan monia käteviä työkaluja monimutkaisten GraphQL-toimintojen toteuttamiseksi. Nämä työkalut eivät ole saatavilla monissa muissa GraphQL-toteutuksista. Tällaisia ovat muun muassa sivutuksen, välimuistin ja kirjautumisjärjestelmien hallinta. (The Apollo Graph Platform n.d.)

4.1.3 Apollo Server -palvelinsovellus

Apollo Server eli Apollo alustan palvelinsovelluksen sovelluskehys keskittyy datavirtojen hallitsemiseen palvelinpuolella. Se luo raamit kommunikointiin eri datalähteiden kanssa, kun dataa käsitellään. Sen yhtenä tärkeimmistä tehtävistä on palvella Apollon GraphQL-palvelinta ja käsitellä asiakassovellusten GraphQL-operaatiot. (The Apollo Graph Platform. n.d.)

Apollo Server voidaan käyttöönottaa useimmissa Node.js pohjaisissa kehitysympäristöissä joko suoraan tai middlewarena toisessa Node.js-sovelluskehyksessä. Apollo Server on suunniteltu toimimaan yhteen Typescript-ohjelmointikielen kanssa, josta Apollo hyödyntää sen tyypitysjärjestelmän tyyppimäärittelyjä. (Introduction to Apollo Server n.d.)

Apollo Serverin rakenteesta voidaan erotella sen toiminnan kannalta merkittävimpinä kokonaisuuksina GraphQL-skeema (schema), GraphQL-ratkaisijat (resolvers) ja GraphQL-palvelin. Skeemassa määritellään kaikki GraphQL-rajapinnassa käytössä olevat kentät tyyppeineen (fields and types), kyselyoperaatiot, joita rajapintaan voidaan tehdä (query types) ja muutoksia aiheuttavia operaatiot,

joita rajapintaan voidaan tehdä (mutation types). Puolestaan ratkaisijoissa ovat ne keinot, kuinka operaatiot suoritetaan ja kuinka skeemassa määritetyt kentät täytetään käytössä olevien datalähteiden tiedoilla. Viimeisekseen on GraphQL-palvelin, jota edellä mainitut kokonaisuudet palvelevat. (Schema basics n.d.)

4.2 PostgreSQL

PostgreSQL on avoimen lähdekoodin relaatiotietokantajärjestelmä, jota on aktiivisesti kehitetty jo yli 30 vuotta. Se on yksi varteenotettava vaihtoehto monille tietokantajärjestelmille, koska se on saavuttanut hyvän maineen luotettavana ja tehokkaana tietojen hallinnoimisen järjestelmänä. (PostgreSQL 2022.)

PostgreSQL tukee pääosin SQL-standardin mukaisia käytänteitä, jos ne eivät ole ristiriidassa PostgreSQL:n SQL-kieltä laajentavien ominaisuuksien kanssa. Laajentavat ominaisuudet keskittyvät parantamaan PostgreSQL:n käyttömahdollisuuksia mitä erilaisempiin käyttökohteisiin tiedon varastoimisen järjestelmänä. Tietojen eheys, järjestelmän vikasietoisuus ja kyky hallita tietoa, oli se sitten määrältään suurta tai pientä, ovat PostgreSQL:n valtteja nähden muihin tietokantajärjestelmiin. (PostgreSQL 2022.)

4.3 Prisma

Prisma on eräänlainen ORM (object-relational mapping) -työkalu eri tietokantajärjestelmien ja eri ohjelmointikirjastojen välille. Prisman avulla tietokannan ja Node.js-pohjaisen sovelluksen välille generoidaan tyyppiturvallinen kommunikointirajapinta. Tietokantatoimenpiteet pystytään suorittamaan tämän kommunikointirajapinnan kautta samalla ohjelmointikielellä kuin sovellus muutenkin on toteutettu. (What is Prisma? n.d.)

ORM on työkalu tai ohjelmointitekniikka, millä muunnetaan yhteensopimattomien järjestelmien välinen yhteys ja kommunikointitapa toimimaan olio-orientoituneella (object-oriented) ohjelmointikielellä toteuttaen. Tyypillisimmillään tarkoitus on luoda muunnos relaatio- ja olioesityksien välille. Sovelluskehittäjä pystyy tämän

muunnoksen ansiosta toimimaan relaatiotietokantoihin päin olio-ohjelmointikielillä, joita mahdollisesti jo käytetään muun sovelluksen toteutukseen. (Paterson 2020.)

Prismalla ja muilla vastaavilla ORM-työkaluilla tehdään sovelluskehittäjistä tuotteliaampia tietokantojen parissa työskentelyssä, kun toimitaan tutuilla olio-ohjelmointikielillä (Why Prisma? n.d.). Perinteisesti sovelluskehittäjältä on vaadittu SQL-kielen osaamista sovellusprojektissa muuten käytettävien olio-ohjelmointikielien lisäksi, kun on siirrytty kehittämään sovelluksessa mukana olevaa relaatiotietokantaa. Relaatiotietokantojen parissa työskentely on saattanut olla sovelluskehitystä hidastava tekijä, kun SQL-kielisten tietokantakyselyiden toteuttaminen ja virheenkorjaus on saattanut viedä paljon sovelluskehittäjän työaikaa muun sovelluksen kehittämiseltä. Tähän ongelmaan Prisma on tuonut oman ORM-työkalumaisen ratkaisunsa.

Merkittävimpänä komponenttina Prismassa on Prisma Schema, jonka täytyy sisältää kokoelma malleja, jotka edustavat relaatiotietokannan tietokantatauluja (Prisma schema n.d.). Prisma Schemasta käytettään myös pelkkää nimeä skeema. Siihen pohjaavat muut Prisman komponentit toiminnallisuutensa.

Skeeman pohjalta Prisma Migrate -komponentti rakentaa tietokannan tietokantamallin skeeman mukaiseksi säilyttäen myös mahdollisesti jo olemassa olevat tiedot tietokannassa (Prisma Migrate n.d.). Prisma Client -komponentilla lopulta generoidaan tietokannan tietokantamalliin ja Prisman skeemaan soveltuvat tyyppiturvalliset tietokantakyselyt Typescript-ohjelmointikielelle (What is Prisma? n.d.).

Prisma soveltuu käytettäväksi useimpien erilaisten GraphQL-palvelin toteutusten kanssa, koska Prisma ei ole riippuvainen mistään tietystä GraphQL-toteutuksesta (Prisma in your stack – GraphQL n.d.). GraphQL:n ratkaisijoissa (resolvers) käytetään Prisma Client -komponentin generoimia tietokantakyselyjä samaan tapaan kuin mitä tahansa muutakin ORM- tai SQL-työkalua käytettäisiin.

4.4 Typescript

Typescipt on Microsoftin kehittämä laajennus Javascript-ohjelmointikielelle. Laajennuksella tarkoitetaan sitä, että Typescript sisältää kaikki tavallisen Javascript-ohjelmointikielen ominaisuudet, mutta tarjoaa lisäksi uusia ohjelmointikieltä laajentavia ominaisuuksia, joita voidaan halutessa käyttää. Teknisesti Typescipt ei ole oma ohjelmointikielensä, vaikka sitä siksi tavanomaisesti kutsutaan. (Educative n.d.)

Merkittävin Typescript-ohjelmointikielen ominaisuus on sen sisältämä valinnainen tyypitysjärjestelmä, jota hyödyntämällä voidaan helpottaa virheiden ja bugien huomaamista ohjelmakoodista sovelluskehityksen aikana (Educative n.d.). Typescriptin tyypitysjärjestelmän käytöllä mahdollistetaan staattisesti tyypitettyjen tietotyyppien käyttö ohjelmakoodissa. Tätä javascript ei ole mahdollistanut, koska se on dynaamisesti tyypitetty ohjelmointikieli (Typescriptlang n.d.).

