Niko Viitiö

TAUSTAJÄRJESTELMÄN TOTEUTUS TAPAAMISALUSTA-VERKKOSOVEL-LUKSELLE

Opinnäytetyö

Tekniikan ammattikorkeakoulututkinto

Peliohjelmoinnin koulutus

2025

Kaakkois-Suomen ammattikorkeakoulu

Kaakkois-Suomen ammattikorkeakoulu

Tutkintonimike	Insinööri (AMK)
Tekijä	Niko Viitiö
Työn nimi	Taustajärjestelmän toteutus tapaamisalusta-verkkosovellukselle
Toimeksiantaja	Normogames Oy
Vuosi	2025
Sivut	47 sivua
Työn ohjaaja(t)	Marko Oras

TIIVISTELMÄ

Tässä opinnäytetyössä on dokumentoitu taustajärjestelmän suunnittelu ja toteutus verkkosovellukseen. Taustajärjestelmä koostuu WWW-palvelimesta, REST-rajapinnasta sekä relaatiotietokannasta. Työ toteutettiin Normogames Oy:lle, jonka saamaan asiakastyöhön taustajärjestelmä otettiin käyttöön. Verkkosovelluksen nimeksi tuli TeamUp.

Tavoitteena oli luoda taustajärjestelmä, joka soveltuu tämän asiakastyön verkkosovelluksen käyttötapauksille. Sovellus toimii yrittäville opiskelijoille sekä yrittäjyydestä kiinnostuneille oppilaille tapaamisalustana, johon he voivat lisätä ilmoituksia. Sovelluksen toiminta edellyttää taustajärjestelmältä CRUD-toimintojen suorittamista tietokannassa, mikä oli tämän opinnäytetyön tutkimusongelma. Työssä tutkittiin kuinka CRUD-toiminnot tapahtuvat vaiheittain asiakkaan lähettämästä HTTP-pyynnöstä aina itse tietokantakyselyn suorittamiseen sekä kuinka ne voidaan implementoida REST-rajapintaan.

Työssä toteutettiin suunniteltu taustajärjestelmä. Sen toiminnallisuuden osalta täytyi kuitenkin tehdä kompromisseja, joissa tasapainoteltiin käytettävyyden ja hyvien käytäntöjen välillä. Toteutuksessa käytettiin LiteSpeed-palvelinta ja rajapinta ohjelmointiin PHP-ohjelmointikielellä. Taustajärjestelmän tietokantana käytettiin MySQL-relaatiotietokantaa, johon rajapinta suorittaa SQL-kyselyt valmistellun lauseen avulla. Taustajärjestelmä kehitettiin siinä käytettävien teknologioiden hyvien käytäntöjen mukaisesti.

Työn tuloksena valmistui taustajärjestelmä, joka pystyy suorittamaan kaikki neljä CRUD-toimintoa: tietokantatauluun lisäämisen, lukemisen, päivittämisen sekä poistamisen. Sen lisäksi moderaattorille toteutettiin kirjautumislogiikka, jonka avulla sovelluksen väärinkäyttöä ehkäistään. Rajapinnan toteutuksessa syntyi paljon lähdekoodia, jota voidaan hyödyntää vastaavissa projekteissa.

Asiasanat: taustajärjestelmä, rest-rajapinta, tietokanta, verkkosovellus

Kaakkois-Suomen ammattikorkeakoulu

Degree title	Bachelor of Engineering
Author	Niko Viitiö
Thesis title	Implementation of a backend system for a meeting platform web application
Commissioned by	Normogames Oy
Time	2025
Pages	47 pages
Supervisor	Marko Oras

ABSTRACT

This thesis contains a documentation for designing and implementing a backend system for a web application. The backend system consists of a web server, a REST API and a relational database and it was deployed as part of the commissioner's client project.

The purpose of this thesis was to create a backend system that would meet the use case requirements of the client project and would serve as a meeting platform for students pursuing entrepreneurial studies or having an interest in entrepreneurship where they can discuss their ideas and expertise. The functionality of the web application requires the backend system to perform CRUD operations on the database. The study aimed to thoroughly examine CRUD operations from the HTTP request sent by the client to the execution of database query. Also, how they can be implemented in a REST API.

The designed backend system was successfully implemented. However, compromises had to be made in terms of balancing usability and best practices. The implementation utilized a LiteSpeed server, and the API was programmed using PHP. The backend system's database was MySQL relational database, with API executing SQL queries using prepared statements. The backend system was developed by following the best practices for the technologies used.

As a result, a backend system was created that can perform all the four CRUD operations: creating a new row into a database table, reading from it, updating it and deleting it. Additionally, a login system for moderators was implemented to prevent misuse of the application. The API implementation produced a great amount of source code that can be used in other similar projects. The web application was named TeamUp.

Keywords: backend, rest API, database, web application

SISÄLLYS

1 JOHDANTO

Tässä luvussa käsitellään taustajärjestelmiä yleisesti ja käydään läpi opinnäytetyön tavoite sekä toimeksiantaja.

1.1 Taustajärjestelmät

Taustajärjestelmät ovat nykyajan palveluiden selkäranka. Useat sovellukset tarvitsevat niitä toimiakseen ja ne ovat käytettävissä tietokoneen tai mobiililaitteen avulla kaikkialla, missä internetyhteys on saatavilla. Monet palvelut myös toimivat useammilla eri alustoilla tarjoten saumattoman kokemuksen. Esimerkiksi päivittäistavarakaupoista ainakin Lidl tarjoaa kanta-asiakkuuttaan ainoastaan digitaalisena mobiilisovelluksena (Mikä on Lidl Plus? s.a.).

Taustajärjestelmän tehtävänä näissä sovelluksissa on käsitellä ja hallinnoida asiakkaiden lähettämiä tietopyyntöjä. Taustajärjestelmän tulee myös varmistaa pyyntöjen oikeellisuus sekä estää haitalliset tietopyynnöt. Taustajärjestelmä vastaa kaikesta sovelluksen datasta ja välittää sitä asiakkaille takaisin vastauksina tietopyyntöihin.

Opinnäytetyössä kehitettävä sovellus vaatii myös taustajärjestelmän. Sovelluksen tarkoitus on toimia tapaamisalustana yrittäjille sekä yrittäjähenkisille korkeakouluopiskelijoille. Siinä pitää pystyä lisäämään uusia ilmoituksia, selaamaan jo olemassa olevia ilmoituksia sekä myös moderoimaan näitä ilmoituksia. Moderointitoimintoja ovat julkaiseminen, piilottaminen sekä poistaminen.

1.2 Työn tavoite

Tämän opinnäytetyön tavoitteena on valmistaa toimiva, selkeä, yksinkertainen sekä asiakastyön kriteerit täyttävä taustajärjestelmä. Verkkosovelluksen, jolle taustajärjestelmä kehitetään, on määrä toimia prototyyppinä. Sen avulla testataan sovelluksen toimintaa ja kartoitetaan sen tarvetta asiakkaalle. Näiden lisäksi työssä painotetaan hyvien käytäntöjen mukaisia ratkaisuja sekä otetaan

taustajärjestelmän kehityksessä mahdollinen jatkokehitys huomioon. Taustajärjestelmä tulee käyttämään REST-arkkitehtuurimallia hyödyntävää rajapintaa sekä MySQL-tietokantaa.

REST on lyhenne sanoista Reprensetational State Transfer. Se on rajapintojen suunnitteluun kehitetty arkkitehtuurimalli, joka perustuu tilattomuuteen. Tilattomuudella tarkoitetaan sitä, että asiakkaan (WWW-selain) lähettämät HTTP-pyynnöt sisältävät aina kaiken tarvittavan tiedon, jotta rajapintaa pystyy toimintoja suorittamaan. Tämä myös tarkoittaa sitä, että asiakkaan ja taustajärjestelmän ei tarvitse tietää toisistaan mitään, toisin sanoen niiden nykyistä tilaa. Tilattomuus mahdollistaa myös vaivattoman verkkosovelluksen kehityksen taustajärjestelmän sekä frontendin välillä. (What is REST? s.a.)

Frontend tarkoittaa sovelluksen käyttäjälle näkyvää osaa, jonka avulla se käyttää sovellusta. Tässä työssä verkkosovelluksen ollessa kyseessä frontend on sama asia kuin WWW-selain. WWW-selain siis toimii käyttöliittymänä, joka lähettää HTTP-pyyntöjä rajapinnalle. REST-rajapinta taas vastaa sovelluksen sisällön päivittämisestä asiakkaan lähettämien pyyntöjen mukaan.

Työn alkuvaiheessa tutkitaan REST-rajapintaa ja tietokantoihin liittyviä CRUD (Create, Read, Update, Delete) toimintoja. Näiden pohjalta suunnitellaan kokonaisuus, joka toteutetaan. Työn lopuksi käsitellään tuloksia ja pohditaan tutkimuskysymysten vastauksia. Työn on tarkoitus toimia pohjana sekä esimerkkinä yhdenlaisesta, toimivasta kokonaisuudesta. Työssä käytettyjä ratkaisuja voidaan hyödyntää vastaavissa projekteissa. Toteutuksessa käytetään PHP-ohjelmointikieltä ja SQL-kyselykieltä MySQL-tietokannassa. Valitut teknologiat perustuvat toimeksiantajan käyttämään WWW-palvelimeen, jossa näille on valmis tuki. Kumpikin näistä teknologioista on ilmainen, joka sekin on valinnan perusteena.

1.3 Toimeksiantaja

Työ toteutetaan Normogames Oy:lle, joka on vuonna 2022 perustettu kotkalainen pelialan yritys. Heidän päätoimenaan on videopelien kehittäminen, mutta myös alihankinnan tarjoaminen muille yrityksille. Toisin sanoen he tekevät

myös asiakastöitä, jollaiseen tämä opinnäytetyö myös valmistetaan. (Normogames Oy s.a.)

Toimeksiantajalla on tarve taustajärjestelmäosaamiselle, sillä he tarvitsevat niitä myös omissa videopeleissään. Niissä käytetään kolmansien osapuolien tarjoamia palveluita, jotka ovat Steamworks sekä Epic Online Services. Valve Corporation on kehittänyt Steamworks taustajärjestelmän ja Epic Online Services on taas Epic Games yrityksen kehittämä taustajärjestelmä. Molemmat näistä taustajärjestelmistä ovat videopelien tarpeiden mukaan räätälöityjä ja tarjoavat laajalti eri ominaisuuksia. Varjopuolena näissä kuitenkin on se, että ne eivät aina sovellu jokaiseen videopeliin eli toisin sanoen ne eivät pysty tarjoamaan sovelluskohtaisia ominaisuuksia. Ratkaisuna tähän olisi videopeliä varta vasten suunniteltu ja toteutettu taustajärjestelmä. Koska Normogames Oy tarvitsee taustajärjestelmää videopeleissään sekä asiakastöissään, tulee tämä työ tarjoamaan tietoa taustajärjestelmän kehittämiseen liittyvistä seikoista.

2 TUTKIMUSASETELMA

Tämä työ tehdään koska toimeksiantaja tarvitsee sovelluksen, jonka se toimittaa saamaansa asiakastyöhön kolmannelle osapuolelle. Sovelluksen on määrä käyttää verkkosivustoa käyttöliittymänään, jolloin kyseessä on verkkosovellus. Tämä sovellus tulee tarvitsemaan taustajärjestelmää, sillä sen sisältö tulee olemaan käyttäjien luomaa ja moderaattorien tarkastamaa. Työssä tutkitaan CRUD-toimintoja sekä REST-rajapintaa, sillä niiden avulla sovelluksen tarvittavat ominaisuudet pystytään implementoimaan.

Tutkimusongelmana tässä työssä on CRUD-toimintoja suorittavan taustajärjestelmän toteutus hyvien käytäntöjen mukaisesti. Tutkimusongelmasta saadaan johdettua tutkimuskysymykset, jotka ovat:

- Mitä ovat CRUD-toiminnot?
- Kuinka ne implementoidaan REST-rajapintaan?
- Miten taustajärjestelmä tehdään hyvien käytäntöjen mukaan?

Vastaukset tutkimuskysymyksiin saatiin työn suunnittelun ja toteutuksen aikana. CRUD-toimintojen vaiheet ja niiden sisällyttäminen REST-rajapintaan

tutkittiin ja dokumentoitiin tähän työhön. Dokumentoidun työn avulla toimeksiantaja voi hyödyntää tätä työtä ja sen sisältämää lähdekoodia tulevaisuudessa muihin sovelluksiin.

2.1 Tutkimusote

Tutkimus on interventionististä, tarkemmin sanottuna konstruktiivista tutkimusta, sillä se edellyttää ongelman ratkaisemista rakentamalla toimiva malli. Tutkimuksen on kuitenkin myös liityttävä aikaisempaan teoriaan. (Kananen 2017.)

Konstruktiivinen tutkimus keskittyy tosielämän ongelmiin, ja sen avulla pyritään ratkaisemaan niitä. Konstruktio on abstrakti käsite, joka tarkoittaa mitä tahansa ihmisen luomaa kuten suunnitelmia, diagrammeja ja tietojärjestelmämalleja. (Lukka 2001.)

Koska työssä toteutetaan taustajärjestelmä kokonaisuudessaan, siinä hyödynnetään aiheesta kertovaa kirjallisuutta ja internetistä saavissa olevaa tietoa. Tutkimusongelma ratkaistaan luomalla toimiva malli. Työssä perustellaan käytettyjä ratkaisuja, sillä taustajärjestelmä voidaan toteuttaa monilla erilaisilla rajapinnoilla ja tietokannoilla.

