jamk

Sovellus työpajatyöskentelyn pelillistämiseen

Aapo Väre

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

jamk | Jyväskylän ammattikorkeakoulu
University of Applied Sciences

jamk

Kuvailulehti

Väre, Aapo

Sovellus työpajatyöskentelyn pelillistämiseen

Jyväskylä: Jyväskylän ammattikorkeakoulu. Toukokuu 2024, 52 sivua

Tieto- ja viestintätekniikan tutkinto-ohjelma (AMK). Opinnäytetyö AMK.

Julkaisun kieli: suomi

Julkaisulupa avoimessa verkossa: kyllä

Tiivistelmä

Tavoitteena oli kehittää pelillistetty sovellus työpajatyöskentelyn fasilitointiin. Toimeksiantajana oli Digiaargh, joka sivutoimintanaan järjestää asiakkaille työpajoja. Toimeksiantajan toiveena oli kustomoitu sovellus ominaisuuksilla, jotka innostaisivat ja motivoisivat työpajan osallistujia heidän työskennellessään.

Toteutus tehtiin käyttäen Pythonilla ja Flaskilla kehitettyä backendia, joka reitittää frontendista saadun datan ja lisää sen tietokantaan, noutaa tietoa tietokannasta ja tarjoaa sen esitettäväksi frontendissa ja noutaa työpajassa esiintyvät kysymykset ulkoisesta tiedostosta ja tarjoaa ne esitettäväksi frontendissa. Tietokanta kehitettiin MySQL-teknologialla. Frontend kehitettiin käyttäen JavaScriptin Vue.js-sovelluskehystä. Frontendiin kehitettiin useita uudelleenkäytettäviä komponentteja, joista jokaisella on oma toiminnallisuutensa. Osa komponenteista tehtiin esittämään kysymyksiä käyttäjälle joihin käyttäjä voi vastata ja nämä vastaukset lähetetään sitten backendin kautta tietokantaan. Osa komponenteista tehtiin datan tietokannasta hakua varten, kuten esimerkiksi käyttäjätietoja, jotka noutamisen jälkeen esitetään käyttäjälle. Projekti kontitettiin lopussa Dockerin avulla kahteen konttiin, toinen sisältäen backendin ja toinen frontendin.

Tavoitteet toteutuivat osittain. Varsinainen pelillistäminen jäi vähäiseksi ja sovellus ei valmistunut käyttövalmiksi. Sovellus jäi ”proof of concept”-tasolle. Valmistuneita komponentteja voidaan jatkokehityksessä käyttää suoraan.

Avainsanat (asiasanat)

Python, Flask, MySQL, JavaScript, Vue, Docker, Fullstack

Muut tiedot (salassa pidettävät liitteet)

jamk

Description

Väre, Aapo

Application for gamification of workshop work

Jyväskylä: JAMK University of Applied Sciences, May 2024, 52 pages

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

Permission for open access publication: Yes

Language of publication: Finnish

Abstract

The objective was to develop a gamified application for facilitating workshop work. The client was Digi-aargh, a company that organises workshops for clients as a side activity. The client wanted a customized application with features that would inspire and motivate the workshop participants while they were working.

The implementation was done using a backend developed in Python and Flask that routes data from the frontend and adds it to the database, retrieves data from the database and provides it for presentation in the frontend, and retrieves workshop questions from an external file and provides them for presentation in the frontend. The database was developed using MySQL technology. The frontend was developed using JavaScript's Vue.js framework. Several reusable components were developed for the frontend, each with its own functionality. Some of the components were designed to ask questions to the user, which the user can answer, and these answers are then sent to the database via the backend. Some components were made to retrieve data from the database, such as user data, which after retrieval is presented to the user. The project was containerized at the end using Docker into two containers, one containing the backend and the other the frontend.

The objectives were partially met. The actual gamification was limited, and the application was not ready for use. The application remained at the "proof of concept" level. The completed components can be used directly in further development.