Typescriptin tyypitysjärjestelmää vahvasti suositellaan käytettäväksi, koska tyypitysjärjestelmä on Typescriptin tärkein lisä ohjelmoimiseen. Typesciptiä voidaan käyttää, kaikista sen tuomista ominaisuuksista huolimatta, täysin samaan tapaan kuin Javascriptiä, mutta tällöin ohjelmakoodin muuttujat ovat dynaamisesti tyypitettyjä. Tämän vapaaehtoisuuden takia Typescript ei ole aidosti vahvasti tyypitetty staattinen ohjelmointikieli. Tyypitys on vain sovelluskehittäjälle päin näkyvää, koska Typescript joudutaan aina ennen ohjelmakoodin suorittamista kokoamaan kääntäjällä Javascriptiksi. (Typescriptlang n.d.)

Staattisesti tyypitetyn ohjelmointikielen käyttäminen on sovelluskehittäjälle usein työläämpää, mutta sillä tavoin virheiden määrää saadaan minimoitua, suurten ohjelmistojen ylläpitoa helpotettua ja koodista saadaan optimoidumpaa (Baeldung 2021). Sovelluskehityksen aikana on hyvä käyttää koodieditoria, joka ymmärtää nämä ohjelmointikielen tyypitykset ja osaa analysoida ohjelmakoodia jo ohjelmakoodin kirjoitusvaiheessa ennen lopulliseksi sovellustuotokseksi kääntämistä. Esimerkiksi Visual Studio Code koodieditorista löytyy monipuolinen tuki sekä Javascriptille että Typesciptille (Microsoft 2022).

4.5 Node.js

Node.js on avoimeen lähdekoodiin perustuva Javascriptin ajoympäristö, joka mahdollistaa järjestelmistä ja internetselaimista riippumattomasti Javascript-ohjelmakoodin suorittamisen. Ajoympäristön toiminta perustuu V8 Javascript -moottoriin, joka on käytössä monissa nykypäivän internetselaimissa. Ennen Node.js-ajoympäristön olemassaoloa ainoastaan internetselaimien omia Javascript-moottoreita voitiin käyttää Javascript-ohjelmakoodin ajoympäristöinä. Node.js-ajoympäristön julkaisun myötä Javascript-ohjelmointikielen suosio lähti kasvamaan yleiskäyttöisempänä ohjelmointikielenä. (Herron 2020, 9–10.)

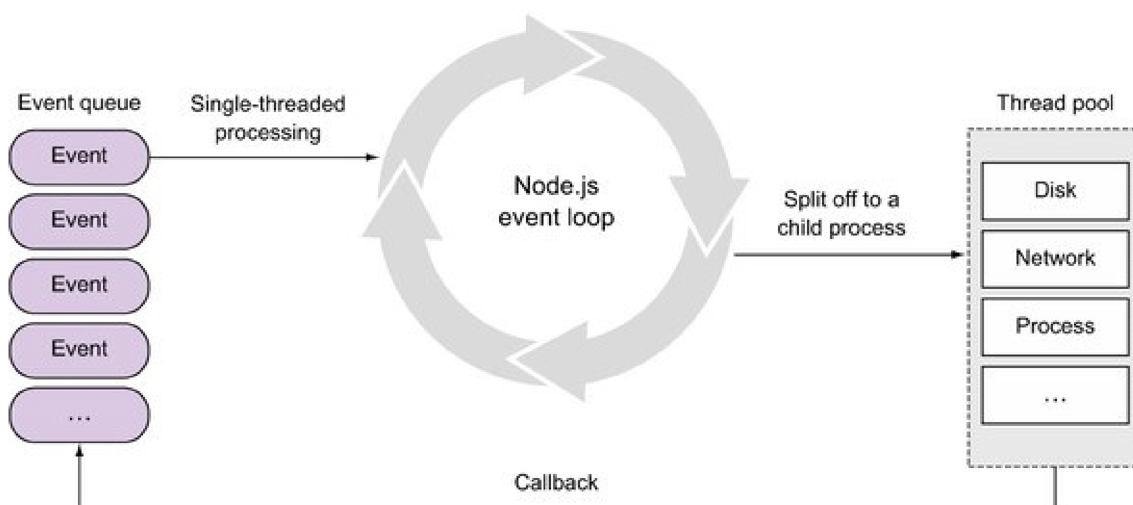
Ajansaatossa sovelluskehitysmarkkinoille on tullut tarjolle, niin frontend- kuin backend-puolen sovelluskehitykseen, useita erilaisia Javascript-pohjaisia sovelluskehyksiä, jotka tarvitsevat Javascript-ohjelmakoodin ajoympäristön. Verkkosovellus voidaan nykyään kokonaisuudessaan toteuttaa Node.js-ajoympäristöjä käyttämällä. Se on helpottanut ja nopeuttanut verkkosovelluskehitystyötä, kun eri sovelluskerroksilla voidaan toimia yhtenäisemmillä tavoilla sekä samoilla ohjelmointikielillä. (Herron 2020, 15–16.)

4.5.1 Arkkitehtuuri

Node.js-ajoympäristön arkkitehtuuri poikkeaa tyypillisistä ohjelmakoodien ajoympäristöistä siten, että se on lähestulkoon yksisäikeinen tapahtumaohjattu arkkitehtuuri, joka ei perustu monisäikeisyyden hyödyntämiselle ohjelmoinnissa tyypilliseen tapaan. Yksisäikeisyys Node.js-ajoympäristössä koskettaa vain niin kutsuttua tapahtumasilmukkaa (event loop), joka käsittelee kaikki ohjelmakoodin tapahtumat ja jakaa niitä omiin prosesseihin suoritettaviksi. (Herron 2020, 17–18.)

Kuviossa 4 on havainnollistettu Node.js-ajoympäristön tapahtumaohjattua arkkitehtuuria. Kuvion keskiössä oleva tapahtumasilmukka vastaanottaa jonossa olevat tapahtumat ja ohjaa ne käytössä oleville säikeille ja prosesseille suoritettaviksi. Valmistuneet tapahtumat ohjataan sen jälkeen takaisin tapahtuman kutsujalle. Ohjelmakoodissa tätä samanaikaisuutta ja monisäikeisyyttä lähestytään takaisinkutsufunktioin (callback function) tai asynkronisin funktioin (async function),

jotka ajoympäristö käsittelee tapahtumina, jolloin ohjelmasuoritteen suoritus ei esty. (Herron 2020, 17–21.)



KUVIO 4. Node.js-ajoympäristön tapahtumasilmukka (Hochhaus & Schoebel 2016)

Node.js:n arkkitehtuurin ansiosta saavutetaan pienempi muistin kokonaiskäyttö ja pienemmät latenssit kuormittavien ohjelmasuorituksien aikana, kun suoritus on jaettuna pienemmiksi tapahtumiksi. Itse suoritettavasta ohjelmakoodista saadaan yksinkertaisempaa ja helpommin ymmärrettävää, kun monisäikeisyys ei ole ohjelmakoodin tasolla hyödynnettävissä perinteisin keinoin. Mahdolliset ohjelmointivirheet ovat myös helpommin havaittavissa ohjelmakoodista. (Herron 2020, 19–20)

4.5.2 NPM ja YARN

NPM on komentoriviltä käytettävä Node.js:n oletuspakettienhallintatyökalu, jolla voidaan tuoda kolmannen osapuolen ohjelmamoduuleja Node.js-pohjaisiin sovellusprojekteihin. Sovellusprojekteissa näitä ohjelmamoduuleita kutsutaan riippuvuuksiksi, joita sovellusprojekti tarvitsee suoritukseensa. Näihin riippuvuuksiin viittaukset koostetaan Node.js-sovellusprojekteissa package.json-tiedostoon standardin mukaisessa formaatissa, jota NPM ymmärtää. (Herron 2020, 108–110.)