2.2 Aineisto ja analyysi

Tarvittava aineisto on PHP-ohjelmointikielestä sekä SQL-kyselykielestä kertovat dokumentaatiot. Niiden lisäksi tarvittiin tietoa REST-arkkitehtuurimallista ja CRUD-toiminnoista. REST-rajapinnan on tarkoitus suorittaa asiakkaan HTTP-pyyntöjen mukaan CRUD-toimintoja tietokannassa, joten MySQL-relaatiotietokannan dokumentaatiot ovat myös tarpeellisia.

Sekundäärisen aineiston kokoaminen alkoi lähteiden keräämisellä. Merkittävimpiä lähteitä olivat käytettävien teknologioiden omat dokumentaatiot (PHP & MySQL). Niiden lisäksi lähteinä käytettiin aiheesta kertovaa kirjallisuutta sekä verkkoartikkeleita, kuin myös ohjelmointia opettavia sivustoja, joita ovat muun muassa GeeksforGeeks ja W3Schools. Nämä opetussivustot tarjoavat kattavasti tietoa PHP-ohjelmointikielestä sekä SQL-kyselykielestä.

Kun lähteinä käytetään paljon internetistä saatavilla olevaa tietoa, on muistettava, että niiden sisältö saattaa muuttua tulevaisuudessa vastaamaan uusia työssä käytettävien teknologioiden ominaisuuksia. Tämä koskee myös kirjallisuutta, jonka sisältämät aiheet voivat vanhentua vuosien saatossa. IT-ala on jatkuvasti kehittyvä ja olemassa olevia teknologioita päivitetään sekä myös täysin uusia teknologioita ilmestyy markkinoille jatkuvasti.

Primääristä aineistoa tehtiin itse suunnittelussa ja toteutuksessa. Se koostuu REST-rajapinnan lähdekoodista, joka sisältää myös CRUD-toiminnot. Nämä rakennettiin WWW-palvelimen päälle, johon myös MySQL-tietokanta luotiin. Kokonaisuus koostuu WWW-palvelimen kansiorakenteesta, rajapinnan skripteistä sekä MySQL-tietokannasta.

2.3 Tutkimuksen tavoite

Tutkimuksen tavoitteena oli selvittää CRUD-toimintojen toteutukseen liittyvät vaiheet sekä mitä niiden implementoinnissa tulee ottaa huomioon REST-rajapinnassa. Tavoitteena lopputulokselle on toimiva dokumentoitu taustajärjestelmä, joka kykenee CRUD-toimintoihin. Se koostuisi kolmesta pääosasta: WWW-palvelimesta, REST-rajapinnasta sekä MySQL-tietokannasta.

Dokumentoidun työn ja taustajärjestelmän on tarkoitus toimia mallina, jota voitaisiin jatkokehittää ja soveltaa myös muihin sovelluksiin. CRUD-toimintoja käytetään aina tietokannan yhteydessä, sillä ilman niitä tietokanta olisi tyhjä. Selkeys niiden toiminnasta yleisesti helpottaa vastaavien taustajärjestelmien kehityksessä.

Tutkimusvaiheessa tutkittiin erilaisia tapoja, kuinka tietokantayhteyksiä voidaan rakentaa käyttämällä PHP-ohjelmointikieltä. PHP on nimittäin saanut useamman eri laajennuksen tietokantayhteyksiä varten sen elinaikanaan ja ne eroavat toisistaan toiminnoillaan. (PHP Connect to MySQL s.a.). Lisäksi tutkittiin myös HTTP-pyyntöjen ja SQL-lauseiden osuutta CRUD-toimintojen implementoinnissa REST-rajapintaan.

3 TEOREETTINEN VIITEKEHYS

Tässä luvussa käydään läpi työhön liittyvää teoriaa. Teoria koostuu käytettävistä työkaluista sekä teknologioista. REST-rajapinnan osalta perehdytään sen toimintamalliin.

Kehitysympäristö, lyhennettynä IDE (Integrated Development Environment) on ohjelmisto, joka on kehitetty varta vasten sovellusten ohjelmoimiseen. Kehitysympäristöjä on lukuisia ja ne tukevat eri ohjelmointikielien syntakseja. Koodia voidaan kirjoittaa periaatteessa millä tahansa tekstieditorilla, mutta niistä puuttuu kehitysympäristöjen tarjoamia ominaisuuksia. Nämä ominaisuudet ovat muun muassa virheiden tunnistaminen (esimerkiksi alleviivauksella) ja niiden korjaukseen tarjottavat ehdotukset. Kehitysympäristön avulla helpotetaan ohjelmointia. Tässä työssä käytetään Visual Studio Code-kehitysympäristöä sekä CPanel-palvelinkäyttöliittymään integroitua tekstieditoria.

PHP on suosittu ja paljon käytetty palvelinohjelmointiin tarkoitettu ohjelmointikieli. Se on dynaamisesti tyypitetty ja tukee HTTP-protokollaa, joka on vaatimus tämän työn toteutukselle. PHP on ohjelmointikielenä vanha ja se on käynyt monen eri version elinaikanaan läpi. Uusien versioiden yhteydessä se on saanut uusia ominaisuuksia sekä sen olemassa olevia ominaisuuksia on paranneltu. PHP on julkaistu ensimmäisen kerran vuonna 1994. (History of PHP s.a.)

PHP:n tärkeimpiin ominaisuuksiin lukeutuu sen kyky upottautua HTML-tiedostoihin. Vaihtoehtoisesti myös HTML-kieltä voidaan sisällyttää PHP-tiedostoihin. Tällä mahdollistetaan dynaamisten verkkosivujen luominen. PHP myös tukee useita eri tietokantaohjelmistoja kuten MySQL-tietokantaa. (PHP Introduction 2024.)

HTTP on lyhenne sanoista Hypertext Transfer Protocol. Se on asiakas-palvelin tyyppinen protokolla, jossa asiakas ja palvelin kommunikoivat viesteillä. Asiakas on usein WWW-selain, mutta se voi olla myös esimerkiksi botti. Viestit tapahtuvat asiakkaan lähettämillä HTTP-pyynnöillä (request) joihin palvelin vastaa (response). (An overview of HTTP s.a.)

Tässä työssä kehitettävän verkkosovelluksen toiminta rakentuu HTTP-protokollaan. Sen avulla käyttäjät lähettävät erityyppisiä HTTP-pyyntöjä REST-rajapinnalle, joka suorittaa CRUD-toimintoja tietokannassa. Rajapinta vastaa asiakkaalle suoritetun toiminnon jälkeen.

MySQL-tietokanta on ilmainen, avoimen lähdekoodin relaatiotietokantaohjelmisto. Nimensä mukaisesti se käyttää SQL-kyselykieltä. SQL-kyselykieli on luotettava, nopea sekä helppokäyttöinen. MySQL-tietokantaa käyttävät monet tunnetut palvelut kuten Facebook, Google sekä Adobe. Se on suosittu tietokantaratkaisu WWW-ohjelmoinnissa, sillä se tarjoaa saumattoman integraation PHP- ja Javascript-ohjelmointikielien kanssa. (What is MySQL? 2024.)

WWW-palvelin on ohjelmisto, jota ajetaan tietokoneella. Palvelimet sijaitsevat usein palvelinsaleissa, joissa on lukuisia tietokoneita. WWW-palvelinta voidaan kuitenkin myös ylläpitää omalla tietokoneella. WWW-palvelimen tehtävänä on vastaanottaa asiakkaan lähettämiä HTTP-pyyntöjä ja vastata niihin toimittamalla pyydetyt dokumentit, joita voivat olla verkkosivut tai muut resurssit. HTTP-pyynnöt voivat olla erityyppisiä riippuen niiden metodeista, joita ovat muun muassa GET, POST, PUT ja DELETE. Niiden avulla rajapinta suorittaa CRUD-toimintoja tietokannassa. Rajapinta sekä tietokanta tarvitsevat WWW-palvelimen toimiakseen. Erilaisia palvelinohjelmistoja on useita, joita ovat muun muassa Apache, Nginx ja Microsoft IIS. (Web-palvelin 2023.)

REST-rajapinta on ohjelmointirajapinta, joka käyttää REST-arkkitehtuurimallia. Sen tarkoituksena on tehdä eri järjestelmien välinen kommunikointi helpommaksi internetissä. REST-rajapinnan kanssa toimivia järjestelmiä kutsutaan usein RESTful-järjestelmiksi, jonka kriteereinä ovat niiden tilattomuus sekä asiakkaan ja palvelimen erottaminen toisistaan. Käytännössä tämä tarkoittaa, että asiakas ja palvelin voidaan kehittää itsenäisinä osina sovellusta eikä niiden missään vaiheessa tarvitsee tietää toisistaan. Tällä tavoin voidaan muokata rajapinnan ohjelmalogiikka ilman, että käyttöliittymän (frontend) logiikkaa joudutaan muokkaamaan. Tilattomuus näkyy myös asiakkaan lähettämissä HTTP-pyynnöissä. Jokaisen pyynnön on sisällettävä kaikki rajapinnan tarvitsevat tiedot kullekin toiminnolle. (What is REST? s.a.)

CRUD-toiminnot kuvaavat neljää eri toimintoa, joilla käsitellään resursseja. Niitä ovat luominen (create), lukeminen (read), päivittäminen (update) sekä poistaminen (delete). Käytännön mukaisesti resurssit nimetään monikossa, esimerkiksi ”posters”. CRUD-paradigmaa käytetään laajasti verkkosovelluksissa, sillä sen avulla saadaan helposti muistettava ja selkeä rakenne, jonka avulla luodaan kokonaisia toimivia malleja. (What is CRUD? s.a.)

REST-rajapinnan yhteydessä nämä neljä CRUD-toimintoa vastaavat eri HTTP-pyyntöjä, joita asiakas lähettää palvelimelle. Niiden avulla taustajärjestelmä pystyy suorittamaan toimintoja tietokannassa. Toimintojen suorittamisen jälkeen palvelin vastaa toiminnon perusteella asiakkaalle.

4 SUUNNITTELU

Tässä luvussa käydään läpi työn tekniseen toteutukseen vaikuttavia asioita ja kuinka taustajärjestelmä tullaan rakentamaan. Lisäksi tarkastellaan taustajärjestelmän toimintaa kaavioita apuna käyttäen.

4.1 Käyttötarkoitus ja käyttökohteet

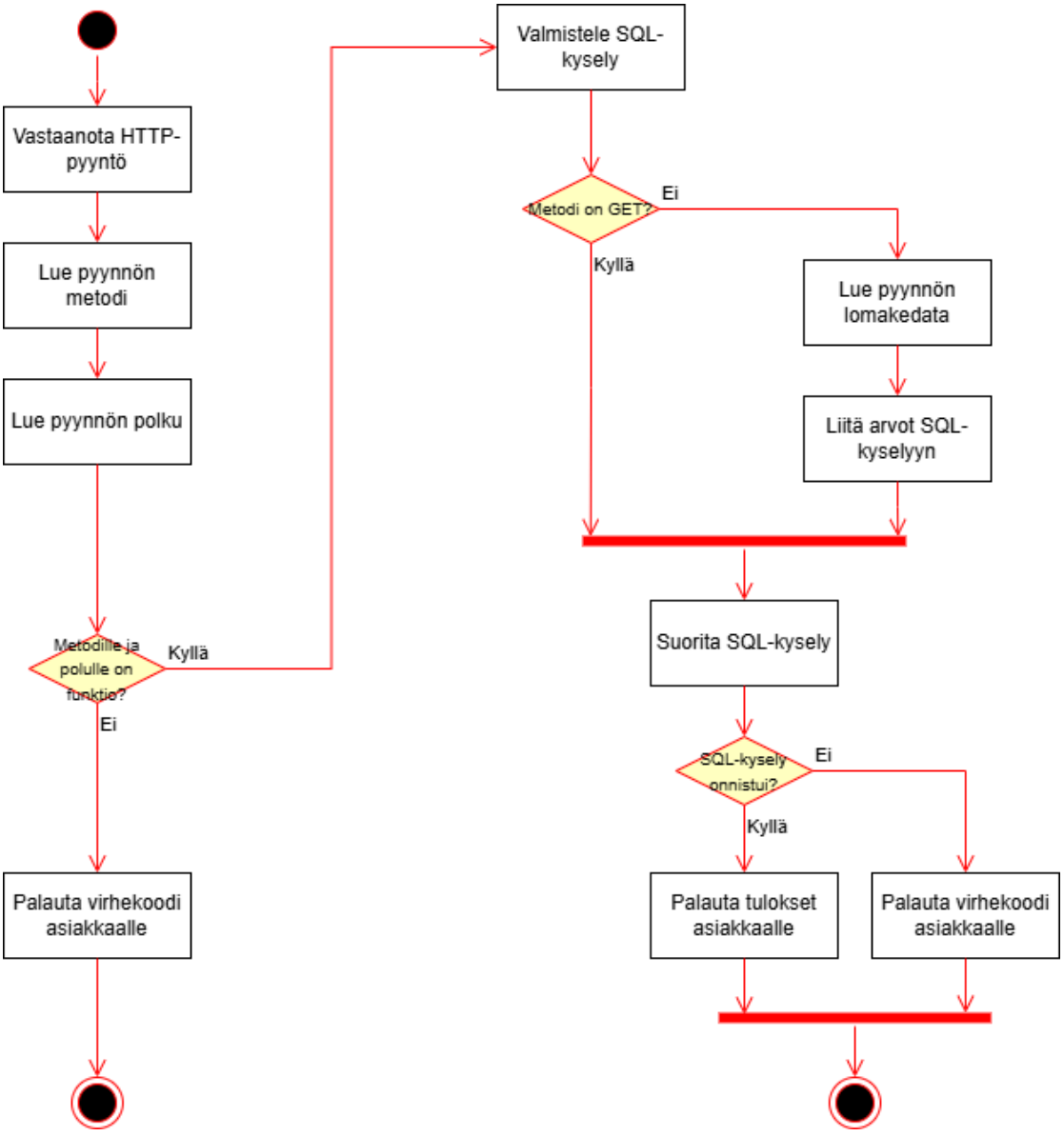
Työssä toteutettava taustajärjestelmä tulee osaksi TeamUp-nimistä verkkosovellusta, joka tehdään toimeksiantajan asiakastyöhön. Sovellusta käytetään verkkosivuston kautta ja sen tarkoituksena on yhdistää yrittäjyydestä kiinnostuneet sekä jo yrittävät opiskelijat toistensa kesken. Sovelluksen toiminta pohjautuu ilmoituksiin, joita käyttäjät voivat lisätä ja selata. Ennen ilmoituksen näkymistä muille, moderaattorin tulee joko julkaista tai poistaa se.