Keywords/tags (subjects)

Python, Flask, MySQL, JavaScript, Vue, Docker, Fullstack

Miscellaneous (Confidential information)

Sisältö

Kuviot

1 Johdanto

1.1 Opinnäytetyön tausta ja tavoitteet

Nykyajan alati muuttuvassa maailmassa yrityksillä on jatkuva tarve kouluttaa henkilöstöään ja kehittää heidän osaamistaan mukautuakseen markkinoiden jatkuviin muutoksiin. Tämä tarve kattaa laajan kirjon osa-alueita, alkaen teknisistä eli ”kovista” taidoista, kuten uusien ohjelmistojen ja työkalujen käytön opettelusta, aina pehmeämpiin taitoihin, kuten tehokkaaseen kommunikointiin, tiimityöskentelyyn, johtajuuteen, asiakaspalveluun ja ongelmanratkaisukykyyn. Teknisten taitojen hallinta on usein välttämätöntä päivittäisten työtehtävien suorittamiseksi tehokkaasti, kun taas pehmeät taidot parantavat yhteistyötä ja työn sujuvuutta sekä edistävät positiivista työympäristöä. Yksi tehokas tapa järjestää henkilöstökoulutusta on järjestää johonkin tiettyyn osa-alueeseen painottuvia työpajoja. Työpajatyöskentelyssä, joka voi kestää yhdestä päivästä useampaan päivään, osallistujat työskentelevät tiiviisti työpajan vetäjän johdolla. Vetäjä on usein asiantuntija tai kouluttaja, joka ohjaa osallistujia harjoitusten, keskustelujen ja käytännön tehtävien kautta. Tämä vuorovaikutteinen ja osallistava oppimistapa mahdollistaa syvällisemmän ymmärryksen ja taitojen nopeamman omaksumisen.

Opinnäytetyön toimeksiantaja Digiaargh järjestää sivutoimintana yritysasiakkailleen monipuolisia ja räätälöityjä työpajoja. Nämä työpajat voivat keskittyä erilaisiin tavoitteisiin riippuen yrityksen tarpeista ja tavoitteista. Esimerkiksi jotkin työpajat voivat keskittyä työyhteisön kommunikaation parantamiseen, jossa tavoitteena on kehittää tehokkaampia viestintästrategioita ja parantaa tiimien välistä yhteistyötä. Toiset työpajat taas voivat painottaa työyhteisön ihmissuhteiden lähentämistä, mikä voi sisältää tiimien keskinäisen luottamuksen vahvistamista, konfliktien ratkaisua ja yhteishengen kohottamista.

Opinnäytetyön tavoitteena oli kehittää sovellus, jota Digiaargh voi jatkossa käyttää työpajojen fasilitointiin. Keskeisimpinä ominaisuuksina sovellukseen haluttiin kysymyslomakkeita, joihin osallistujat vastaavat omilla tietokoneillaan heidän vastaustensa datan tallentuessa tietokantaan. Nämä lomakkeet sisältäisivät myös kysymyksiä, joissa osallistujat pääsisivät vastaamaan luovan kirjoittamisen keinoin, joko lyhyemmillä tekstivastauksilla tai pidemmillä tarinoilla. Tätä dataa tahdottiin myös pelillistää osallistujien motivoimiseksi ja innostamiseksi. Yksi esimerkki tästä pelillistämisestä

on datan visualisointi erilaisilla kaavioilla. Muita haluttuja ominaisuuksia olivat muun muassa mahdollisuus äänestää muiden käyttäjien vastauksia ja salasanasuojaus, jotta ulkopuoliset eivät pysty pääsemään sovellukseen käsiksi. Projektin suunnitteluvaiheessa päädyttiin fullstack-ratkaisuun, missä projekti koostuisi tietokannasta, backendista sekä frontendistä.

1.2 Digiaargh Oy

Digiaargh on jyväskyläläinen vuonna 2021 perustettu digitoimisto, joka vuodesta 2024 eteenpäin on siirtynyt Tampereelle. Digiaarghin toimialaan kuuluvat WordPress-verkkosivut ja WooCommerce-verkkokaupat. Digiaargh tarjoaa myös viestinnän tukea ja tekstisisältöjä, kuten verkkosivutekstejä ja blogikirjoituksia. Vuonna 2023 yrityksellä oli 2 työntekijää ja liikevaihto oli 30 tuhatta euroa.