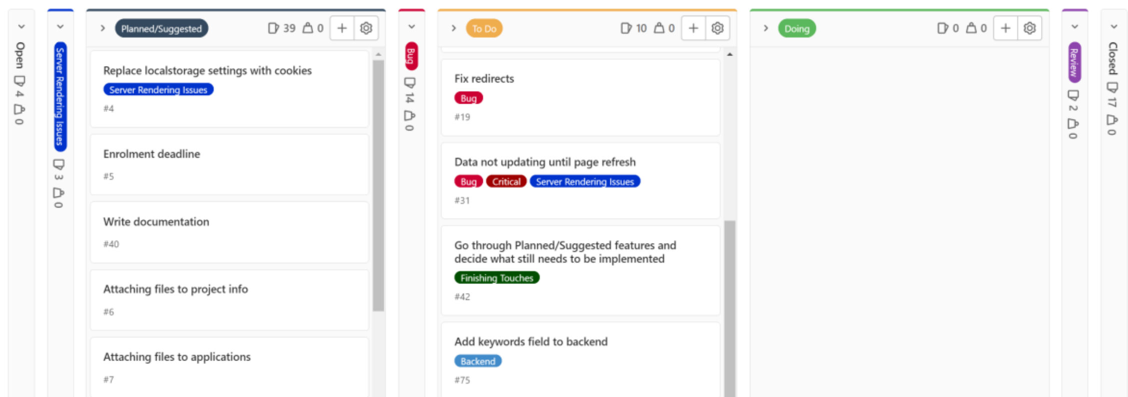
Node.js-ajoympäristö ei rajoita, mitä pakettienhallintatyökalua ohjelmaprojektissa käytetään. Sen vuoksi Node.js:n kehittäjäyhteisö on kehittänyt NPM:n kanssa kilpailevia pakettienhallintatyökaluja. Yksi merkittävimpiä on YARN. Sen väitetään olevan nopeampi, turvallisempi ja luotettavampi kuin NPM (Herron 2020, 128–129).

5 PROJEKTORIN ESITTELY

5.1 Sovelluksen suunnittelu

Projektorille tehtiin suunnittelu- ja vaatimusmäärittelytyötä niin asiakkaan kuin sovelluskehittäjien toimesta. Suunnittelutyön tuloksena syntyi tekninen määrittely ja valinnat sopivista teknologioista. Suunnittelu- ja vaatimusmäärittelytyötä jatkettiin sovelluksen toteuttamisen aikana, koska sovelluksen lopullinen muoto käyttötarkoitukseen ei ollut täysin vielä selvillä.

Suunnittelutyön tuloksia pilkottiin sopiviksi tehtäviksi GitLab-versiohallintajärjestelmän tehtävätaululle. Gitlab on Git-versionhallintaan pohjautuva sovellus, joka tarjoaa työkaluja sovelluskehitysprojektin elinkaaren eri vaiheisiin (Gitlab n.d.). Kuvassa 1 on näkyvillä Projektoria varten tehty Gitlab-tehtävätaulu. Tehtäviä tehtävätaululla liikutellaan vasemmalta oikealla sitä mukaan, kun ne valmistuvat edellisestä tehtävävaiheesta.