Taustajärjestelmän REST-rajapinta kehitetään puhtaasti PHP-ohjelmointikielellä ilman lisäkirjastoja. Tavoitteena on luoda selkeä ja yksinkertainen järjestelmä. Tämä mahdollistaa vaivattoman jatkokehityksen, sillä työssä toteutettava taustajärjestelmä toimii prototyyppinä.

TeamUp-sovellusta tulee pystyä käyttämään tietokoneella sekä mobiililaitteilla. Verkkosovellus on hyvä ratkaisu, sillä molemmille alustoille on saatavilla WWW-selaimia. WWW-selaimet toimivat tässä sovelluksessa käyttöliittymänä. Tällä vältytään turhalta kehitystyöltä, sillä muutoin tietokoneelle sekä mobiililaitteille täytyisi ohjelmoida omat käyttöliittymäsovellukset.

4.2 Toimintakaavio

Toimintakaavion (kuva 1) avulla visualisoidaan ohjelman kulkua ja toimintoja järjestelmässä. Sillä kuvataan miten eri toiminnot yhdistyvät toisiinsa ja kuinka järjestelmän tila muuttuu. Se alkaa pisteestä ja päättyy pisteeseen, joiden välissä kuvataan ohjelman kulun haarautumista ehtojen avulla. (Activity Diagrams – Unified Modeling Language (UML) 2025.)



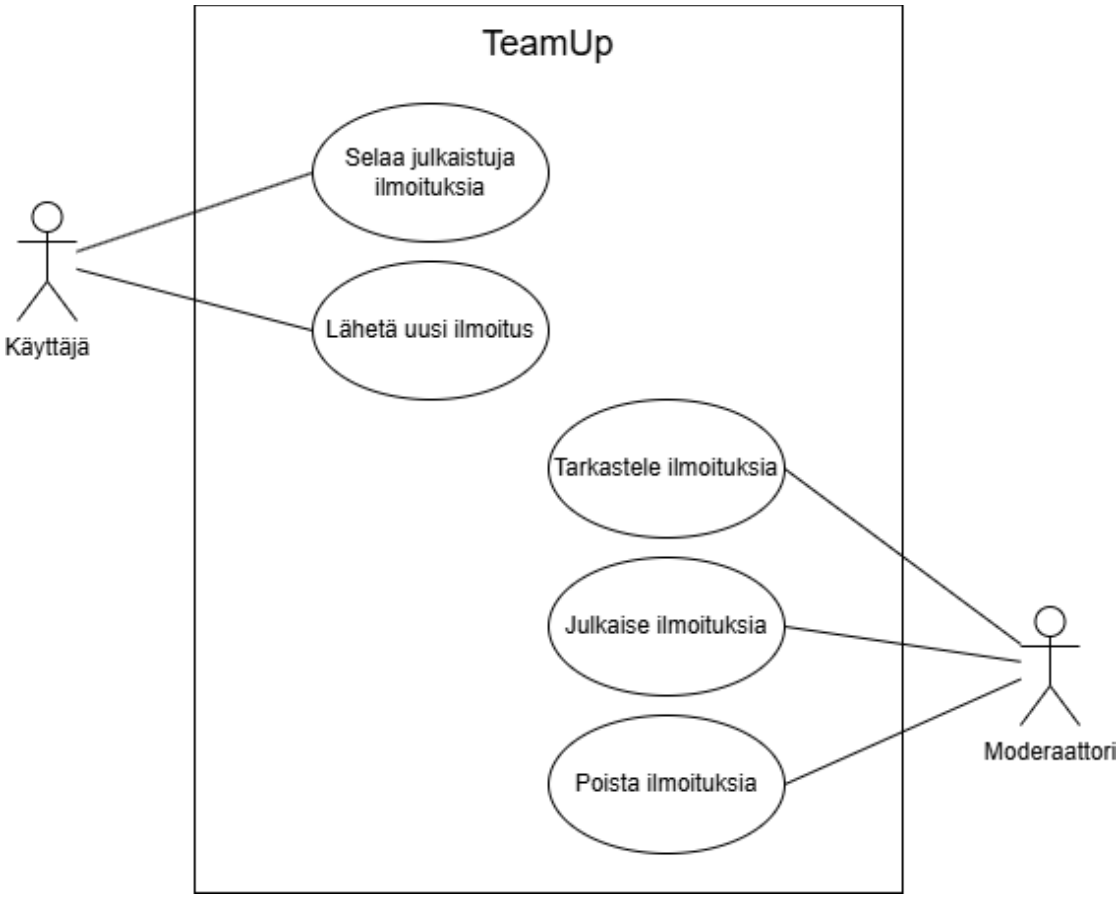
Kuva 1. Rajapinnan toimintakaavio

Kuva 1 näyttää tapahtumaketjun alkaen rajapinnan saamasta HTTP-pyynnöstä päättyen asiakkaalle palautettavaan vastaukseen. Rajapinta käsittelee ensimmäisenä vastaanottamansa HTTP-pyynnön, josta luetaan metodi sekä polku (URL-osoitteen loppuosa). Metodin sekä polun avulla tarkistetaan, onko

niitä vastaavaa funktiota toteutettu. Jos ei ole, rajapinta palauttaa virhekoodin asiakkaalle. Jos funktio taas on toteutettu, rajapinta yrittää suorittaa sen. Funktiossa suoritetaan CRUD-toimintoa vastaava SQL-kysely tietokantaan. Jos metodi ei ole GET-tyyppinen, rajapinta lisää HTTP-pyynnön mukana tulleen lomakedatan avulla SQL-kyselyyn tarvittavat arvot. Viimein SQL-kysely suoritetaan tietokantaan ja jos se onnistuu, palautetaan asiakkaalle kyselyn tulokset. Muussa tapauksessa asiakkaalle palautetaan virhekoodi.

4.3 Käyttötapauskaavio

Käyttötapauskaavion (kuva 2) avulla visualisoidaan toimijoiden tekemiä toimintoja järjestelmässä. Toiminnot kuvataan niin kuin toimijat ne näkevät, eli heidän perspektiivistään. Toimijat kuvataan kaaviossa tikku-ukkoina ja ne voivat olla esimerkiksi käyttäjiä tai toisia järjestelmiä. (Use Case Diagram – Unified Modeling Language (UML) 2025.)



Kuva 2. TeamUp-sovelluksen käyttötapauskaavio

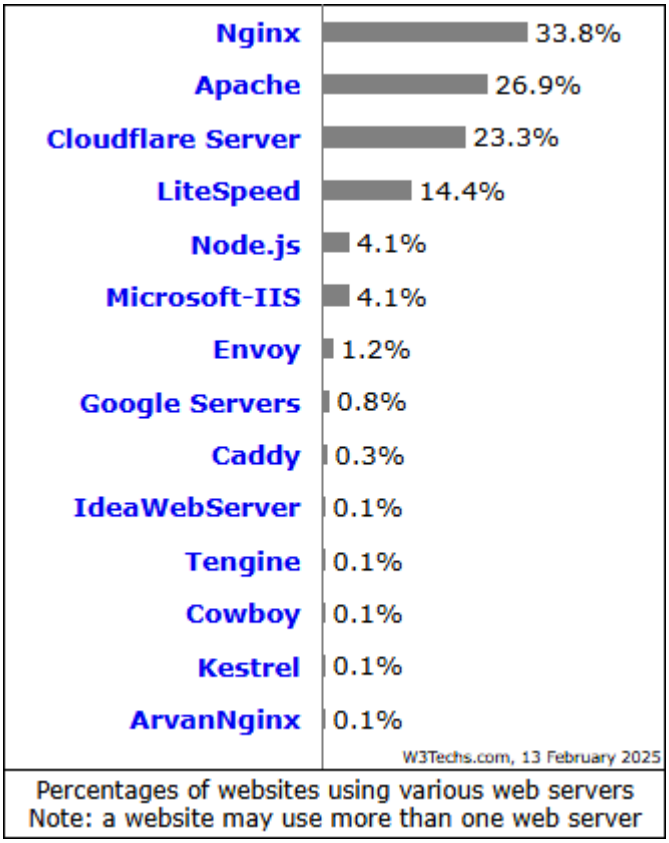
Kuva 2 näyttää, mitä käyttötapauksia sovelluksen toimintaan kuuluu. Käyttäjän on mahdollista lähettää uusia ilmoituksia (jotka ovat piilotettuja) ja selata järjestelmässä olevia jo julkaistuja ilmoituksia. Moderaattori taas pystyy tarkastelemaan kaikkia ilmoituksia, piilotettuja tai julkaistuja. Lisäksi moderaattori pystyy julkaisemaan ilmoituksia sekä poistamaan niitä taustajärjestelmästä.

4.4 WWW-palvelin

Taustajärjestelmän selkäranka on WWW-palvelin. Se on ohjelmisto, joka pyörii tietokoneella, useimmiten palvelinsaleissa. Kun asiakas avaa verkkosivun selaimellaan, selain lähettää HTTP-pyynnön palvelimelle, jossa palvelin käsittelee pyynnön ja palauttaa käyttäjälle oikean dokumentin. Palautettavan dokumentin muoto voi olla staattinen tai dynaaminen. Eri WWW-palvelinohjelmistoja on markkinoilla useampia (kuva 3).

Staattinen dokumentti tarkoittaa tiedostoa, jonka asiakas saa juuri sellaisena kuin se on myös palvelimella. Staattinen sivu voi siis olla esimerkiksi yrityksen kotisivujen etusivu, jossa ei ole tarvetta päivittyvälle, niin sanotulle dynaamiselle sisällölle.

Dynaaminen dokumentti on taas tiedosto, joka generoidaan PHP-skriptillä. Sen sisältö on riippuvainen pyynnöstä sekä siitä, mistä data haetaan, esimerkiksi tietokannasta. Dynaaminen dokumentti on siis muuttuva eikä se ole samanlainen palvelimella kuin miltä se näyttää asiakkaan selaimessa.



Kuva 3. WWW-palvelinten markkinaosuudet (Usage statistics and market shares of web servers. 2025)

Kuva 3 osoittaa eri palvelinohjelmistojen käytön ja suosion helmikuussa 2025. Kuvasta nähdään, että lähes kaikki WWW-palvelimet käyttävät jotain neljästä suosituimmasta palvelimesta, jotka ovat Nginx, Apache, Cloudflare Server sekä LiteSpeed. Näiden neljän palvelimen markkinaosuus on 98,4 %.

Tässä työssä WWW-palvelinohjelmistona toimii LiteSpeed, joka pyörii Domainhotelli-nimisen, suomalaisen yrityksen palvelimilla. Siinä käytetään CPanel-nimistä palvelinkäyttöliittymää, jolla palvelinta ohjataan. Käytettävän palvelun perusteena on, että sitä voidaan käyttää heti, sillä Normogames Oy on vuokrannut sieltä palvelimen omiin tarpeisiinsa. Asiakastyön tilaaja myös haluaa Normogamesin vastaavan palvelimesta.

4.5 Tietokanta

Suunnittelu on tietokannan luomisen ensimmäinen askel. Se suunnitellaan ennen muita komponentteja. On ikävää huomata, että luotu sekä käyttöön otettu tietokanta ei sisälläkään kaikkea tarvittavaa tietoa tai tarvittavia suhteita. (Suehring & Valade 2013, 475.)

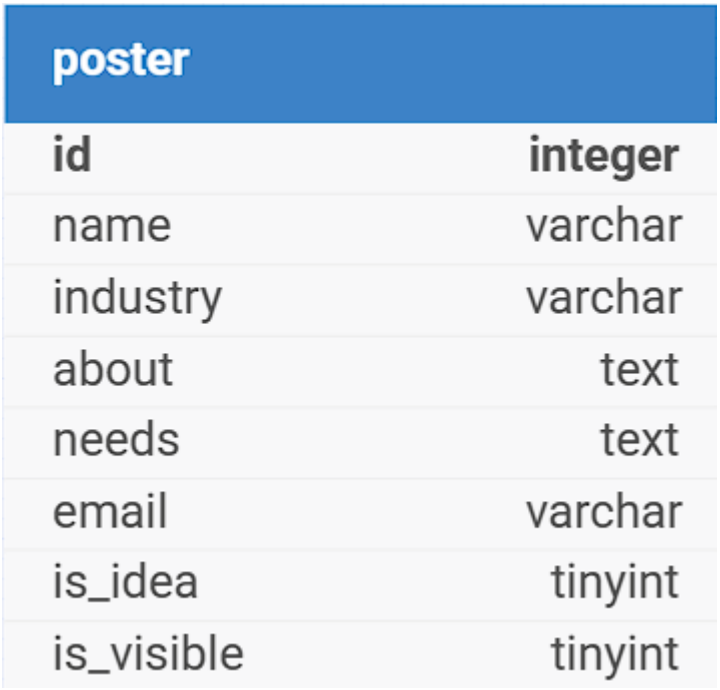
Tässä työssä tietokantaan tullaan tallentamaan kahdenlaisia ilmoituksia käyttäjiltä, jotka ovat joko ideoita (ideas) tai asiantuntijoita (experts). Ideat ovat nimensä mukaisesti ideoita liiketoiminnasta, joiden ilmoittajilta kuitenkin puuttuu tarvittavaa osaamista sen käynnistämiseen tai kasvattamiseen. Asiantuntijat ovat taas henkilöitä, joilla on osaamista, mutta ei yritystä tai tiimiä. Sovelluksen tarkoituksena on yhdistää nämä henkilöt toisiinsa.