2 Tutkimusasetelma

Työn toimeksiantaja Digiaargh toivoo heidän tarpeisiinsa kustomoitua ohjelmaa, jolla he voivat järjestää entistä innovatiivisempia ja osallistujille kiinnostavampia työpajoja. Opinnäytetyön tavoitteena on kehittää sovellus tähän tarkoitukseen. Oleellista on kustomisaatio toimeksiantajan tarpeisiin, joita ei pystytä jo olemassa olevilla ohjelmilla ja sovelluksilla toteuttamaan täysin halutulla tavalla. Projekti on fullstack-sovelluskehitystä, johon kuuluu tietokannan suunnittelu ja toteutus, backendin suunnittely ja toteutus, frontendin suunnittelu ja toteutus sekä kontitus lopussa Dockerilla. Oleellista on myös työskentelyn pelillistäminen käyttäjien motivaation lisäämiseksi.

Tutkimuskysymykset:

1. Miten dataa voidaan pelillistää?
2. Millaisia ominaisuuksia työpajasovellukseen tarvitaan?
3. Mikä motivoi käyttäjiä?

Näihin kysymyksiin pyritään vastaamaan käytännön kehitystyön avulla. Tutkimuskysymykset mahdollistavat syvemmän ajattelun valmiin projektin ulkonäöstä etenkin käyttäjien näkökulmasta.

Opinnäytetyön teossa käytettiin OpenAI:n ChatGPT-tekoälyä apuna. Tekoälyllä tuettiin työn suunnittelua ja toteutuksen kehitystä.

3 Alusta ja ympäristö

3.1 Tietokanta

Sovelluksen toteutukseen valittiin tietokannan teknologiaksi MySQL. MySQL on avoimen lähdekoodin relaatiotietokannan hallintajärjestelmä, joka saa nimensä yhdistelmänä "My" - yhteisperustaja Michael Wideniuksen tyttären My nimestä - ja "SQL" - lyhenne Structured Query Languagesta. Relaatiotietokanta organisoituu yhteen tai useampaan tietotauluun, joissa tiedot voivat olla keskinäisessä yhteydessä, mikä auttaa tietojen järjestämisessä. SQL on kieli, jota ohjelmoijat käyttävät tietojen luomiseen, muokkaamiseen ja hakemiseen relaatiotietokannasta sekä käyttäjien pääsyn valvontaan. Lisäksi MySQL-tyyppinen RDBMS-tietokannanhallintajärjestelmä toimii yhdessä käyttöjärjestelmän kanssa relaatiotietokannan toteuttamiseksi tietokoneen tallennusjärjestelmässä. Se hallinnoi käyttäjiä, mahdollistaa verkkokäytön ja helpottaa tietokannan eheyden testaamista sekä varmuuskopioinnin luomista. (MySQL 8.0 Reference Manual, 2024)

MySQL:n loi ruotsalainen yritys MySQL AB, jonka perustivat ruotsalaiset David Axmark, Allan Larsson ja suomalainen Michael Widenius. Widenius ja Axmark aloittivat MySQL:n alkuperäisen kehittämisen vuonna 1994. MySQL:n ensimmäinen versio ilmestyi 23. toukokuuta 1995. (MySQL 8.0 Reference Manual, 2024)

3.2 Backend

3.2.1 Python

Python on monipuolinen korkean tason ohjelmointikieli, jonka suunnittelufilosofia korostaa koodin luettavuutta merkittävän sisennyksen avulla. (Kuhlman 2012)

Python on dynaamisesti tyypitetty kieli ja sisältää automaattisen roskienkeruun sekä tukee erilaisia ohjelmointiparadigmoja, kuten strukturoitua (etenkin proseduraalista), oliopohjaista ja funktionaalista ohjelmointia (General Python FAQ). Sen laajamittaisen standardikirjaston ansiosta sitä kutsutaan usein "patterit mukana" -kieleksi. (Kuchling 2000)

Guido van Rossum aloitti Pythonin kehittämisen 1980-luvun lopulla ABC-ohjelmointikielen seuraajaksi (General Python FAQ). Ensimmäisen version, Python 0.9.0, hän julkaisi vuonna 1991. Python 2.0 julkaistiin vuonna 2000. Vuonna 2008 julkaistu Python 3.0 toi mukanaan merkittäviä uudistuksia ja ei ollut täysin taaksepäin yhteensopiva aiempien versioiden kanssa (Rossum 2009). Python 2:n viimeinen versio, Python 2.7.18, julkaistiin vuonna 2020 (Peterson 2020).

Python säilyy jatkuvasti yhtenä suosituimmista ohjelmointikielistä ja sillä on laajaa käyttöä erityisesti koneoppimisessa.