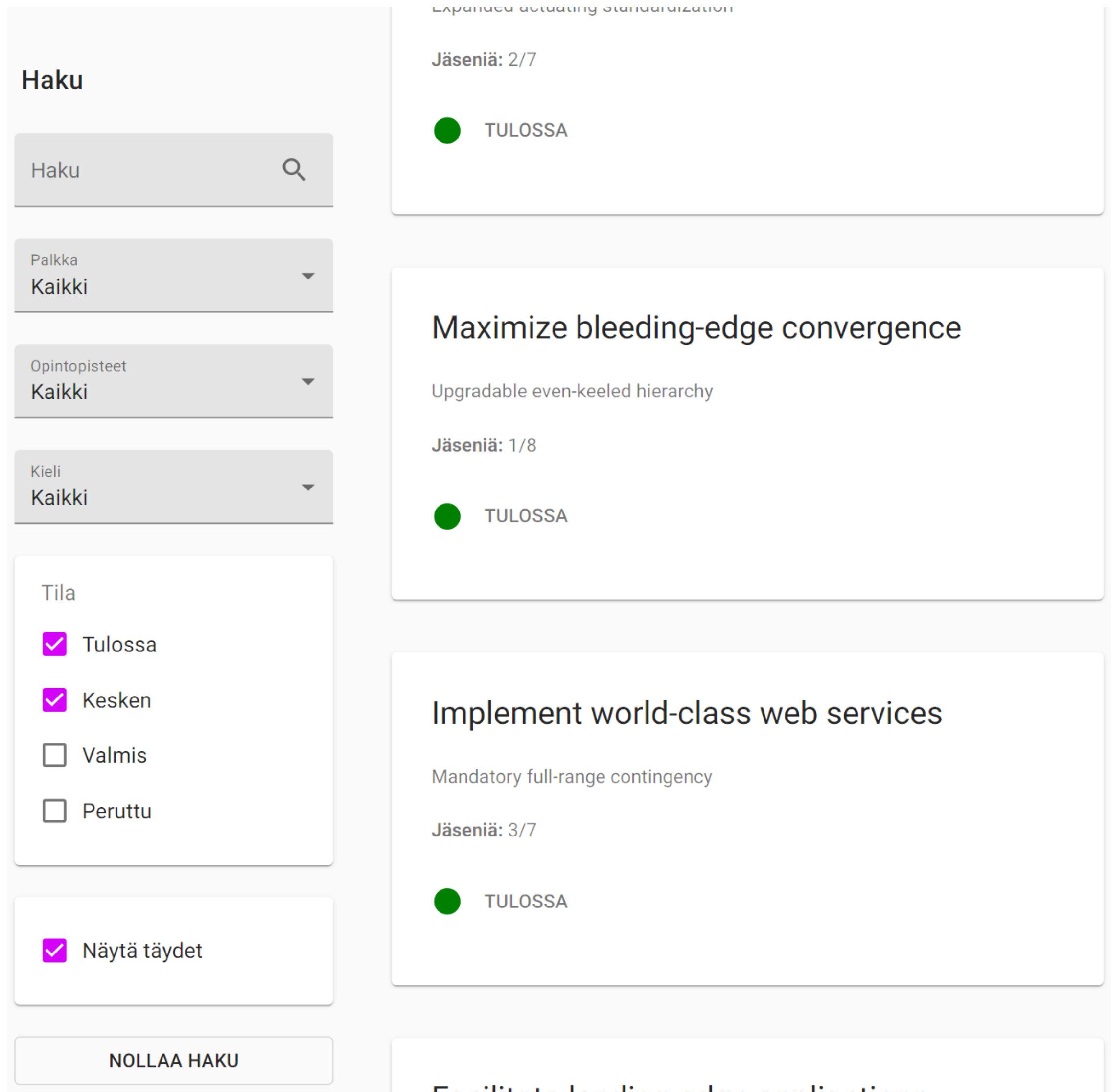


KUVA 1. Projektorin GitLab-tehtävätaulu

5.2 Sovellukseen toteutettu toiminnallisuus

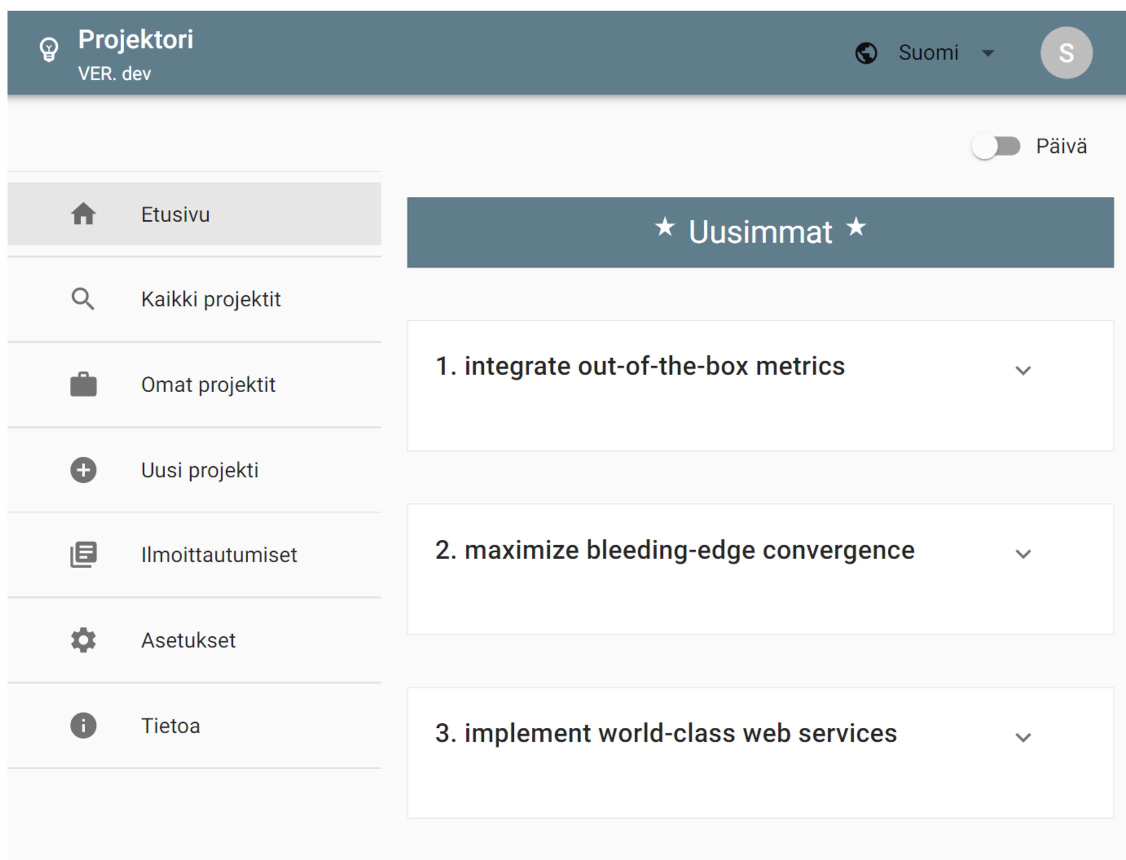
Projektorissa sovelluskäyttäjä pystyy hakemaan erilaisin hakuehdoin työmahdollisuuksia eli tietueita, jotka ovat häntä itseä kiinnostavia. Kuvassa 2 on Projektorin käyttöliittymästä näkymä, johon listataan kaikki sovelluksesta löytyvät tietueet.

Kuvan vasemmassa laidassa näkyy myös sovelluksen hakutoiminto, jossa on erilaisia hakuvaihtoehtoja tietueiden hakuun.



KUVA 2. Työmahdollisuuksien listaus- ja hakunäkymä Projektorista

Projektorissa on kirjautumisjärjestelmä. Sisään kirjautunut sovelluskäyttäjä, käyttäjätason mukaan lisätä, muokata ja poistaa tietueita sekä ilmoittautua tietueisiin osallistujaksi. Kuvassa 3 on näkymä, jonka vain sisään kirjautunut sovelluskäyttäjä näkee. Vasemmassa laidassa on näkyvillä osa niistä toiminnoista, joita sisään kirjautunut sovelluskäyttäjä voi sovelluksessa tehdä.



KUVA 3. Sisään kirjautuneen käyttäjän -näkymä Projektorissa

Projektorissa on yksinkertainen ilmoittautumisten hallintamahdollisuus tietueen hallinnoijalle. Tämä tarkoittaa sitä, että esimerkiksi tietueen luonut opettaja pystyy hallinnoimaan luomansa tietueen ilmoittautumisia, kun opiskelijat ovat ilmoittaneet halukkuutensa ottaa osaa tietueeseen.

Projektorissa on myös kokeellisesti testattu datan visualisoimista sanapilvien avulla. Niiden avulla voidaan tuoda esiin visuaalista statistiikkaa sovellukseen kerääntyneestä datasta. Tämä voisi mahdollistaa niin sanottujen ”kuumien” sanojen seuraamisen Projektorin tietueista. Kuvassa 4 on sovellukseen tehty sanapilvi. Mitä useammin jokin tietty sana mainitaan Projektoriin tehdyissä tietueissa, sen suuremmalla fontilla sana on nähtävissä sanapilvessä.

Suosituimmat avainsanat

KUVA 4. Sanapilvi esimerkkidatalla Projektorissa

6 PROJEKTORIN BACKEND-PUOLEN SOVELLUSKEHITYSTYÖ

Tässä luvussa käydään läpi Projektorin backend-puolen sovelluskehitystyötä projektiin valituilla backend-teknologioilla. Backend-puolen sovellus rakentuu aiemmin mainittuun Server-konttiin, jossa käytetään Node.js-ajoympäristöä. Ajoympäristön merkittävimpänä sovelluksena on Apollo Server, joka palvelee GraphQL-palvelinta. Server-kontista on vielä yhteys Database-konttiin, jossa suoritetaan PostgreSQL-tietokantapalvelinta. Tälle tietokantapalvelimelle kohdistuu koko sovelluksen suorituksen aikana tietokantakyselyitä Apollo Server -sovellukselta, kun GraphQL-palvelimen rajapintaa kutsutaan validilla GraphQL-rajapintaoperaatiolla. Sovelluskehityksen aikana tietokantapalvelimen tietokantamallia kehitetään Server-kontin sisältämän ohjelmakoodin avulla sekä komentorivikomennoilla.

6.1 Tietokanta

Sovelluksen Database-kontissa toimivaa tietokantaa käytetään täysin Server-kontista käsin Prisma-nimisen Node.js-ohjelmamoduulin kautta. Prisma-ohjelmamoduuli saadaan tuotua sovellusprojektiin komentorivikomennolla `yarn add prisma --dev`. Komento asentaa Prisman komentorivityökalun, jota käytetään sovelluskehityksen aikana Prisma Scheman, Prisma Migraten ja Prisma Clientin komentamiseen. Prisma Client -ohjelmamoduuli tuodaan vielä erikseen sovellusprojektiin komentorivityökalun käytettäväksi komentorivikomennolla `yarn add @prisma/client`.