Loogista olisi ajatella, että näin ollen tietokantaan suunniteltaisiin näille molemmille omat taulut, joihin tarvittavat tiedot tallennettaisiin, sillä asiantuntijan ei tarvitse kertoa liiketoiminnastaan eikä idean ilmoittajan kertoa osaamisestaan. Kuitenkin voidaan ajatella, että nämä molemmat ovat ilmoituksia, jolloin niille riittää vain yksi taulu, jossa on kentät kuvaukselle (about), tarpeille (needs) sekä onko ilmoitus idea (is_idea).

Sovellukseen ei tule myöskään käyttäjiä tai kirjautumista, sillä se tahdotaan pitää mahdollisimman yksinkertaisena. Näin ollen ilmoittajan tiedot tallennetaan samaan poster-tauluun. Voidaan ajatella, että tämä on väärä tapa toteuttaa tietokanta, mutta kun tutkitaan ilmoituksen luomista tarkemmin, nähdään ettei se olekaan tässä tapauksessa parempi ratkaisu.

Ilmoitusta tehtäessä käyttäjältä kysytään tarpeelliset tiedot sekä koko nimi ja sähköposti. Tämän jälkeen ilmoituksen tiedot tallennettaisiin poster-tauluun ja nimi sekä sähköpostiosoite tallennettaisiin user-tauluun. User ja poster-taulu voitaisiin yhdistää tunnuksen (id-kenttä) avulla. Jatkossa asiakkaan luodessa uuden ilmoituksen, rajapinta voisi tarkistaa, onko tällä nimellä ja sähköpostilla olemassa jo käyttäjää (user) ja luoda ainoastaan uuden rivin poster-tauluun. Tämä on ideaalitilanne, jossa sovellus vielä toimii. Koska oikeassa elämässä tapahtuu virheitä, tämä ei toimi. Jos käyttäjä syöttää nimensä tai sähköpostiosoitteensa väärin (tai niin, että se eroaa aiemmin syötetystä), niin rajapinta luulee, ettei asiakkaalla ole vielä käyttäjää (user) ja luo sen uudelleen. Tästä voidaan päätellä, ettei user-taulun implementoinnilla saavuteta sovelluksen käytettävyyden kannalta mitään hyötyä ja että user-taulun käyttäminen tarvitsisi käyttäjätilien sekä kirjautumisen toteutuksen.

Tällä tavalla toteutettu tietokanta ei ole optimaalisin ratkaisu, mutta se soveltuu hyvin työhön sen ollessa prototyyppi. Rajapinta toteutetaan kuitenkin resurssikeskeisesti, joka mahdollistaa tietokannan muokkaamisen mahdollisimman helposti jatkokehityksen osalta. Poster-taulun suunnittelussa hyödynnettiin DB Designer-nimistä työkalua (kuva 4).



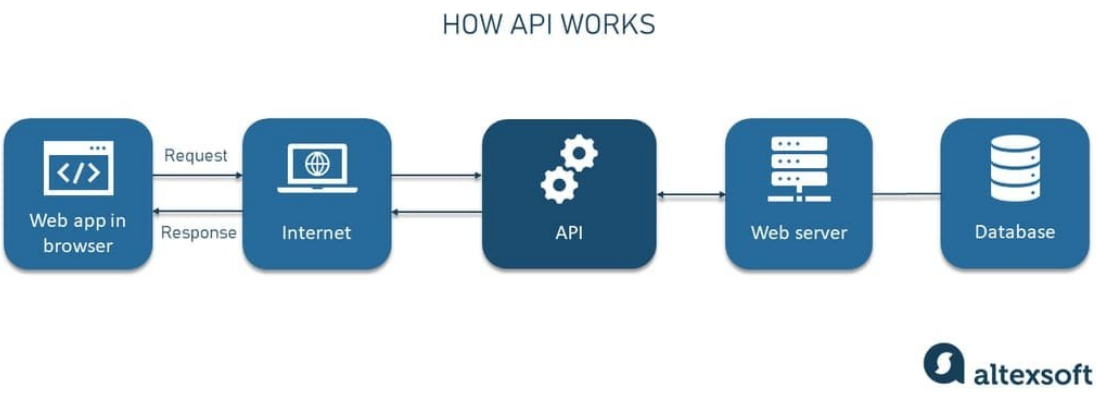
Kuva 4. Poster-taulu DB Designer-työkalussa

Kuva 4 näyttää, suunnitellun poster-tietokantaulun sisällön. Taulun ”id”-kenttää hyödynnetään yksittäisen ilmoituksen lukemisessa, päivittämisessä ja poistamisessa. Suurin osa poster-taulun kentistä on varchar tai text-tyyppisiä, joihin käyttäjien antamat syötteet tallennetaan. Molemmat näistä tyypeistä tukevat pitkiä merkkijonoja, mutta varchar-tyyppisessä kentässä voidaan määrittää sen maksimipituus, esimerkiksi 255 merkkiä. Text-tyyppisen kentän viemä tila tietokannassa on aina 65535 merkkiä. Loput tietokantataulun kentistä (”is_idea” & ”is_visible”) ovat tinyint-tyyppisiä, joilla määritellään ilmoituksen tyyppi (ideas tai experts) sekä onko ilmoitus julkaistu. (MySQL Data Types. s.a.)

4.6 Rajapinta

Jotta sovellus voi kommunikoida tietokannan kanssa, se tarvitsee rajapinnan. Rajapinnan tehtävänä on vastata asiakkaan lähettämiin HTTP-pyyntöihin sekä käsitellä ne ennalta määritellyllä tavalla. Rajapinta toimii sovelluksen osana,

joka sijoittuu asiakkaan ja tietokannan väliin yhdistäen nämä kaksi kokonaisuutta saumattomasti (kuva 5).



Kuva 5. Rajapinnan paikka asiakkaan ja palvelimen välissä (What is API: Definition, Types, Specifications, Documentation. 2024)

Kuva 5 selventää rajapinnan osuutta verkkosovelluksessa. Vaikka rajapinta sijaitseekin WWW-palvelimella, on rajapinta silti se jolle asiakkaan HTTP-pyynnöt ohjataan. Rajapinta toteutetaan käyttäen REST-arkkitehtuurimallia. Sen resurssikeskeinen lähestymistapa soveltuu erinomaisesti tässä työssä kyseessä olevaan verkkosovellukseen, jossa jatkuvasti kommunikoidaan tietokannan kanssa. Rajapinnan toteutus tapahtuu PHP-ohjelmointikielellä, koska se integroituu saumattomasti MySQL-tietokantojen kanssa ja mahdollistaa koodin upottamisen suoraan HTML-sivuille.

4.6.1 Resurssit

Resurssit vastaavat yleisesti tietokannassa olevia tauluja. Jokaiselle resurssille tehdään omat PHP-ohjelmointikielellä luodut CRUD-skriptit. Niiden avulla suoritetaan sovelluksen toiminnan kannalta tarpeellisia toimintoja tietokannassa.

Tässä työssä resurssit kuitenkin vastaavat ilmoitusten eri tyyppejä, joita on kaksi: ideas ja experts. Koska tietokantaan luodaan vain yksi poster-taulu, joka tulee sisältämään kaikki ilmoitukset, lasketaan sekin omaksi resurssikseen. Ajatuksena on, että jatkokehityksessä tietokantaa voitaisiin muuttaa tarpeen mukaan vastaamaan rajapinnan resursseja, jossa esimerkiksi kirjautuneen käyttäjän tunnuksen (id) avulla yhdistettäisiin nämä taulut.

Koska kyseessä on prototyyppi, ei käyttäjiä eikä kirjautumista vaadita sovelluksen käyttöön. Moderaattorille toteutetaan kirjautuminen, jonka on julkaistava ilmoitukset ennen niiden päätymistä julkiseksi. Näin estetään aiheettomat ilmoitukset tai ilmoitukset, joissa on virheitä. Kaikkiin ilmoituksiin vaaditaan koko nimi sekä sähköpostiosoite.

4.6.2 CRUD-toiminnot

CRUD on lyhenne sanoista Create (luo), Read (lue), Update (päivitä) ja Delete (poista). Ne kuvaavat toimintoja, joilla hallitaan resursseja. Se ei kuitenkaan määrittele kuinka toimintoja kutsutaan rajapinnan kautta, vaan sen tehtävänä on määritellä kuinka tietokannassa olevaa dataa hallitaan.

Create on toiminto, jolla tarkoitetaan uuden resurssin luomista. MySQL-kielessä se on muotoa INSERT, jolla syötetään tietokantatauluun uusi rivi. Asiakas kutsuu tätä rajapinnan kautta POST tyyppisellä HTTP-pyynnöllä.

Read on toiminto, jolla tarkoitetaan olemassa olevien resurssien lukemista. MySQL-kielessä se on muotoa SELECT, jolla valitaan kaikki rivit tietokantataulusta valinnaisten ehtojen mukaisesti. Asiakas kutsuu tätä rajapinnan kautta GET tyyppisellä HTTP-pyynnöllä.

Update on toiminto, jolla päivitetään olemassa olevaa resurssia. MySQL-kielessä se on myös muotoa UPDATE, jolla päivitetään olemassa olevan tietokantataulun rivin tietoja. Asiakas kutsuu tätä rajapinnan kautta PUT tyyppisellä HTTP-pyynnöllä.

Delete on toiminto, jolla poistetaan olemassa oleva rivi tietokantataulusta. MySQL-kielessä se on myös muotoa DELETE, jolla poistetaan rivi tietokannan taulusta. Asiakas kutsuu tätä rajapinnan kautta DELETE tyyppisellä HTTP-pyynnöllä.

Näille neljälle eri toiminnolle löytyy vastaavat termit HTTP-protokollan tietopyynnöistä kuten myös SQL-kielen lauseista (taulukko 1). Niiden termit eivät

ole aina samoja keskenään, mutta niillä tarkoitetaan samaa asiaa ja niitä käytetään samassa yhteydessä. Rajapinnassa ne nimetään CRUD-mallin mukaisesti.

Taulukko 1: CRUD-toimintoihin liittyvät termit

HTTP	CRUD	SQL
GET	READ	SELECT
POST	CREATE	INSERT
PUT	UPDATE	UPDATE
DELETE	DELETE	DELETE

Taulukossa 1 on termejä, jotka liittyvät taustajärjestelmässä suoritettaviin CRUD-toimintoihin. HTTP-protokollassa näitä kutsutaan metodeiksi, rajapinnassa CRUD-toiminnoiksi ja SQL-kyselykielessä lauseiksi. Niiden ymmärtäminen on tärkeää, sillä kaikkia niitä tarvitaan, jotta taustajärjestelmä pystyy CRUD-toimintoja tietokannassa suorittamaan.

Tässä työssä toteutetaan tarvittavat CRUD-toiminnot jokaiselle resurssille. Jokainen resurssi ei kuitenkaan vaadi kaikkia neljää toimintoa, johtuen sovelluksen toiminnallisuudesta ja resurssien keskenään jaettavasta tietokannan taulusta. Rajapinta toteuttaa seuraavat CRUD toiminnot resursseittain:

- Posters
    - Read, Update ja Delete
- Ideas
    - Create ja Read
- Experts
    - Create ja Read

Syy näille resurssikohtaisille CRUD-toimintojen eroille on sovelluksen käyttötapauksissa. Käyttäjä pystyy ainoastaan luomaan ideas tai experts-tyyppisen resurssin eikä pelkkää ilmoitusta (poster). Kun moderaattori taas julkaisee tai poistaa ilmoituksen, toiminnan kannalta ei ole väliä, onko ilmoitus ideas tai experts-tyyppiä, jolloin toiminto suoritetaan posters-resurssille.

4.6.3 URL-uudelleenohjaus

URL on lyhenne sanoista Uniform Resource Locator. Sen tarkoituksena on kertoa, missä resurssi sijaitsee tietoverkossa kuten internetissä. Se sisältyy

URI-osoitteeseen, joka taas on lyhenne sanoista Uniform Resource Identifier. URI-osoite muodostuu kolmesta osasta: skeemasta (http/https), omistajasta (verkkotunnus) ja polusta (resurssin sijainti). (URIs s.a.)

URL-uudelleenohjauksen avulla saavutetaan helposti muistettava ja ymmärrettävä rajapinta. URL-uudelleenohjauksella nimensä mukaisesti ohjataan WWW-palvelimelle saapuvat HTTP-pyynnöt haluttuun osoitteeseen. Näitä osoitteita kutsutaan rajapinnan päätepisteiksi. Ne kertovat missä tietty resurssi eli skripti sijaitsee palvelimella. Esimerkkinä voidaan ajatella rajapinnan juurikansion olevan root-niminen. Sen sisällä on api-kansio, missä kaikki skriptit sijaitsevat (taulukko 2).