3.2.2 Flask

Flask on Pythonilla kirjoitettu mikroverkkokehys, julkaistu alun perin vuonna 2010. Sitä pidetään mikrokehyksenä, koska sen käyttö ei vaadi erityisiä työkaluja tai kirjastoja. Siinä ei sisälly tietokannan abstraktiokerrosta, lomakkeiden validointia tai muita komponentteja, joita löytyy yleensä kolmannen osapuolen kirjastoista. Sen sijaan Flask tarjoaa tukensa laajennuksille, jotka voivat lisätä sovelluksen ominaisuuksia ikään kuin ne olisivat osa Flaskia itsessään. Näihin laajennuksiin kuuluvat muun muassa oliorelationaaliset kartoittajat, lomakkeiden validointi, tiedostojen käsittely, erilaiset avoimet todennustekniikat ja monia muita yleisiä kehykseen liittyviä työkaluja. (Flask Documentation)

Flaskin loi Pocoo-yhteisön jäsen Armin Ronacher. Pocoo oli kansainvälinen Python-harrastajien ryhmä, perustettu vuonna 2004 (Pocoo.org). Ronacherin mukaan Flaskin idea oli alun perin aprillipila, mutta se sai niin suuren suosion, että siitä kehitettiin vakavasti otettava sovellus. Nimen valinnassa leikittiin aiemman Bottle-kehyksen kanssa (Ronacher 2011). Huhtikuussa 2016 Flaskin sekä siihen liittyvien kirjastojen kehitys siirtyi vastaperustetulle Pallets-projektille (Ronacher 2016).

3.3 Frontend

3.3.1 JavaScript

JavaScript toimii ohjelmointikielenä World Wide Webin keskeisten tekniikoiden HTML:n ja CSS:n rinnalla. Vuodesta 2024 lähtien 98,9 prosenttia verkkosivustoista hyödyntää JavaScriptiä asiakaspuolella verkkosivujen käyttäytymisen ohjaamiseen, usein myös kolmannen osapuolen kirjastojen

avulla (Usage statistics of JavaScript as client-side programming language on websites, 2024). Yleisimmissä verkkoselaimissa on oma JavaScript-moottori, joka suorittaa koodin käyttäjien laitteilla.

JavaScript on korkean tason kieli, joka usein käyttää ajonaikaista kääntämistä, ja se noudattaa ECMAScript-standardia. Kieli tarjoaa dynaamisen tyypityksen, prototyyppipohjaisen oliosuuntautuneisuuden ja ensimmäisen luokan funktiot. Se on moniparadigmaattinen ja tukee tapahtumapohjaista, funktionaalista ja imperatiivista ohjelmointityyliä. Lisäksi JavaScript sisältää sovellusohjelmointirajapinnat (API) tekstiin, päivämääriin, säännöllisiin lausekkeisiin, vakiomuotoisiin tietorakenteisiin ja DOM-malliin (Document Object Model). (JavaScript documentation, 2024)

Vaikka ECMAScript-standardi ei sisällä syöttö- ja tulostoimintoja (I/O), kuten verkko-, tallennus- tai grafiikkatoimintoja, käytännössä verkkoselain tai muu suoritusaikainen järjestelmä tarjoaa JavaScriptille rajapinnat I/O-toimintoja varten.

3.3.2 Vue

Vue.js, tunnettu yleisesti nimellä Vue ja lausuttu "view", on avoimen lähdekoodin frontend JavaScript-kirjasto käyttöliittymien ja yksisivuisten sovellusten rakentamiseen (Introduction, n.d). Kirjaston on kehittänyt Evan You, ja sitä ylläpitävät Evan You sekä muut aktiiviset ydintiimin jäsenet (Meet the Team, n.d.).

Vue.js:ssä on inkrementaalisesti mukautuva arkkitehtuuri, joka painottaa deklaratiivista renderöintiä ja komponenttien koostamista. Ydinkirjasto keskittyy pääasiassa näkymäkerrokseen (Introduction, n.d). Vue.js mahdollistaa HTML:n laajentamisen HTML-attribuuteilla, jotka tunnetaan nimellä direktiivit. Nämä direktiivit tarjoavat lisätoiminnallisuuksia HTML-sovelluksille, ja ne voivat olla joko sisäänrakennettuja tai käyttäjän määrittelemiä (What is Vue.js?, n.d.).