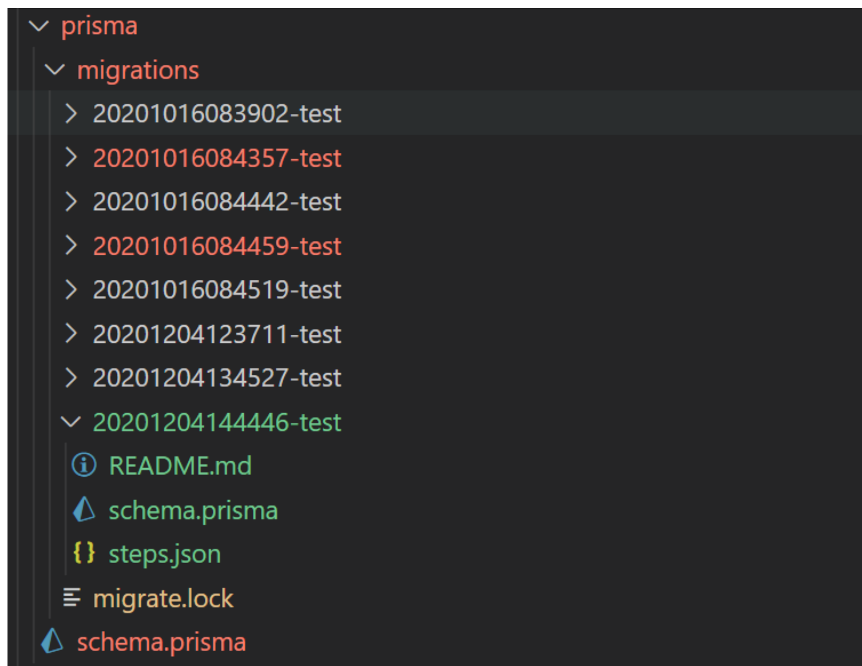
Komentorivikomennolla `yarn prisma init --datasource-provider postgresql` luodaan projektin juureen prisma-niminen kansio, joka sisältää Prisman skeematiedoston (Prisma Schema). Skeemasta löytyy määriteltävät asetukset liittyen Prisman käyttämään datalähteeseen ja Prisma Clientin generoimiseen. Samaiseen skeemaatiedostoon toteutetaan Prisman määrittelemällä tavalla malleina Projektorin tarvitsema tietokantamalli, jollainen sen halutaan olevan PostgreSQL-tietokannassa.

Kuvassa 5 näkyy esimerkiksi skeemaan tehty User-malli. Yksittäinen skeemaan tehty malli edustaa sovelluskehityksessä käytettävää olioesitystä, että tietokannassa sijaitsevaa tietokantataulua. Malli sisältää kentät, kenttien tyypit, mallien väliset suhteet ja mahdolliset lisäattribuutit liittyen kenttien ja mallien käyttäytymiseen.

```
33  model User {
34    id        String    @id @default(uuid())
35    email     String    @unique
36    name      String?
37    userLevel UserLevel @default(USER)
38    avatarUrl String?
39    gitlabId  Int?      @unique
40    createdAt DateTime  @default(now())
41    updatedAt DateTime  @updatedAt
42
43    memberDetails ProjectMember[]
44    settings      Settings?
45
46    initiatedProjects Project[]
47  }
```

KUVA 5. User-malli Prisman skeematiedostossa

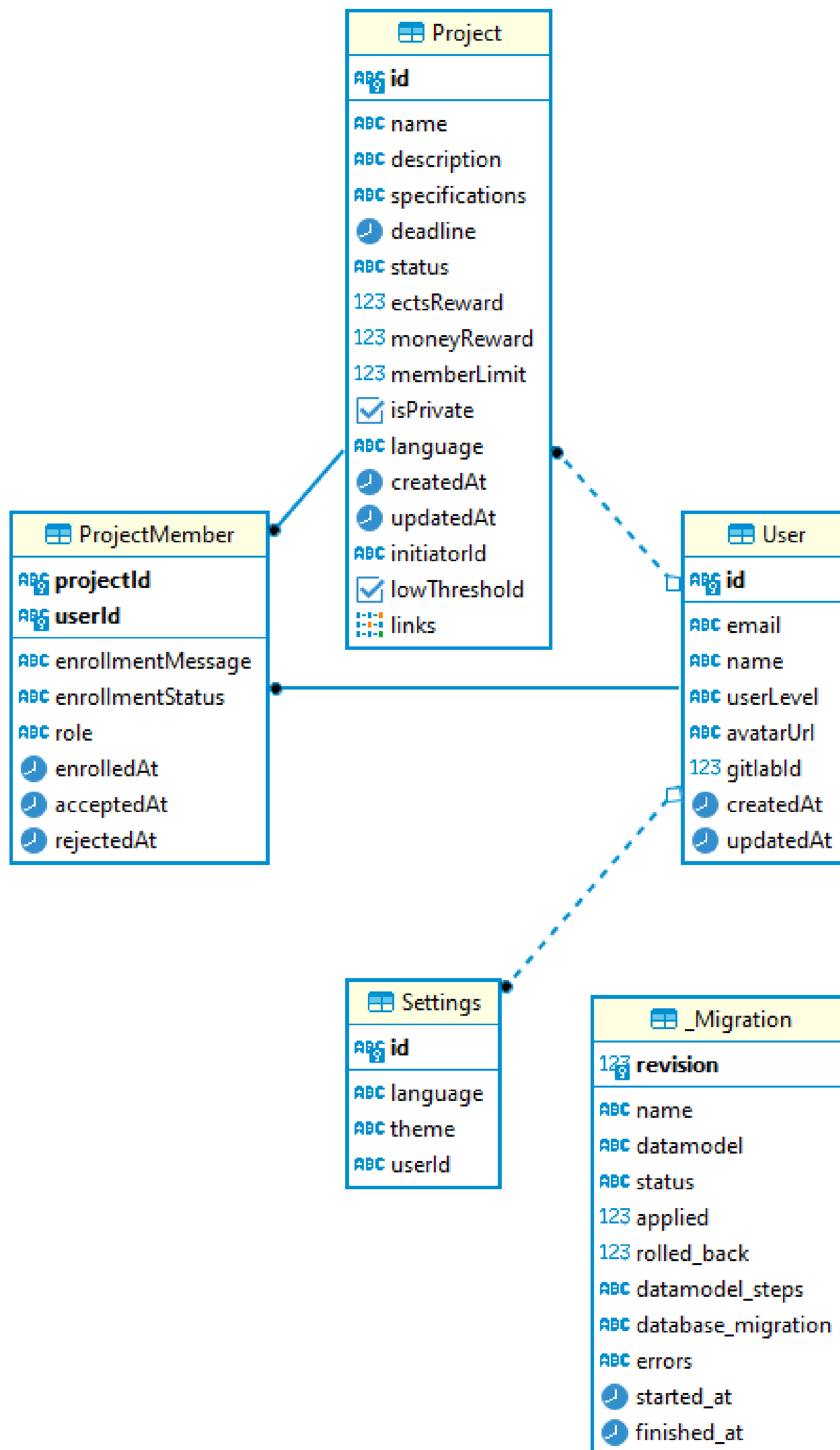
Prisman skeeman mallien perusteella luodaan tietokannan tietokantamalli. Komentorivikomennolla `yarn prisma migrate dev` Prisma generoi sql-käskyjä sisältävän migraation, jolla tietokannan tietokantamalli saadaan synkronoitua Prisman skeeman mukaiseksi. Yksittäisessä migraatiossa on huomioituna jo historiassa aiemmin tehdyt skeemamuutokset. Sovellusprojektin Prisman migrations-kansioon muodostuu historiaa kaikista tietokantaan tehdyistä migraatioista, kuten kuvassa 6 näytetään. Sovelluskehityksen aikana migraatioita voi muodostua useampia, kun Prisman skeemaa muutetaan ja muutokset halutaan synkronoida tietokantaan asti.



KUVA 6. Prisman luoma migraatiohistoria

Kuvassa 7 on näkyvillä Projektorin tietokantamalli tietokantakaaviona kuvattuna, kun tarkastellaan, miltä tietokanta näyttää tietokantapalvelimella silloin, kun kaikki Prisman migraatiot ovat ajettuna tietokantaan. Tietokannan _Migration-tietokantataulu säilyttää tietokantapalvelimella migraatiohistorian onnistumis-, poisto- ja muuttumishistorioineen. Muut tietokannan tietokantataulut ovat Prisman skeeman mallien mukaisia.