Taulukko 2: Rajapinnan päätepisteet URL-uudelleenohjauksella ja ilman

Ilman URL-uudelleenohjausta	**URL-uudelleenohjaus**
root/api/addPoster.php?name=new	POST /api/posters
root/api/getPoster.php?id=1	GET /api/posters
root/api/updatePoster.php?id=1&name=updated	PUT /api/posters
root/api/deletePoster.php?id=1	DELETE /api/posters

Taulukossa 2 voidaan nähdä esimerkkien avulla rajapinnan päätepisteiden eroja ilman URL-uudelleenohjausta ja sen kanssa. Taulukon avulla nähdään, miten paljon selkeämpi ja muistettavampi URL-osoitteesta tulee, kun uudelleenohjausta käytetään. Ilman URL-uudelleenohjausta käyttävässä rajapinnassa jokaiselle CRUD-toiminnolle sekä resurssille olisi niiden mukaan nimetty skripti, joka on muotoa ”addPoster.php”. Tällaisessa rajapinnassa HTTP-pyynnön mukana tuleva data sisällytettäisiin URL-osoitteeseen kyselymerkkijonona, joka pidentää URL-osoitetta. URL-uudelleenohjauksella saavutetaan selkeä URL-osoite, joka koostuu ainoastaan api-kansiosta ja siellä olevasta resurssin mukaan nimetystä kansiosta, joka on taulukon esimerkissä ”posters”. CRUD-toiminto suoritetaan HTTP-pyynnön tyypin mukaan (esimerkiksi GET) ja pyynnön yhteydessä saapuva data voi olla JSON-muotoista tai lomakedataa.

4.6.4 Tietokantayhteys

Rajapinnan tulee pystyä yhdistämään tietokantaan, jotta CRUD-toimintoja voidaan suorittaa. Rajapinta ei ymmärrä tietokantaa tai sen käyttämää kyselykieltä, vaan sen tehtävänä on vastaanottaa asiakkaan antama syöte HTTP-pyynnöstä, käsitellä ja sisällyttää se ennalta määritettyihin SQL-lauseisiin. Koska rajapinta toteutettiin PHP-ohjelmointikielellä, on tutkittava sille saatavissa olevia tietokantayhteyden luomiseen tarkoitettuja laajennuksia. Näitä laajennuksia on kaksi, jotka ovat MySQLi ja PDO. Molemmille laajennuksilla voidaan yhdistää MySQL-tietokantaan.

MySQLi on lyhenne sanoista MySQL Improved. Se on kehitetty korvamaan alkuperäinen MySQL-PHP laajennus sekä hyödyntämään MySQL version 4.1.3 tai sitä uudempien versioiden toimintoja. Se sisältyy PHP versioon 5 ja sitä uudempiin versioihin. Se tukee ainoastaan MySQL-tietokantaa.

PDO on lyhenne sanoista PHP Data Objects. Se tarjoaa selkeän ja yksinkertaisen rajapinnan, jonka avulla voidaan yhdistää useisiin eri tietokantoihin mukaan lukien MySQL-tietokanta. Sen varjopuolena on, että se ei tue aivan kaikkia samoja toimintoja, joita MySQLi-laajennus tukee (kuva 6).

Comparison of MySQL API options for PHP

	PHP's mysqli Extension	PDO (Using PDO MySQL Driver and MySQL Native Driver)
PHP version introduced	5.0	5.0
MySQL development status	Active development	Active development
API supports Charsets	Yes	Yes
API supports server-side Prepared Statements	Yes	Yes
API supports client-side Prepared Statements	No	Yes
API supports Stored Procedures	Yes	Yes
API supports Multiple Statements	Yes	Most
Supports all MySQL 4.1+ functionality	Yes	Most

Kuva 6. MySQLi ja PDO-laajennusten ominaisuudet (Overview s.a.)

Kuva 6 näyttää kummankin laajennuksen tuen MySQL-tietokannan ominaisuuksille. Molemmat laajennuksista ovat moderneja toteutuksia ja edelleen aktiivisessa kehityksessä. Tämän työn osalta merkittävin tuki on valmistelluille lauseille (prepared statement), sillä sen avulla estetään SQL-injektio.

SQL-injektio tarkoittaa haavoittuvuutta, jossa sovellus ei tarkista asiakkaan antamaa syötettä, vaan luottaa siihen sokeasti ja tekee kyselyn tietokantaan. SQL-injektion toiminta perustuu sovelluksen valmiiksi määrittämään lauseeseen, johon lisätään asiakkaan antamat arvot. Käytetään pseudokoodia esimerkkinä.

SELECT * FROM poster WHERE id = (asiakkaan syöte).

Yllä oleva lause hakee tietokannasta kaikki rivit poster-taulusta, jossa rivin id-kentän arvo vastaa asiakkaan antamaa arvoa. Oletusarvoisesti ”id” olisi numero yhdestä ylöspäin. Jos asiakas antaa ainoastaan numeron, lause näyttäisi tältä:

SELECT * FROM poster WHERE id = 1.

Todellisuudessa asiakkaaseen ei voida kuitenkaan luottaa. Jos käyttäjä ymmärtää tietokantojen toimintaa ja omaa pahoja intressejä voi hän helposti hyödyntää tätä tietoturva-aukkoa haitallisiin tarkoitusperiin. Käyttäjä pystyy silloin syöttämään lisää SQL-kyselykieltä jo valmiin lauseen sekaan muuttujan ”id” tilalle. Tällöin lause voisi näyttää tältä:

SELECT * FROM poster WHERE id = 1 OR 1 = 1.

Tässä tapauksessa yllä oleva lause palauttaa kaikki rivit, huolimatta siitä onko tietokannan poster-taulussa riviä, jonka id olisi 1. Tämä tapahtuu siksi, että ehto ”OR 1 = 1” on aina tosi. Väärissä käsissä SQL-injektiolla saadaan aikaan ikäviä asioita, varsinkin jos tietokantaan tallennetaan henkilötietoja kuten käyttäjänimiä ja salasanoja.

Tässä työssä toteutetaan tietokantayhteys PDO-laajennuksen avulla. Sen valinta oli selvä, sillä se jakaa lähes samat ominaisuudet kuin MySQLi, mutta tukee useita tietokantoja. Eri tietokantojen tuki voi olla sovelluksen jatkokehityksessä merkittävä asia, joten tuki niille on varma valinta.

5 TOTEUTUS

Tässä luvussa tarkastellaan taustajärjestelmän eri toimintoja tarkemmin hyödyntämällä apuna luotua lähdekoodia. Ensimmäisenä käydään läpi tietokantayhteyden luomiseen liittyvät komponentit, jonka jälkeen perehdytään CRUD-toimintojen toteutukseen.

5.1 Rajapinnan toteutus

Rajapinta koostuu WWW-palvelimella sijaitsevasta api-kansiosta. Api-kansion sisällä sijaitsevat kaikki rajapinnan skriptit sekä resurssien mukaan nimetyt alikansiot (posters, ideas & experts). Näissä kansioissa sijaitsevat CRUD-toimintojen resurssikohtaiset skriptit. PHP-skriptien ohjelmoinnissa on pyritty noudattamaan PHP:n mukaisia nimeämiskäytäntöjä, eli muuttujien sekä funktioiden nimet kirjoitetaan pienillä kirjaimilla, kun taas luokkien nimet kirjoitetaan isoilla alkukirjaimilla (# PHP coding standards s.a).

Näitä skriptejä kutsutaan rajapinnassa päätepisteiden avulla. Tämän takia kansiorakenne on merkittävä, sillä pääpisteet kertovat mistä tarvittava PHP-tiedosto löytyy. Kaikki asiakkaan lähettämät HTTP-pyynnöt ohjataan masterskriptille, josta kutsutaan näitä päätepisteitä, joissa CRUD-toiminnot suoritetaan.

5.1.1 URL-uudelleenohjaus

URL-osoitteen uudelleenohjaus tapahtuu erityisen .htaccess-tiedoston avulla (kuva 7). Tiedostolla ei ole nimeä vaan pelkästään sen tyyppi. Tämän takia on mahdollista, että tiedostoa ei löydä tietokoneen tai WWW-palvelimen resurssienhallinnasta. Muokkaamalla Windowsin tai WWW-palvelimen resurssienhallinnan asetuksia (näytä piilotetut kohteet) .htaccess-tiedosto kuitenkin saadaan näkymään.

```
# Enable the RewriteEngine
RewriteEngine On

# Set the base directory
RewriteBase /matchmaking/

# Redirect requests to the master.php file
RewriteRule ^api/(.*)$ api/master.php [QSA,L]
```

Kuva 7. .htaccess-tiedosto

Kuva 7 näyttää .htaccess-tiedoston sisältämän lähdekoodin. Se käyttää URL-uudelleenohjauksessa rewrite-moduulia, joka on Apache-palvelimen ominaisuus. Sen toiminta tapahtuu seuraavanlaisesti:

1. **RewriteEngine On** lause käynnistää rewrite-moduulin.
2. **RewriteBase** määrittää juurikansion, johon tulevat pyynnöt käsitellään.
3. **RewriteRule** muokkaa kaikki URL-osoitteet muotoon ”api/master.php”, joiden alkuperäiseen URL-osoitteeseen sisältyi ”api/”.

Tässä työssä WWW-palvelimena käytetään LiteSpeed-nimistä palvelinta. Se on Apachen kaltainen palvelin, joka tukee Apachen ominaisuuksia. Tämän ansiosta tässä työssä voidaan hyödyntää .htaccess-tiedostoa sekä rewrite-moduulia. (LiteSpeed Web Server. s.a.)

5.1.2 Config.php

Tietokantayhteyden luominen edellyttää tietokantaan tarvittavia käyttäjätunnuksia. Koska rajapinnan tulee luoda yhteys tietokantaan useammassa eri skriptissä, saavutetaan selkeämpi lopputulos käyttämällä taustajärjestelmän tunnuksien määrittämiseen omaa config-tiedostoa (kuva 8). Jotta näitä tunnuksia voidaan hyödyntää myöhemmin muissa skripteissä, tarvitsee config-tiedosto jatkossa vain sisällyttää muihin skripteihin.

```php
<?php
// Database configuration
define('DB_HOST', '********');     // The hostname of the database server
define('DB_USER', '********');  // The username for accessing the database
define('DB_PASS', '********'); // The password for the database user
define('DB_NAME', '********');  // The name of the database

// Moderator configuration
define('MOD_USER', '********');
define('MOD_PASS', '********');
define('MOD_TIMEOUT', 600);
?>
```

Kuva 8. Config-skripti ja kirjautumistiedot

Kuva 8 näyttää config-tiedoston lähdekoodin (tunnukset ovat sensuroitu). Tietokantaan kirjautumiseen käytettävät tunnukset tallennetaan julkisiin, vakiona pysyviin muuttujiin (public constant) käyttämällä PHP-ohjelmointikielen define-funktiota. Define-funktiossa muuttujalle annetaan nimi sekä sen arvo, jonka on oltava joko numero (integer) tai merkkijono (string). Jatkossa muuttujiin refe-roidaan niiden nimillä. Kaikki muuttujat ovat ”MOD_TIMEOUT”-muuttujaa lu-kuunottamatta merkkijonoja.

5.1.3 Database.php

Kuten config-tiedoston on tarkoitus selkeyttää lähdekoodia estämällä tarpee-ton ohjelmalogiikan kopioiminen muihin skripteihin niin sama pätee myös Da-tabase-luokkaan (kuva 9). Se on apuluokka, jota käytetään muissa skripteissä varsinaisen tietokantayhteyden luomiseen. Sen ainoana tehtävänä on hakea kirjautumiseen tarvittavat tunnukset config-tiedostolta ja luoda tietokantayh-teys.

```php
<?php
require_once 'config.php';

class Database {
    private $host = DB_HOST;
    private $db_name = DB_NAME;
    private $username = DB_USER;
    private $password = DB_PASS;
    public $conn;

    // Get the database connection
    public function getConnection() {
        $this->conn = null;

        try {
            $this->conn = new PDO("mysql:host=" . $this->host . ";dbname=" . $this->db_name, $this->username, $this->password);
            $this->conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
            $this->conn->exec("set names utf8");
        } catch (PDOException $exception) {
            echo "Connection error: " . $exception->getMessage();
        }

        return $this->conn;
    }
}
?>
```

Kuva 9. Database-luokka

Kuva 9 näyttää, kuinka Database-luokka luo tietokantayhteyden MySQL-tietokantaan. Yhdistämisessä käytetään PDO-laajennusta, josta luodaan instanssi, joka tallennetaan julkiseen ”$conn”-muuttujaan. Tätä muuttujaa referoidaan jatkossa muissa skripteissä, joissa CRUD-toiminnot suoritetaan. Mikäli yhteyden luomisessa tapahtuu virhe, taustajärjestelmä palauttaa vikakoodin sisältävän viestin asiakkaalle.

5.1.4 Master.php

Rajapinnan tehtävänä on suorittaa CRUD-toimintoja tietokannassa. Koska sovelluksessa on useampia resursseja, joille näitä toimintoja tarvitsee tehdä, on hyvä selkeyttää ohjelmalogiikan kulkua osiin. Tässä apuna käytetään master-skriptiä (kuva 10), jolle asiakkaan lähettämät HTTP-pyynnöt ohjataan. Master-skripti käsittelee pyynnön tyypin sekä saa oikean resurssin uudelleenohjatusta URL-osoitteesta.

```php
// Get the request method. Override it with _method variable if set (used to imitate PUT/DELETE methods)
$method = $_SERVER['REQUEST_METHOD'];
if ($method == 'POST' && isset($_POST['_method'])) {
    $method = strtoupper($_POST['_method']);
}

// Get the request uri. It looks something like /matchmaking/api/ideas
$uri = $_SERVER['REQUEST_URI'];
$parsedUrl = parse_url($uri); // Extract the query string if it exists (e.g., ?visible=1)

// Remove the /matchmaking/ from the URI
$path = $parsedUrl['path'];
$path = str_replace('/matchmaking', '', $path);

$queryParams = [];
if (isset($parsedUrl['query'])) {
    parse_str($parsedUrl['query'], $queryParams);
}
```

Kuva 10. HTTP-pyynnön käsittely

Kuva 10 näyttää, kuinka asiakkaan lähettämä HTTP-pyyntö käsitellään. Rajapinnalle saapuvat pyynnöt ovat aina joko GET tai POST-tyyppisiä. Tämä johtuu siitä, että asiakas lähettää pyynnöt lomakkeelta, joka ei salli muun tyyppisiä pyyntöjä. Master-skriptissä pyynnön tyyppi saadaan käyttämällä ”$_SERVER[’REQUEST_METHOD’]”-funktiota, jonka antama arvo tallennetaan ”$method”-muuttujaan. Jos pyyntö on POST-tyyppiä, tarkastetaan lisäksi lomakedatasta ”_method”-kentän arvo, joka päällekirjoitetaan alkuperäisen arvon päälle. Tällä tavalla saadaan toteutettua myös PUT ja DELETE-tyyppiset pyynnöt, joita tarvitaan moderaattorin toiminnoille.