Evan You kehitti Vuen työskenneltyään Googlella ja käyttäen AngularJS:ää useissa projekteissa. Hän kuitenkin halusi kehittää jotain kevyempää ja päätti hyödyntää Angularin parhaita osia (Cromwell 2016). Vue.js julkistettiin ensimmäisen kerran helmikuussa 2014 (You 2014).

3.4 Docker

Docker on sarja PaaS-tuotteita (Platform as a Service), jotka hyödyntävät käyttöjärjestelmätason virtualisointia toimittaakseen ohjelmistot kontteina tunnettuihin paketteihin (O'Gara 2013). PaaS on pilvipalveluiden muoto, joka tarjoaa modulaarisen kokonaisuuden laskenta-alustasta ja sovelluksista ilman monimutkaista infrastruktuurin hallintaa. Kehittäjät voivat helposti luoda, kehittää ja hallinnoida ohjelmistoja ilman käynnistys- ja ylläpitotyön vaivaa (Butler 2013).

Dockerissa on tarjolla sekä ilmaisia että premium-tasoja. Dockerissa näitä kontteja isännöi Docker Engine -niminen ohjelmisto, joka julkaistiin ensimmäisen kerran vuonna 2013 Docker, Inc. -yrityksen toimesta. Docker on työkalu, joka automatisoi sovellusten käyttöönoton kevyissä konteissa, mahdollistaen niiden tehokkaan toiminnan erilaisissa ympäristöissä erillään toisistaan. (What is a Container? n.d.)

Konteissa ohjelmistot, kirjastot ja konfiguraatiotiedostot ovat eristettyjä toisistaan, ja ne kommunikoivat keskenään tarkasti määritellyillä kanavilla (Docker frequently asked questions (FAQ), n.d.). Koska kaikki kontit jakavat yhden käyttöjärjestelmän ytimen palvelut, ne käyttävät vähemmän resursseja verrattuna virtuaalikoneisiin (What is a Container? n.d.).

4 Toteutus

4.1 Suunnittelu

Projektin alussa käytiin läpi hyvin lyhyesti aihe työpajasovelluksesta ja keskityttiin valitsemaan projektissa käytettävät teknologiat. Suunnitteluvaiheessa tietokannan teknologia jäi vielä auki, mutta sovelluksen backendiin valittiin teknologiaksi Python Flask-kehyksellä ja frontendiin JavaScriptin Vue.js kehys. Tässä kohtaa toimeksiantajalla ei ollut vielä valmista suunnitelmaa heidän haluamalleen sovellukselle, joten kehitystyö alkoi kehitystyökalujen asentamisella virtuaaliympäristöön odottaessani tarkempia ohjeita. Projektin toisessa palaverissa käytiin läpi työpajatyöskentelyä projektiin orientoitumiseksi sekä toimeksiantajan laatimaa suunnitelmaa.