Projektorin tietokantamallissa User-taulun tietueella on yhden suhde moneen -tyyppinen yhteys Project-tauluun, mikä Projektori-sovelluksessa tarkoittaa sitä, että projektilla on oltava yksi sovelluskäyttäjä omistajana ja sovelluskäyttäjä voi omistaa useampia projekteja. User-taulun tietueesta löytyy myös yhden suhde moneen -yhteys ProjectMember-välitauluun, johon tulee yhden suhde moneen -yhteys Project-taulusta. Projektori-sovelluksessa tämä tarkoittaa sitä, että sovelluskäyttäjä voi olla ilmoittautuneena jäsenenä useammassa eri projektissa ja sovelluskäyttäjän jäsenyys projektiin sisältää lisätietoja.



KUVA 7. Prisman skeeman pohjalta tietokantaan generoitunut tietokantamalli tietokantakaaviona

Komentorivikomennolla `yarn prisma generate` generoidaan Prisma Clientista tuore versio Prisman skeemaan pohjautuen. Tämä sisältää tyyppiturvallisen rajapinnan tietokantaan sekä valmiit tietokantaan kohdistuvat CRUD-operaatiofunktiot. Näitä operaatiofunktioita hyödynnetään GraphQL-ratkaisijoissa.

6.2 Apollo Server -palvelinsovelluksen konfigurointi

Apollo Server -sovelluskehyksestä tuodaan Express-sovelluskehyksen kanssa integroituva versio sovellusprojektiin komentorivikomennolla `yarn add apollo-server-express graphql`. Express-sovelluskehys tarjoaa selkeän ja joustavan ohjelmointirajapinnan Node.js:n HTTP-rajapintaan (Express n.d.). Samalla komennolla tuotiin myös Apollo Server -sovelluskehyksen tarvitsema GraphQL, josta Apollo Server -sovelluskehys saa logiikan GraphQL-ominaisuuksien käyttämiseen.

Express-sovelluskehystä käyttämällä sovelluksen rajapinta voidaan rakentaa halutessa samanaikaisesti GraphQL-palvelimesta sekä REST API:sta. Projektorin rajapinta, tästä huolimatta, rakentuu pelkästään GraphQL-palvelimen GraphQL-rajapinnan varaan, eikä näin ollen REST API:lle ole tarvetta. Express-sovelluskehys kuitenkin tarjoaa mahdollista myöhempää tarvetta varten monipuolisia ominaisuuksia, joilla voidaan vastata sovelluskehityksen aikaisiin haasteisiin koskien esimerkiksi GraphQL-palvelimen tietoturvaa, autentikointia ja verkkoliikenteen rajoituksia.

Kuvassa 8 luodaan ApolloServer -luokasta ApolloServer -instanssi, joka vaatii vähintään Apollon GraphQL-skeeman ja -ratkaisijoiden antamista argumentteina objektissa. Samaisessa objektissa annetaan vapaaehtoisena argumenttina context-argumentti, jota käytetään hyödyksi GraphQL-ratkaisijoissa autentikoituineen käyttäjän tunnistamisessa.

```
39  const server = new ApolloServer({
40    typeDefs,
41    resolvers,
42    context: async ({ req }: any) => {
43      console.log("Server context function...");
44
45      // If user has valid JWT, req.user will contain user data
46      // Otherwise req.user will be undefined
47      const { user } = req;
48
49      // Send context variables to resolvers
50      return { currentUser: user };
51    },
52  });
```

KUVA 8. Apollo Server -instanssin luominen

ApolloServer-instanssi, joka vastaa GraphQL-palvelimen toteutuksesta, asetetaan Express-sovelluskehykseen middlewarena. Kuvassa 9 ApolloServer-instanssin applyMiddleware-metodille annetaan argumentteina objektissa Express-sovelluskehyksen instanssi ja rajapintapäätepisteen verkkopolku, jossa GraphQL-palvelimen halutaan sijaitsevan. Lopuksi Express-sovelluskehys käynnistetään sen listen-metodilla määriteltyyn TCP-porttiin kuuntelemaan yhteyksiä.

```
54  server.applyMiddleware({ app, path: "/api/graphql" });
55
56  // The `listen` method launches a web server.
57  app.listen({ port }, () =>
58    console.log(
59      `🚀 Server ready at http://localhost:${port}${server.graphqlPath}`
60    )
61  );
```

KUVA 9. GraphQL-palvelimen asettaminen Express-sovelluskehykseen

6.3 GraphQL-palvelin

6.3.1 Skeema

Apollo GraphQL-palvelimen GraphQL-skeemassa (GraphQL schema) on GraphQL-rajapinnan esittely. Skeemasta käytetään ohjelmakoodissa nimitystä typeDefs. typeDefs on jaoteltu seuraavasti:

- *Types* sisältää rajapinnan edustamat objektityypit kenttineen, joita rajapinta kykenee käsittelemään.
- *Enums* sisältää rajapinnan enumeraatiotyypit, jotka on haluttu määritellä ja jotka eivät ole osa GraphQL-spesifikaation valmiita tyyppejä.
- *Queries* sisältää rajapinnan kyselyoperaatioiden esittelyt.
- *Mutations* sisältää rajapinnan muutoksia aiheuttavien operaatioiden esittelyt.

Kuvassa 10 on esiteltynä esimerkiksi GraphQL-skeemassa oleva User-objektityyppi kenttineen. User-objektityypillä on 13 kenttää, joista valtaosa on määritetty GraphQL-spesifikaation valmiilla tyypeillä. Enumeraatiotyyppinen kenttä on esimerkiksi userLevel-kenttä, jolle enumeraatiotyyppinä on UserLevel. Objektityyppinen kenttä on esimerkiksi joinedProjects, jolla tyyppinä on lista Project-objektityyppiä. Kappaleessa 6.3.3 havainnollistetaan, kuinka esimerkiksi tätä User-objektityyppiä voidaan hakea rajapinnasta sopivalla GraphQL-kyselyoperaatiolla.

```
18  type User {
19    id: ID!
20    email: String!
21    name: String
22    # Exists if gitlab authProvider has been used
23    gitlabId: Int
24    avatarUrl: String
25    userLevel: UserLevel
26    createdAt: String
27    updatedAt: String
28    # Personal settings
29    settings: Settings
30    # Limitless use: userLevel: ADMIN or higher, Restricted use: Only to himself/herself.
31    joinedProjects(enrollmentStatus: EnrollmentStatus): [Project]
32    # Requirements: defined in "members"
33    memberDetail: MemberDetail
34    # Limitless use: userLevel: ADMIN or higher, Restricted use: Only to himself/herself.
35    userStatistics: CountMyData
36    # Logged-in. Limitless use: userLevel: ADMIN or higher, Restricted use: Only to himself/herself
37    initiatedProjects: [Project]
38  }
```

KUVA 10. User-objektityyppi GraphQL-skeemassa

GraphQL-skeeman kyselyoperaatiot ja muutoksia aiheuttava operaatiot ovat erityisiä GraphQL-objektityyppejä. Kyselyoperaatiot määritellään Query-tyypin alaisuuteen ja muutoksia aiheuttavat operaatiot Mutations-tyypin alaisuuteen. Nämä tyypit ovat samankaltaisia kuin muut objektityypit, mutta niiden tulee sisältää kaikki rajapinnan CRUD-operaatioiden esittelyt.