HTTP-pyynnön käsittelyn jälkeen suoritetaan varsinainen CRUD-toiminto. Se suoritetaan resurssille, joten rajapinnan tulee tietää mille resurssille HTTP-pyyntö on tarkoitettu. Tämä onnistuu jäsentämällä pyynnön URL-osoite, joka on esimerkiksi muotoa ”matchmaking/api/ideas”. Jäsentäminen tapahtuu käyttämällä ”parse_url”-funktiota, jonka palauttama arvo tallennetaan ”$parsedUrl”-muuttujaan. Tästä muuttujasta saadaan URL-osoitteen polku käyttämällä ensin ”$parsedUrl[’path’]”-funktiota. Polusta saadaan lopulta rajapinnan päätepiste poistamalla siitä osa ”matchmaking/”. Tämän jälkeen päätepisteen avulla suoritetaan oikea koodilohko switch case-funktiossa (kuva 11).

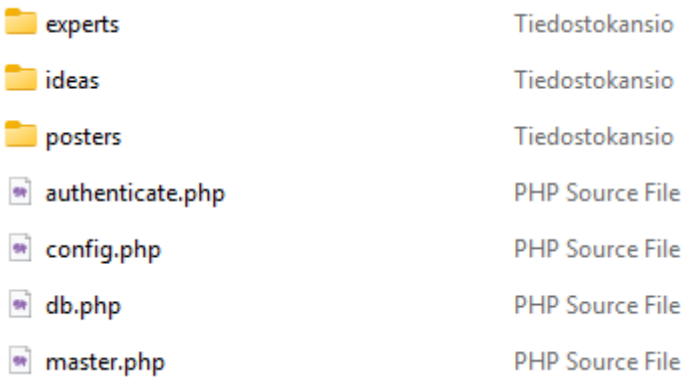
```php
switch (true) {
    // --- MODERATOR LOGIN ---
    // Checking for credentials and redirecting to moderate.php/login.php
    case ($method == 'POST' && $path == '/api/authenticate'):
        require 'authenticate.php';
        break;

    // --- IDEAS ---
    // Getting all ideas with optional visibility filter
    case ($method == 'GET' && $path == '/api/ideas'):
        $show_visible = isset($queryParams['visible']) ? (int)$queryParams['visible'] : null;
        $ideas = require 'ideas/read.php';
        $_SESSION['posters'] = $ideas; // Store results in session
        $_SESSION['filter'] = 'ideas'; // Store filter in session
        header("Location: ../browse.php");
        break;

    // Adding new idea
    case ($method == 'POST' && $path == '/api/ideas'):
        require 'ideas/create.php'; // Redirects to new page
        break;

    // --- EXPERTS ---
    // Getting all experts  with optional visibility filter
    case ($method == 'GET' && $path == '/api/experts'):
        $show_visible = isset($queryParams['visible']) ? (int)$queryParams['visible'] : null;
        $experts = require 'experts/read.php';
        $_SESSION['posters'] = $experts; // Store results in session
        $_SESSION['filter'] = 'experts'; // Store filter in session
        header("Location: ../browse.php");
        break;

    // Adding new expert
    case ($method == 'POST' && $path == '/api/experts'):
        require 'experts/create.php'; // Redirects to new page
        break;

    // --- POSTERS ---
    // Getting all posters with optional visibility filter
    case ($method == 'GET' && $path == '/api/posters'):
        $show_visible = isset($queryParams['visible']) ? (int)$queryParams['visible'] : null;
        $posters = require 'posters/read.php';
        $_SESSION['posters'] = $posters; // Store results in session
        $_SESSION['filter'] = 'all'; // Store filter in session

        if (isset($_SESSION['is_moderating'])) {
            if ($_SESSION['is_moderating']) {
                header("Location: ../moderate.php");
            } else {
                header("Location: ../browse.php");
            }
        } else {
            header("Location: ../browse.php");
        }
        break;

    // Getting one poster with id
    case ($method == 'GET' && preg_match('/\/api\/posters\/[1-9]/', $uri)):
        //Get the id
        $id = basename($uri);
        $poster = require 'posters/read.php';
        $_SESSION['poster'] = $poster; // Store result in session
        header("Location: ../../browse.php");
        break;

    // Updating poster (toggle is_visible)
    case ($method == 'PUT' && preg_match('/\/api\/posters\/[1-9]/', $uri)):
        //Get the id
        $id = basename($uri);
        require 'posters/update.php'; // Redirects to new page
        break;

    // Deleting expert with id
    case ($method == 'DELETE' && preg_match('/\/api\/posters\/[1-9]/', $uri)):
        //Get the id
        $id = basename($uri);
        require 'posters/delete.php'; // Redirects to new page
        break;

    // --- DEFAULT (INVALID REQUEST) ---
    default:
        http_response_code(404);
        echo json_encode(['error' => 'Invalid API request']);
        break;
}
?>
```

Kuva 11. CRUD-toimintojen switch case-funktio

Kuva 11 näyttää, miten päätepisteen ja HTTP-pyynnön tyypin avulla suoritetaan oikea CRUD-toiminto. Toiminnon ja resurssin koodilohkon suorittamisen ehtona käytetään päätepistettä (URL-osoitteen loppuosaa) sekä HTTP-pyynnön tyyppiä. Switch case-funktiossa valintakriteerinä käytetään normaalisti muuttujaa, mutta tässä tapauksessa siinä käytetään boolean arvoa tosi (true). Tämä johtuu siitä, että jokaisen ”case”-koodilohkon ehtona on useampi eri muuttuja. Varsinainen CRUD-toiminto suoritetaan sisällyttämällä resurssin oma CRUD-mallin mukaan nimetty skripti, esimerkiksi ”ideas/read.php”. Toisen PHP-tiedoston sisällyttäminen tapahtuu joko ”require”, ”require_once”, ”include” tai ”include_once”-funktiolla. ”Require” ja ”include”-funktioiden ero on siinä, että ”require”-funktio keskeyttää koodin suorittamisen ja antaa vikakoodin, jos tiedostoa ei löydy. Sitä käytetään master-skriptissä, sillä CRUD-toimintoa ei voida suorittaa, jos tiedoston sisällyttäminen epäonnistuu. (PHP Include Files. s.a.)

5.2 CRUD-toimintojen toteutus

CRUD-toiminnot käyttävät toteutuksessa PHP:n PDO-laajennusta, jossa on tuki prepared statement-funktiolle. Sen avulla suoritetaan tietokantakyselyjä turvallisesti, sillä se estää SQL-injektion. CRUD-toiminnot on toteutettu resursseittain ja ne sijaitsevat jokainen oman resurssin mukaan nimetyssä alikansiossa. Alikansiot sijaitsevat api-kansiossa, jossa kaikki rajapinnan skriptit sijaitsevat (kuva 12).



Kuva 12. Rajapinnan kansiorakenne (api-kansio)

Kuva 12 näyttää rajapinnan kansiorakenteen. Sen nimeäminen on tärkeää, sillä sitä käytetään URL-osoitteen uudelleenohjauksessa master-skriptille

(päätepisteet). Se voidaan nimetä muuksikin kuin api:ksi, mutta nimen olisi hyvä viitata rajapintaan, olla selkeä sekä itsensä selittävä. CRUD-toiminnon vaiheet sovelluksessa tapahtuvat seuraavanlaisesti:

1. Asiakas lähettää HTTP-pyynnön lomakkeelta. Pyynnön tyyppi on joko GET tai POST.
2. URL-osoitteen avulla pyyntö ohjataan master-skriptille.
3. Master-skripti käsittelee pyynnön tyypin ja tarvittaessa päällekirjoittaa sen PUT tai DELETE-tyyppiseksi.
4. Master-skripti käyttää switch case-funktiota, jossa oikean lohkon ehtona toimii päätepiste sekä HTTP-pyynnön tyyppi.
5. Master-skripti suorittaa oikean lohkon, jonka sisällä se kutsuu resurssin omaa CRUD-toiminnon mukaan nimettyä skriptiä (esimerkiksi ideas/create.php).

Yllä oleva lista esittää CRUD-toimintojen vaiheet kronologisessa järjestyksessä. CRUD-toiminto alkaa aina asiakkaan lähettämästä HTTP-pyynnöstä, jonka jälkeen rajapinta käsittelee sen tyypin ja URL-osoitteen. Näiden avulla se osaa suorittaa oikean toiminnon master-skriptissä.

5.2.1 Create-toiminto

Resurssin luominen, eli uuden rivin lisäys tapahtuu käyttämällä ”INSERT” SQL-lausetta. Lisäksi SQL-lauseeseen annetaan tietokantataulun kenttien mukaiset arvot, mutta id-kentälle ei anneta arvoa, sillä se on auto_increment-tyyppinen. Id-kenttä saa siis aina uuden arvon automaattisesti, kun uusi rivi on lisätty tietokantaan. Muiden kenttien arvot saadaan HTTP-pyynnöstä lomakedatana, jotka lisätään valmisteltuun SQL-lauseeseen (kuva 13).

```php
<?php
// Error logging
ini_set('display_errors', 1);
ini_set('display_startup_errors', 1);
error_reporting(E_ALL);

// Get data from the HTML form
$name = $_POST['name'];
$industry = $_POST['industry'];
$about = $_POST['about'];
$needs = $_POST['needs'];
$email = $_POST['email'];

// Hardcoded values for is_idea & is_visible
$is_idea = true;  // True as this is an idea
$is_visible = false;  // False, moderator needs to approve this

// Get database connection
include("db.php");
$db = new Database();
$conn = $db->getConnection();

// Create query and prepared statement to prevent SQL injection
$query = "INSERT INTO `poster` (`name`, `industry`, `about`, `needs`, `email`, `is_idea`, `is_visible`)
                        VALUES (:name, :industry, :about, :needs, :email, :is_idea, :is_visible)";
$stmt = $conn->prepare($query);

// Bind values to parameters
$stmt->bindParam(':name', $name);
$stmt->bindParam(':industry', $industry);
$stmt->bindParam(':about', $about);
$stmt->bindParam(':needs', $needs);
$stmt->bindParam(':email', $email);
$stmt->bindParam(':is_idea', $is_idea);
$stmt->bindParam(':is_visible', $is_visible);

// Execute array
if ($stmt->execute()) {
    header("Location: ../create.php?status=success");
    exit();
} else {
    header("Location: ../create.php?status=failure");
    exit();
}
?>
```

Kuva 13. Resurssin lisääminen tietokantaan

Kuva 13 näyttää ideas-tyyppisen resurssin lisäämisen lähdekoodin. Se käyttää valmisteltua lausetta (prepared statement), johon liitetään arvot, jotka saadaan lomakedatasta. Koska lisäämiseen käytetään POST-tyyppistä HTTP-pyyntöä, käytetään lomakedatan arvojen hakemisessa ”$_POST”-funktiota. Lisäksi kaksi muuta muuttujaa (”$is_idea” & ”$is_visible”) määritellään kovakoodattuina. Ideas-resurssin ollessa kyseessä ”$is_idea”-muuttujan arvo on tosi ja ”$is_visible”-muuttujan arvo on epätosi, sillä ilmoitukset ovat oletuksena piilotettuja, kun ne lisätään tietokantaan.

5.2.2 Read-toiminto

Resurssien lukeminen, eli rivien valitseminen tietokantataulusta tapahtuu käyttämällä ”SELECT” SQL-lausetta (kuva 14). Sen jälkeen annetaan *-merkki, jonka avulla tietokanta osaa palauttaa taulusta kaikkien kenttien arvot. Jos tietokannan halutaan palauttavan vain tiettyjen kenttien arvot, on *-merkin sijasta annettava kenttien tunnukset, esimerkiksi ”name, email”. Lisäksi ”SELECT” SQL-lauseelle voidaan antaa ”WHERE”-ehto, jolla määritellään mitkä kaikki rivit tietokanta palauttaa kyselyn tuloksena.

```php
<?php
// Error logging
ini_set('display_errors', 1);
ini_set('display_startup_errors', 1);
error_reporting(E_ALL);

// Get database connection
include("db.php");
$db = new Database();
$conn = $db->getConnection();

// Create query and check if should show visible/hidden rows
$query = "SELECT * FROM `poster` WHERE is_idea = 1";
if ($show_visible === 1) {
    $query .= " AND is_visible = 1";
} elseif ($show_visible === 0) {
    $query .= " AND is_visible = 0";
}

// Create prepared statement to prevent SQL injection
$stmt = $conn->prepare($query);
$stmt->execute();

// Fetch all results as an associative array
$posters = $stmt->fetchAll(PDO::FETCH_ASSOC);
return $posters;
?>
```

Kuva 14. Ideoiden lukeminen tietokannasta

Kuva 14 näyttää read-toiminnon lähdekoodin idea-resurssille. Tässä käytetään myös valmisteltua lausetta, mutta sen toteutus eroaa create-toiminnosta, sillä käyttäjä ei lukemisen yhteydessä anna mitään arvoja. Tällöin valmisteltuun lauseeseen ei tarvitse liittää kenttien arvoja lainkaan. Valmis SQL-kysely

luodaan ehtolauseella, jossa tarkistetaan, tuleeko sen näyttää julkaistut vai piilotetut ilmoitukset. Tässä käytetään ”$show_visible”-muuttujaa, jonka rajapinta saa URL-osoitteesta kyselymerkkijonona.

5.3 Moderoinnin toteutus

Moderaattoria tarvitaan sovelluksessa käyttäjien tekemien ilmoitusten tarkastamiseen, sillä ne voivat sisältää epäasiallista tai arkaluontoista tietoa. Moderaattori on ainut, jolla on valtuudet päivittää sekä poistaa ilmoituksia, joten se tarvitsee oman kirjautumislogiikan. Näiden kahden CRUD-toiminnon (update & delete) suorittamisen yhteydessä tarkistetaan ensin käyttäjän kirjautuminen moderaattoriksi, jotta muut käyttäjät eivät pysty vahingossa tai tarkoituksella näitä toimintoja tekemään. Tarkistuksilla parannetaan sovelluksen tietoturvaa.

Tarkistamisessa hyödynnetään PHP-ohjelmointikielen sessiota. Siihen voidaan tallentaa muuttujia, jotka säilyvät tausta riippumatta siitä, lataako asiakas toisen verkkosivun. Sessiossa muuttujat säilyvät oletuksena niin kauan kun käyttäjän WWW-selain pysyy päällä. (PHP Sessions. s.a.)

5.3.1 Moderaattorin kirjautuminen

Koska työssä ei käytetä omaa user-taulua tietokannassa, jota käytettäisiin kirjautumiseen, ei tämän takia voida antaa tietyille käyttäjille moderointioikeuksia. Tämän takia moderaattorin kirjautuminen on toteutettu config-tiedostossa määritellyillä muuttujilla (”MOD_USER” & ”MOD_PASS”), jotka ovat samat kaikille moderaattoreille (kuva 15).

```php
<?php
require_once 'config.php';
session_start(); // Start a new session or resume an existing session

// Store values from the html form into variables
$username = $_POST['username'];
$password = $_POST['password'];

// Check for valid moderator credentials
if ($username === MOD_USER && $password === MOD_PASS) {
    // Regenerate session ID to prevent session fixation
    session_regenerate_id(true);

    // Store username and password in session variables
    $_SESSION['username'] = $username;
    $_SESSION['password'] = $password;

    // Store the last activity time
    $_SESSION['last_activity'] = time();

    // Redirect to the moderator.php
    $_SESSION['is_moderating'] = true; // Start moderating
    header("Location: posters"); // Gets all posters and redirects to moderate.php page
    exit();
} else {
    // Failed to login, return to login.php
    $_SESSION['is_moderating'] = false; // Not moderating anymore
    header("Location: ../login.php");
    exit();
}
?>
```

Kuva 15. Moderaattorin kirjautuminen

Kuva 15 näyttää moderaattorin kirjautumiseen käytetyn lähdekoodin authenticate-skriptissä. Kirjautuminen tapahtuu asiakkaan lähettämästä POST-tyyppisestä HTTP-pyynnöstä, jossa on lomakedatana asiakkaan syöttämät käyttäjätunnukset. Nämä saadaan tallennettua muuttujiksi käyttämällä ”$_POST[’username’]” ja ”$_POST[’password’]”-funktioita. Kun asiakkaan syöte on tallennettu muuttujiin, vertaillaan näitä ehtolauseessa config-tiedostosta löytyviin muuttujiin. Jos ehtolause täyttyy, tallennetaan tunnukset, aikaleima sekä boolean arvo ”is_moderating” PHP-sessioon. Nämä tarkistetaan jatkossa aina kun asiakas yrittää suorittaa moderaattorioikeuksia vaativia toimintoja. Aikaleimaa käytetään automaattisen uloskirjautumisen toteutuksessa, jossa on kymmenen minuutin raja.

5.3.2 Moderaattorin toiminnot

Moderaattorilla on erityisiä oikeuksia, mitä muilla käyttäjillä ei ole. Moderaattori on ainut, joka pystyy päivittämään tietokantataulun rivejä sekä myös poista-

maan niitä. CRUD-toiminnoista nämä vastaavat UPDATE sekä DELETE-toimintoja, jotka rajapinnassa käsitellään vastaavina HTTP-pyynnön metodeina (PUT & DELETE). Kun rajapinta on suorittaa tällaisen CRUD-toiminnon, se tarkistaa PHP-sessiosta aikaleiman voimassaolon sekä onko asiakas kirjautunut moderaattoriksi.

5.3.3 Ilmoituksen päivittäminen

Ilmoituksen päivittämistä käytetään tässä työssä sen näkyvyyden päivittämiseen, eli onko ilmoitus julkaistu vai ei. Sille on tietokannan poster-taulussa ”is_visible”-kenttä, joka on tinyint-tyyppiä. Sen arvoksi tallennetaan joko yksi tai nolla, joka vastaa boolean arvoja tosi ja epätosi. Kaikki ilmoitukset ovat oletuksena piilotettuja, jolloin ”is_visible”-kentän arvo on nolla.

Ilmoituksen päivittäminen tapahtuu moderaattoria varten kehitetystä omasta verkkosivusta. Sieltä käsin moderaattori lähettää lomakkeen avulla POST-tyyppisen HTTP-pyynnön rajapinnalle. Pyynnön URL-osoitetta, eli päätepistettä käytetään oikean resurssin päivittämiseen (kuva 16). Päätepiste on tällaisessa pyynnössä muotoa ”api/posters/1”, jossa numero yksi viittaa id-kenttään.

```php
<?php
require_once 'config.php';
session_start(); // Start a new session or resume an existing session

// Session has correct credentials -> check for timeout
if (isset($_SESSION['username']) && isset($_SESSION['password'])) {
    if (isset($_SESSION['last_activity']) && (time() - $_SESSION['last_activity']) > MOD_TIMEOUT) {
        // Session has timed out. Destroy it and return to login.php
        session_unset();
        session_destroy();

        header("Location: ../../login.php");
        exit();
    } else {
        // Update last activity timestamp
        $_SESSION['last_activity'] = time();

        // Get database connection
        include("db.php");
        $db = new Database();
        $conn = $db->getConnection();

        $selectQuery = "SELECT * FROM `poster` WHERE id = :id";
        $stmt = $conn->prepare($selectQuery);

        // Bind the id value to the query
        $stmt->bindParam(':id', $id, PDO::PARAM_INT);

        // Execute the select query
        if ($stmt->execute()) {
            // Fetch the result
            $poster = $stmt->fetch(PDO::FETCH_ASSOC);
            $is_visible = ($poster['is_visible'] == 0) ? true : false;

            $query = "UPDATE `poster` SET is_visible = :is_visible WHERE id = :id";
            $stmt = $conn->prepare($query);

            // Bind values to parameters
            $stmt->bindParam(':is_visible', $is_visible);
            $stmt->bindParam(':id', $id, PDO::PARAM_INT);

            // Execute the query
            if ($stmt->execute()) {
                header("Location: ../../moderate.php?status=success");
                exit();
            } else {
                header("Location: ../../moderate.php?status=failure");
                exit();
            }
        } else {
            header("Location: ../../moderate.php?status=failure");
            exit();
        }
    }
} else {
    // Session has no username nor password. Return to login.php
    header("Location: ../../login.php");
    exit();
}
?>
```

Kuva 16. Päivittämisen ohjelmalogiikka

Kuva 16 näyttää, kuinka tietokantataulun rivejä voidaan päivittää. Siinä käytetään ”UPDATE” SQL-lausetta, jonka lisäksi annetaan tietokantataulun kenttien tunnukset käyttämällä ”SET” SQL-lausetta. Viimeisenä on annettava ”WHERE” ehto, jolla määritellään mille riveille päivitys suoritetaan. Jos ”WHERE” ehdon jättää pois, SQL-kyselyn suorituksessa taulusta päivitetään kaikki rivit. (PHP MySQL Update Data. s.a.).

5.3.4 Ilmoituksen poistaminen

Tietokannasta voidaan poistaa joko yksittäisiä tai useampi rivejä samanaikaisesti, mutta kokonaisen taulun poistaminen kaikkine riveineen on myös mahdollista. Näille on olemassa omat SQL-lauseet: ”DELETE FROM”, jolla poistetaan rivejä sekä ”DROP TABLE”, jolla poistetaan koko taulu. (PHP MySQL Delete Data. s.a.). Tämän sovelluksen osalta tarvetta ei ole kuin rivien poistamiseen, joka sekin kuuluu ainoastaan moderaattorille.

Ilmoituksen poistaminen tapahtuu moderaattorin omilta sivuilta käsin kuten myös ilmoituksen päivittäminen. Sen toimintamalli on hyvin samanlainen päivittämisen kanssa. Oikean rivin poistamiseen käytetään ”id”-kentän arvoa ja se sisällytetään URL-osoitteeseen. Poistamisen URL-osoite noudattaa kaavaa ”api/posters/1”, jossa numero yksi viittaa ”id”-kentän arvoon. Poistamisen HTTP-pyyntö on alun perin POST-tyyppiä, mutta se päällekirjoitetaan PUT-tyyppiseksi master-skriptissä, josta myös delete-skriptiä kutsutaan (kuva 17).

```php
<?php
require_once 'config.php';
session_start(); // Start a new session or resume an existing session

// Session has correct credentials -> check for timeout
if (isset($_SESSION['username']) && isset($_SESSION['password'])) {
    if (isset($_SESSION['last_activity']) && (time() - $_SESSION['last_activity']) > MOD_TIMEOUT) {
        // Session has timed out. Destroy it and return to login.php
        session_unset();
        session_destroy();

        header("Location: ../../login.php");
        exit();
    } else {
        // Update last activity timestamp
        $_SESSION['last_activity'] = time();

        // Get database connection
        include("db.php");
        $db = new Database();
        $conn = $db->getConnection();

        // Create query for deleting the row by 'id'
        $query = "DELETE FROM `poster` WHERE id = :id";
        $stmt = $conn->prepare($query);

        // Bind the id value to the query
        $stmt->bindParam(':id', $id, PDO::PARAM_INT);

        // Execute the query
        if ($stmt->execute()) {
            header("Location: ../../moderate.php?status=success");
            exit();
        } else {
            header("Location: ../../moderate.php?status=failure");
            exit();
        }
    }
} else {
    // Session has no username nor password. Return to login.php
    header("Location: ../../login.php");
    exit();
}
?>
```

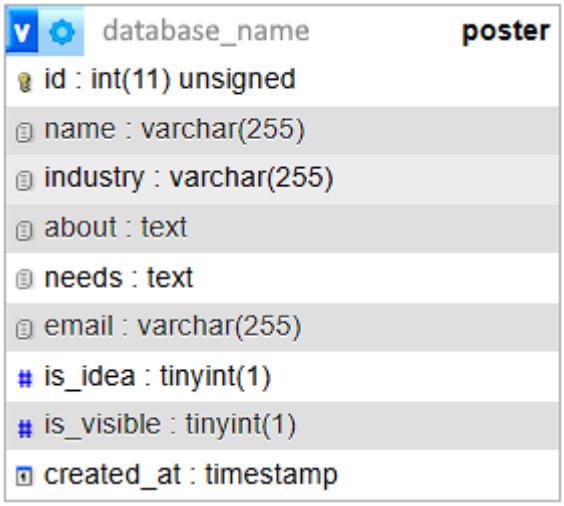
Kuva 17. Poistamisen ohjelmalogiikka

Kuva 17 näyttää, kuinka tietokantataulusta voidaan poista rivi. Poistamiseen käytetään ”DELETE FROM” SQL-lausetta. Sen lisäksi on erittäin suositeltavaa antaa myös ”WHERE”-ehto, jolla määritetään, mikä rivi tai rivit poistetaan. Jos ”WHERE”-ehdon jättää pois, SQL-kyselyn suorituksessa taulusta poistetaan kaikki rivit. (PHP MySQL Delete Data s.a.).

Koska poistaminen on pysyvää, on sen toteutus tehtävä huolellisesti. Tässä sovelluksessa vahingon estämiseksi on poistaminen tehty ainoastaan yhdelle riville kerrallaan ”WHERE”-ehdon ja ”id”-kentän avulla. Käyttöliittymän puolella tämä on huomioitu niin, että ainoastaan ei julkaistut ilmoitukset sisältävät poistamispainikkeen. Tämän toteutukseen käytetään tietokantataulun ”is_visible”-kenttää, joka on tuolloin arvoltaan nolla eli epätosi.

5.4 Tietokannan toteutus

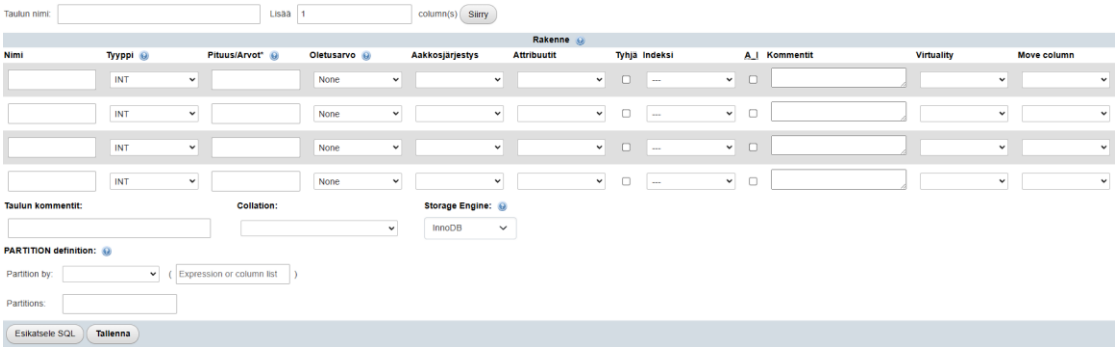
Työhön toteutettu MySQL-tietokanta on hyvin yksinkertainen. Se koostuu yhdestä poster-taulusta, johon tallennetaan ilmoituksen sisältö sekä ilmoittajan yhteystiedot, joita ovat nimi ja sähköposti (kuva 18). Projektin ollessa prototyyppi, ei tietokantaan haluttu tallentaa ylimääräisiä yhteystietoja kuten puhelinnumeroa. Tällä estetään mahdolliset häiriöpuhelut sovelluksen testaajille. Koska sovellukseen ei tehty käyttäjätilejä ei tietokannan tarvitse toiminnallisuuden osalta olla monimutkaisempi.