Kuvassa 11 on esiteltynä esimerkiksi Query-tyypin sisältämät currentUser- ja users-kyselyoperaatiot. currentUser-kyselyoperaatiolla haetaan sovellukseen sisään kirjautuneen käyttäjän käyttäjätiedot ja puolestaan users-kyselyoperaatiolla kerralla useampaa käyttäjätietoa. Molemmat kyselyoperaatiot palauttavat User-objektityyppiä vastauksessaan. users-kyselyoperaatio ottaa lisäksi vastaan vapaaehtoiset id-, search- ja order-argumentit. Näillä voidaan asettaa ehtoja kyselyoperaation suorittamiseen ja myös rajata tuloksia.

```
 8  type Query {
 9      # Requirements: Logged-in.
10      currentUser: User
11
12      # Requirements: Logged-in.
13      # Limitless use: userLevel: ADMIN or higher, Restricted use: Only to himself/herself
14      users(
15        # Find by id
16        id: ID
17        # Search by field
18        search: SearchUserInput
19        # Order by field
20        order: OrderInput
21      ): [User]
22
23  .
24  .
25  .
26
```

KUVA 11. Query-tyypin esittämät currentUser- ja users-kyselyoperaatiot

6.3.2 Ratkaisijat

Skeeman toteuttamiseksi tarvitaan GraphQL-ratkaisijat (GraphQL resolvers). Ratkaisijat sisältävät funktioita, jotka vastaavat GraphQL-skeeman rajapintaoperaatioiden toiminnallisuuden suorittamisesta ja objektityyppien kenttien täyttämisestä tiedoilla.

Kuvassa 12 on Query-tyypin users-kyselyoperaation asynkroninen ratkaisijafunktio, jolla on tarkoitus hakea kerralla useampia käyttäjätietoja tietokannasta. Funktion args-parametri sisältää ehtoja kyselyoperaation suorittamiseen ja currentUser-parametri sisältää järjestelmään sisään kirjautuneen käyttäjän tiedot. Funktio palauttaa onnistuessaan listan User-objektityyppiä.

Funktiossa ensimmäiseksi tarkistetaan sisään kirjautuneen käyttäjän käyttöoikeus rajapintaoperaation suorittamiseen. Jos sisään kirjautunut käyttäjä läpäisee käyttöoikeusvaatimukset, suoritetaan varsinainen rajapintaoperaation toiminnallisuus tietokantakyselyllä.

Tietokantakysely tehdään Prisma Clientin luoman rajapinnan kautta tietokantaan. prisma.user.findMany()-funktiolla kohdennetaan tietokantakysely tietokannan User-tauluun, josta palautetaan kaikki mahdolliset käyttäjätietueet, jotka sopivat

findMany()-funktiolle annettujen argumenttien ehtoihin. Funktio palauttaa listan Prisman User-mallin mukaisia objekteja, mikä annetaan myös users-funktion palautukseen.

```
179   Query: {
180     users: async (_, args, { currentUser }) => {
181       const { id, search, order } = args;
182       if (
183         (!currentUser || currentUser.id !== id) &&
184         (!currentUser || !checkAuth(currentUser.userLevel, UserLevel.ADMIN))
185       ) {
186         throw new Error("You are not authorized");
187       }
188       const users = await prisma.user.findMany({
189         where: {
190           id,
191           email: search?.email ?? undefined,
192           name: search?.name ?? undefined,
193           gitlabId: search?.gitlabId ?? undefined,
194           avatarUrl: search?.avatarUrl ?? undefined,
195           userLevel: search?.userLevel ?? undefined,
196         },
197         orderBy: order
198           ? { [order.field]: order.direction }
199           : { createdAt: "desc" },
200         include: { settings: true },
201       });
202       return users;
203     },
204  .
205  .
206  .
```

KUVA 12. users-kyselyoperaation ratkaisijafunktio

Prisman User-malli (kuva 5) ja GraphQL-skeeman User-objektityyppi (kuva 10) sisältävät suurelta osin samannimisiä kenttiä. Kenttien nimeäminen on tarkoituksella tehty yhtenevästi, sillä oletuksena ratkaisijoiden Query-tyypin ratkaisijafunktiot hyödyntävät palautuksiin annettuja samannimisiä kenttiä objektityypin kenttien täyttämiseen. Niiltä osin, kun User-mallin kentät eivät vastaa User-objektityypin kenttiä tai oletustoiminnallisuus halutaan ohittaa, täytyy luoda User-objektityypin kenttäratkaisijafunktioita ratkaisijoihin. Näitä User-objektityypin kenttäratkaisijafunktioita kutsutaan automaattisesti, kun jokin Query-tyypin ratkaisijafunktio haluaa palauttaa User-objektityypin. User-objektityypillä on muutamia kenttiä, joita varten on tarvinnut luoda kenttäratkaisijafunktioita. Tällainen on esimerkiksi joinedProjects-kenttä, jota varten on täytynyt luoda User-objektityypin kenttäratkaisijafunktio.

6.3.3 Rajapintaoperaatiot

Apollo Serverin mukana tulee GraphQL Playground, joka on käytettävissä kehitysympäristössä GraphQL-rajapintapäätepisteestä. GraphQL Playgroundia voidaan hyödyntää sovelluskehityksen aikana GraphQL-rajapinnan ominaisuuksien tutkimiseen ja GraphQL-rajapintaoperaatioiden testaamiseen.

Kuvassa 13 näkyy GraphQL Playgroundin esittämänä Projektorin rajapintadokumentaatio. Rajapintadokumentaatio perustuu tehtyyn GraphQL-skeemaan. Rajapintaoperaatioiden ja objektityyppien kenttien nimiä klikkaamalla pääsee näkemään tarkemman dokumentaation operaatiosta tai kentästä.

Backend-sovelluskehittäjien vastuulla on ylläpitää rajapintadokumentaatiota niin frontend-sovelluskehittäjiä kuin muita backend-sovelluskehittäjiä varten. Rajapintaoperaatioiden ja objektityyppien kenttien nimeäminen sekä sanallinen lisädokumentaatio ovat tärkeitä olla asiayhteyteen sopivia. Dokumentaatiota lukevan on saatava vaivattomasti käsitys siitä, miten rajapintaa hyödynnettäisiin sopivissa käyttötarkoituksissa.

KUVA 13. Projektorin GraphQL-rajapintadokumentaatio GraphQL Playgroundin esittämänä

Kuvassa 14 on tehty GraphQL-rajapintakyselykielinen kyselyoperaatio GraphQL Playgroundissa. Vasemmalla näkyy users-kyselyoperaatio ja oikealla siihen saatu JSON-muotoinen vastaus rajapinnasta. Esimerkiksi tällä kyselyoperaatiolla haetaan kaikki ADMIN-käyttäjätason käyttäjät. Näistä käyttäjistä palautetaan nimi, käyttäjätaso ja projektit, joihin käyttäjä on liittynyt. Näistä projekteista puolestaan näytetään projektin nimi, projektin toteuttamiseen suunniteltu jäsenmäärä ja käyttäjä, joka on projektin luonut. Projektin luoneesta käyttäjästä näytetään nimi ja käyttäjätaso.