Kuva 18. Toteutunut poster-taulu

Kuva 18 näyttää toteutuneen poster-taulun rakenteen. Se pysyi pääosin samana kuin mitä oli suunniteltu, mutta siihen lisättiin vielä aikaleima-tyyppinen ”created_at”-kenttä, jonka arvo luodaan automaattisesti MySQL-tietokannassa uuden rivin lisäyksen yhteydessä. Sen avulla sovelluksen käyttöliittymässä (frontend) ilmoitukset voidaan järjestellä aikaleiman avulla. Se myös antaa lisää tietoa ilmoituksesta käyttäjille.

Tietokantataulun luominen tapahtui käyttämällä phpMyAdmin-työkalua, joka on tietokannanhallintatyökalu. Se sisältyy domainhotellin käyttämään CPanel-palvelinkäyttöliittymään. Tietokantatauluja voidaan luoda myös SQL-kyselyillä, mutta visualisoitu luomisprosessi helpottaa sen tekemistä (kuva 19).



Kuva 19. Taulun luominen phpMyAdmin-työkalussa

Kuva 19 näyttää phpMyAdmin-työkalun tietokantataulun luomiseen tarkoitetun sivun. Sen avulla voidaan luoda tietokantataulu visuaalisesti lomaketyyppisellä käyttöliittymällä. Tietokantataulu luodaan silti SQL-kyselyllä, joka rakennetaan käyttäjän lomakkeeseen antamien arvojen mukaan. Tämän kyselyn voi tarkistaa painamalla ”Esikatsele SQL”-näppäintä.

6 TULOKSET

Työn tarkoituksena oli toteuttaa toimeksiantajan asiakastyöhön soveltuva taustajärjestelmäkokonaisuus, joka keskittyy CRUD-toimintojen ympärille. Lopputuloksena työssä syntyi asiakastyön kriteerit täyttävä järjestelmä, joka otettiin heti käyttöön. Siitä jouduttiin rajaamaan joitain ominaisuuksia pois, joiden osalta päädyttiin kompromisseihin. Kompromisseissa tasapainoteltiin käytettävyyden ja hyvien käytäntöjen välillä. Kompromissit koskivat tietokannan suunnittelua, sillä prototyyppiin ei haluttu käyttäjätilejä. Tämän takia tietokannasta pyrittiin tekemään mahdollisimman yksinkertainen, jossa useammat erityyppiset resurssit jakavat saman tietokantataulun.

CRUD-toimintojen kaikkia neljää toimintoa hyödynnettiin työssä. Niiden toteutuksen vaiheet tutkittiin aina asiakkaan lähettämästä HTTP-pyynnöstä tietokantakyselyn suorittamiseen. REST-rajapinta on oleellinen osa CRUD-toimintojen implementoimista, sillä se toimii yhdistävänä tekijänä asiakkaan sekä tietokannan välillä. Rajapinta on myös vastuussa CRUD-toimintojen logiikasta sekä tietokannan turvallisesta hallinnasta.

CRUD-toimintojen vaiheet tutkittiin ja ne koostuvat kolmesta eri osasta: Asiakkaan lähettämästä HTTP-pyynnöstä, Rajapinnan CRUD-toiminnoista sekä SQL-kyselyistä. Kaikki näistä kolmesta eri osasta sisältävät CRUD-toimintojen

suorittamiseen tarvittavat metodit/funktiot, vaikka ne ovatkin nimeltään eriäviä. GET-tyyppinen HTTP-pyyntö vastaa READ-toimintoa, joka taas vastaa SQL kielessä käytettävää SELECT-toimintoa.

7 JOHTOPÄÄTÖKSET

CRUD-toimintoja suorittavan taustajärjestelmän toteutuksessa huolellinen suunnittelu ehkäisee toteutuksen aikana ilmaantuvia haasteita. Vaikka WWW-palvelin on verkkosovelluksen selkäranka, on sen toiminnan kannalta tietokannan rakenne merkittävin yksittäinen asia, joka vaikuttaa sovelluksen käyttämiseen. Rajapintaan ohjelmoitavat skriptit ovat riippuvaisia tietokantarakenteesta (taulujen nimet, kenttien nimet), jotta ne pystyvät CRUD-toimintoja suorittamaan. Tietokantarakenteen muuttaminen jälkikäteen on työlästä, sillä skripteissä tietokannan tauluihin ja kenttiin referoidaan nimellä eikä muuttujalla. Tätä voidaan kuitenkin helpottaa hyvien käytäntöjen mukaisesti toteutetussa rajapinnassa apuluokkien sekä config-tiedostojen avulla.

8 POHDINTA

Työ onnistui verrattain hyvin käyttötapausten toiminnallisuuden toteutuksen osalta. Parannettavaa olisi niiden teknisessä toteutuksessa, jossa olisi hyvä vielä tarkistaa HTTP-pyynnöltä saadut arvot, jotta ne olisivat varmasti samaa tyyppiä kuin mitä tietokantatauluun halutaan tallentaa. Tällä parannettaisiin tietoturvaa, jota tässä työssä käsiteltiin pintapuolisesti SQL-injektion osalta. Tietokantarakenteessa olisi myös parannettavaa jakamalla se useampaan eri tauluun resurssien mukaan, vaikka se tarjoaa vaaditun toiminnallisuuden tuollaisenaan.

Työn kaikki vaiheet pyrittiin dokumentoimaan niin, että vastaavan taustajärjestelmän kehittäminen olisi mahdollista myös muiden toimesta. Valmis taustajärjestelmä vastaa pääosin suunniteltua ja PHP-funktioiden toiminnallisuutta pyrittiin kuvaamaan mahdollisimman tarkasti lähteiden avulla. Työn tavoitteena oli ensisijaisesti valmistaa CRUD-toimintoja suorittava taustajärjestelmä toimeksiantajan asiakastyöhön, mutta myös tarjota ohjeita vastaavaan järjestelmän kehittämiseen toimeksiantajan muihin tarpeisiin. Toimeksiantaja tarvitsee taustajärjestelmiä omissa videopeleissään sekä muissa mahdollisissa tulevaisuuden asiakastöissä.

Teoriassa käsiteltiin työssä käytettyjä teknologioita ja malleja yleisellä tasolla. REST-rajapinnan toiminnallisuus voidaan toteuttaa myös muilla ohjelmointikielillä, mutta tietokannassa suoritettaviin CRUD-toimintoihin liittyvät vaiheet pysyvät samoina riippumatta käytettävästä ohjelmointikielestä tai tietokannasta. REST-arkkitehtuurimallia sekä CRUD-toimintoja käytetään usein yhdessä ja ne soveltuivat myös tämän verkkosovelluksen toteutukseen hyvin.

Työn toteutuksessa pyrittiin huomioimaan jatkokehitys. Rajapinta on kehitetty resurssikeskeisesti sekä PDO-laajennusta käyttämällä mahdollistetaan myös muiden relaatiotietokantojen kuin MySQL-tietokannan yhdistäminen taustajärjestelmään. Tietokantarakenteen suunnittelussa olisi kehitettävää, jotta se vastaisi rajapinnan resursseja ja tukisi käyttäjätilien luomista. Käyttäjätilien toteutuksessa olisi mahdollista jakaa moderointioikeuksia halutuille käyttäjille sen sijaan, että moderoinnille olisi vain yhdet tunnukset, joita kaikki moderaattorit käyttävät. Käyttäjätilien avulla olisi myös mahdollista antaa käyttäjien muokata ilmoituksiaan jälkikäteen, joten käyttäjätilejä tukevan taustajärjestelmän toteutus olisi mahdollinen tutkimuskohde. Rajapinnalle lähetettävien HTTP-pyyntöjen sisältämä data voisi olla myös JSON muotoista lomakedatan sijasta, tosin sillä saavutettavia etuja tai mahdollisia haittoja olisi tutkittava tarkemmin. Tämä myös liittyy vahvasti frontend-kehitykseen, jota tässä työssä ei tutkittu.

LÄHTEET

Activity Diagrams – Unified Modeling Language (UML). 2025. GeeksforGeeks. WWW-dokumentti. Saatavissa: https://www.geeksforgeeks.org/unified-modeling-language-uml-activity-diagrams/ [viitattu 15.3.2025].

An overview of HTTP s.a. MDN Web Docs. WWW-dokumentti. Saatavissa: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview [viitattu 13.3.2025].

History of PHP s.a. PHP. WWW-dokumentti. Saatavissa: https://www.php.net/manual/en/history.php.php [viitattu 13.3.2025].

Kananen, J. 2017. Kehittämistutkimus interventiotutkimuksen muotona: Opas opinnäytetyön ja pro gradun kirjoittajalle. Jyväskylä: Jyväskylän ammattikorkeakoulu. Saatavissa: https://kaakkuri.finna.fi/Record/kaakkuri.222127?sid=4983267769 [viitattu 12.11.2024].

LiteSpeed Web Server s.a. LiteSpeed Technologies. WWW-dokumentti. Saatavissa: https://www.litespeedtech.com/products/litespeed-web-server [viitattu 13.3.2025].

Lukka, K. 2001. Konstruktiivinen tutkimusote. Metodix Oy. WWW-dokumentti. Saatavissa: https://metodix.fi/2014/05/19/lukka-konstruktiivinen-tutkimusote/ [viitattu 12.2.2025].

Mikä on Lidl Plus? s.a. Lidl Ky. WWW-dokumentti. Saatavissa: https://www.lidl.fi/c/mikae-on-lidl-plus/s10021504 [viitattu 12.11.2024].

MySQL Data Types s.a. W3Schools. WWW-dokumentti. Saatavissa: https://www.w3schools.com/mysql/mysql_datatypes.asp [viitattu 14.3.2025].

Normogames Oy s.a. Games. WWW-dokumentti. Saatavissa: https://www.normogames.com/games.html [viitattu 13.11.2024].

Overview s.a. PHP. WWW-dokumentti. Saatavissa: https://www.php.net/manual/en/mysqli.overview.php [viitattu 17.3.2025].

PHP Connect to MySQL. s.a. W3Schools. WWW-dokumentti. Saatavissa: https://www.w3schools.com/php/php_mysql_connect.asp [viitattu 13.11.2024].

PHP Include Files. s.a. W3Schools. WWW-dokumentti. Saatavissa: https://www.w3schools.com/pHp/php_includes.asp [viitattu 13.3.2025].

PHP Introduction. 2024. GeeksforGeeks. WWW-dokumentti. Saatavissa: https://www.geeksforgeeks.org/php-introduction/ [viitattu 13.11.2024].

PHP MySQL Delete Data. s.a. W3Schools. WWW-dokumentti. Saatavissa: https://www.w3schools.com/Php/php_mysql_delete.asp [viitattu 12.3.2025].

PHP MySQL Update Data. s.a. W3Schools. WWW-dokumentti. Saatavissa: https://www.w3schools.com/php/php_mysql_update.asp [viitattu 13.3.2025].

PHP Sessions. s.a. W3Schools. WWW-dokumentti. Saatavissa: https://www.w3schools.com/php/php_sessions.asp [viitattu 14.3.2025].

Suehring, S. & Valade, J. 2013. PHP, MySQL, JavaScript and HTML5 All-In-One for Dummies. Hoboken: John Wiley & Sons, Inc. E-kirja. Saatavissa: https://kaakkuri.finna.fi/Record/nelli29_mamk.2560000000100702?sid=4915911228 [viitattu 27.1.2025].

Usage statistics and market shares of web servers. 2025. W3Techs. WWW-dokumentti. Saatavissa: https://w3techs.com/technologies/overview/web_server [viitattu 17.3.2025].

Use Case Diagram – Unified Modeling Language (UML). 2025. GeeksforGeeks. WWW-dokumentti. Saatavissa: https://www.geeksforgeeks.org/use-case-diagram/ [viitattu 15.3.2025].

URIs s.a. MDN Web Docs. WWW-dokumentti. Saatavissa: https://developer.mozilla.org/en-US/docs/Web/URI [viitatttu 14.3.2025].

Web-palvelin. 2023. VSM-box. WWW-dokumentti. Saatavissa: https://vsm-box.com/2023/07/29/web-palvelin/ [viitattu 13.11.2024].

What is API: Definition, Types, Specifications, Documentation. 2024. Altex-Soft. Blogi. Saatavissa: https://www.altexsoft.com/blog/what-is-api-definition-types-specifications-documentation/ [viitattu 17.3.2025].

What is CRUD? s.a. Codecademy. WWW-dokumentti. Saatavissa: https://www.codecademy.com/article/what-is-crud [viitattu 13.11.2024].

What is MySQL? 2024. GeeksforGeeks. WWW-dokumentti. Saatavissa: https://www.geeksforgeeks.org/what-is-mysql/ [viitattu 13.11.2024].

What is REST? s.a. Codecademy. WWW-dokumentti. Saatavissa: https://www.codecademy.com/article/what-is-rest [viitattu 13.11.2024].

PHP coding standards s.a. PHP. WWW-dokumentti. Saatavissa: https://raw.githubusercontent.com/php/php-src/master/CODING_STANDARDS.md [viitattu 14.3.2025].