GraphQL-rajapintakyselykielellä voidaan hakea GraphQL-rajapinnan kautta täsmällisesti halutut kentät, kuten kuvassa 14 osoitetaan. GraphQL-rajapintakysely on rakennettu hierarkkisesti siten, että pelkästään yhdellä rajapintakyselyllä

saadaan haettua toisiinsa kytkeytyvää dataa. Näin vältetään tarve muodostaa useampia erillisiä rajapintakyselyitä.

```
query {
  users(search: { userLevel: ADMIN }) {
    name
    userLevel
    joinedProjects {
      name
      memberLimit
      initiator {
        name
        userLevel
      }
    }
  }
}
```


```
{
  "data": {
    "users": [
      {
        "name": "Matti Meikäläinen",
        "userLevel": "ADMIN",
        "joinedProjects": [
          {
            "name": "generate bricks-and-clicks relationships",
            "memberLimit": 8,
            "initiator": {
              "name": "Maija Meikäläinen",
              "userLevel": "STAFF"
            }
          },
          {
            "name": "transition next-generation applications",
            "memberLimit": 1,
            "initiator": {
              "name": "Erkki Meikäläinen",
              "userLevel": "STAFF"
            }
          }
        ]
      },
      {
        "name": "Liisa Meikäläinen",
        "userLevel": "ADMIN",
        "joinedProjects": [
          {
            "name": "engineer dynamic platforms",
            "memberLimit": 8,
            "initiator": {
              "name": "Pertti Meikäläinen",
              "userLevel": "STAFF"
            }
          }
        ]
      }
    ]
  }
}
```


KUVA 14. users-kyselyoperaatio Projektorin GraphQL-rajapintaan

7 POHDINTA

Tässä opinnäytetyössä rakennettiin valituilla moderneilla teknologioilla Projektori-verkkosovelluksen backend-puolta. Tekemisen aikana perehdyttiin teknologioiden sisältämiin paradigmoihin liittyen backend-puolen sovelluksen rakentumiseen. Lisäksi selvitettiin, millaisiin palasiin nykypäivän verkkosovellus voi olla jaettuna.

Projektori-verkkosovelluksen backend-puolelle valmistui toimintakuntoinen sovellus. GraphQL osoittautui toimivaksi ja joustavaksi teknologiaksi, jolla toteuttaa rajapinta verkkosovellukseen. Projektorin GraphQL:n avulla toteutetusta rajapinnasta löytyy nyt merkittävä osa toivotuista, perustuvanlaatuisista rajapintaoperaatioista, joita Projektorin frontend-puolelta käsin päästään hyödyntämään.

Projektorin kehitys- että suunnittelutyö jäivät kesken, mutta huolella tehty koodi ja dokumentaatio mahdollistavat sovelluksen jatkokehittämisen eteenpäin tietotekniikan koulutusohjelman sisällä tulevaisuudessa. Monet nyt toteutetut toiminnot vaativat vielä pientä hienosäätöä ennen kuin Projektori voidaan julkaista todellisten sovelluskäyttäjien saataville.

Sovellusta rakennettiin hyvin ominaisuuspainotteisesti eteenpäin, jonka vuoksi testaaminen jäi osin puutteelliseksi. Sovelluksen kirjautumisjärjestelmän toimivuus ja rajapinnan käyttöoikeustarkastelun pätevyys olisi hyvä esimerkiksi varmentaa testaamisella. Ylipäätänsä hyviin ohjelmointitapoihin kuuluisi ohjelmallisten yksikkötestien tekeminen, mutta nyt niihin perehtymiseen ei riittänyt aika, joten niiden tekeminen jää jatkokehittäjille. Lisäksi tulevaisuutta ajatellen tarvitaan uusia suurempia suuntaviivoja sen suhteen, mihin suuntaan Projektorin kehitystyötä viedään jatkossa.

Tämän kokoluokan sovellusprojektissa mukana oleminen oli opettavaista. Sovellussuunnittelu- ja tiimityöskentelytaidot kehittyivät paljon. Teknologioihin perehtyminen ja teknologioiden käyttäminen toivat paljon hyödyllistä osaamista.

LÄHTEET

Baeldung. 2021. Statically Typed Vs Dynamically Typed Languages. Verkko-sivu. Luettu 2.3.2022. https://www.baeldung.com/cs/statically-vs-dynamically-typed-languages

Choi, D. 2020. Full-Stack React, Typescript, and Node : Build Cloud-Ready Web Applications Using React 17 with Hooks and GraphQL. Birmingham: Packt Publishing https://andor.tuni.fi/permalink/358FIN_TAMPO/1j3mh4m/alma9911211183505973

Docker. n,d. Docker overview. Verkkosivu. Luettu 24.4.2022 https://docs.docker.com/get-started/overview/

Educative. n.d. What is TypeScript?. Verkkosivu. Luettu 6.3.2022. https://www.educative.io/edpresso/what-is-typescript

Express. n.d. Using middleware Verkkosivu. Luettu 20.6.2022. https://expressjs.com/en/guide/using-middleware.html

Gitlab. n.d. What is GitLab?. Verkkosivu. Luettu 6.3.2022. https://about.gitlab.com/what-is-gitlab/

Haikala, I. & Märijärvi, J. 2004. Ohjelmistotuotanto. Helsinki: Talentum

Herron, D. 2020. Node.js Web Development. Birmingham: Packt Publishing https://andor.tuni.fi/permalink/358FIN_TAMPO/176jdvt/cdi_safari_books_v2_9781838987572

Hochhaus, S. & Schoebel, M. 2016. Meteor in action. 1st edition. Shelter Island, New York: Manning.

Introduction to Apollo Server. n.d. Apollo. Verkkosivu. Luettu 17.2.2022. https://www.apollographql.com/docs/apollo-server/

Kotilainen. S. 2020. Koodi sujahtaa konttiin – sovellusten kehittäminen mullistuu. Tivi. Verkkosivu. Luettu 21.3.2022. https://www.tivi.fi/uutiset/koodi-sujahtaa-konttiin-sovellusten-kehittaminen-mullistuu/7931ccac-1cd7-3c40-b338-be25469cf1dd

Microsoft. 2022. JavaScript and TypeScript in Visual Studio. Verkkosivu. Luettu 2.3.2022. https://docs.microsoft.com/en-us/visualstudio/javascript/javascript-in-visual-studio?view=vs-2022

Paterson, C. 2020. The troublesome "Active Record" pattern. Calpaterson. Verkkosivu. Luettu 13.4.2022. https://calpaterson.com/activerecord.html

PostgreSQL. 2022. About. Verkkosivu. Verkkosivu. Luettu 1.4.2022. https://www.postgresql.org/about

Prisma in your stack - GraphQL. n.d. Prisma. Verkkosivu. Luettu 12.4.2022.
https://www.prisma.io/docs/concepts/overview/prisma-in-your-stack/graphql

Prisma Migrate. n.d. Prisma. Verkkosivu. Luettu 12.4.2022.
https://www.prisma.io/docs/concepts/components/prisma-migrate

Prisma schema. n.d. Prisma. Verkkosivu. Luettu 9.4.2022.
https://www.prisma.io/docs/concepts/components/prisma-schema

Schema basics. n.d. Apollo. Verkkosivu. Luettu 17.2.2022.
https://www.apollographql.com/docs/apollo-server/schema/schema

The Apollo Graph Platform. n.d. Apollo. Verkkosivu. Luettu 17.2.2022.
https://www.apollographql.com/docs/intro/platform

The Software Guild. 2015. Build Your Own Technology Stack. Verkkosivu. Luettu 20.3.2022.
https://www.thesoftwareguild.com/blog/build-your-own-technology-stack/

Typescriptlang. n.d. TypeScript for JavaScript Programmers. Verkkosivu. Luettu 11.3.2022.
https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html

What is Prisma?. n.d. Prisma. Verkkosivu. Luettu 1.4.2022.
https://www.prisma.io/docs/concepts/overview/what-is-prisma

Why adopt GraphQL?. n.d. Apollo. Verkkosivu. Luettu 17.2.2022.
https://www.apollographql.com/docs/intro/benefits

Why Prisma?. n.d. Prisma. Verkkosivu. Luettu 9.4.2022.
https://www.prisma.io/docs/concepts/overview/why-prisma