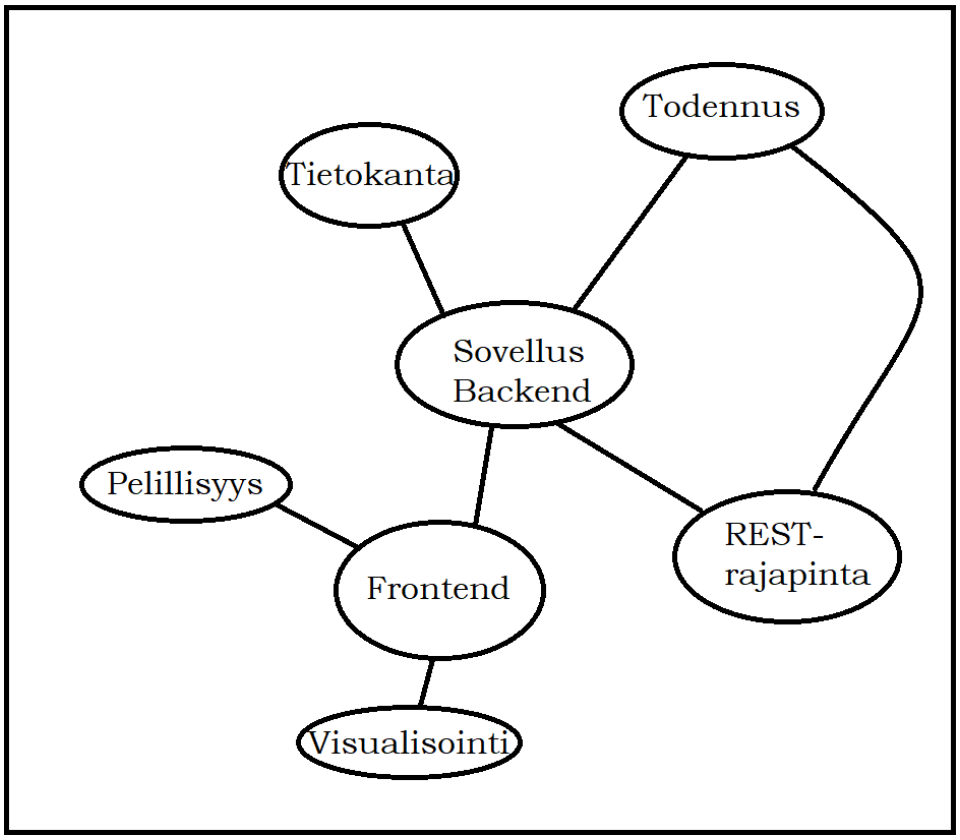
Sovelluksen toteutus määriteltiin olevan ”Ennalta määritelty ”putki”, jonka osallistuja pelaa läpi”. Ominaisuudet ja toiminnallisuudet jaettiin kahteen kategoriaan; pakolliset ja ei-pakolliset. Näistä

pakolliset viittasivat sovelluksen ydintoiminnallisuuksiin, jotka toimeksiantaja oli määritellyt välttämättömiksi. Näihin kuului seuraavat ominaisuudet: Osallistujat voivat vastata kysymyksiin tai suorittaa kirjoitustehtäviä samanaikaisesti omilla laitteillaan. Kysymykset voivat olla esimerkiksi monivalintatehtäviä tai pitkiä tekstivastauksia. Lisäksi osallistujat näkevät toistensa edistymisen ja voivat äänestää tai pisteyttää toistensa vastauksia. Sovellus luo automaattisesti koosteen osallistujien vastauksista tai toimista, ja tämä lähetetään toimeksiantajalle. Vastaukset voidaan säilyttää useamman päivän ajan, mikä on hyödyllistä useampipäiväisissä työpajoissa. Tiedot tuhoutuvat automaattisesti ennalta määritetyn ajan jälkeen tallennuksesta. Järjestelmään kuuluu myös kirjautumisjärjestelmä ja mahdollisuus antaa palautetta.

Ei-pakolliset ominaisuudet viittasivat toiminnallisuuksiin, jotka eivät olleet sovelluksen toiminnan kannalta välttämättömiä, mutta toisivat sille lisäarvoa käyttäjäkokemuksen kannalta. Suhtautuminen näihin ominaisuuksin kehitystyössä oli pitää niitä toisena prioriteettina pakollisten ominaisuuksien rinnalla ja käyttää niihin aikaa vasta ydinominaisuuksien oltua kunnossa. Ei-pakollisiin ominaisuuksiin kuului kyky osallistujille lukea toistensa vastauksia ylläpitäjän määrittelemällä tavalla, osallistujille pisteiden anto heidän edistymisensä perusteella motivaation tukena, kyky muokata vastauksia jälkikäteen niin, että alkuperäinen versio jää talteen, kyky useammalle ihmiselle työskennellä yhdessä saman tehtävän kanssa, kyky viestitellä työpajan vetäjän kanssa tai pyytää apua, alkukartoituskysely, profiilin luominen ja ajastin tai kello, jolla seurata kirjoitusaikaa tehtäväkohtaisesti.

Palaverissa käytiin läpi myös kaksi mahdollista käyttäjäpolkua. Ensimmäisessä käyttäjäpolussa osallistujat kirjautuisivat ensin sovellukseen esimerkiksi pääsykoodilla. Tämän jälkeen he voisivat kirjoittaa mitä tahansa esimerkiksi alkukysymyksen tai -kysymysten pohjalta, jonka jälkeen osallistujien kirjoittamat tekstit olisivat kootusti kaikkien nähtävillä. Osallistujat voisivat lopuksi äänestää vaihtoehtojen välillä. Toisessa käyttäjäpolussa myös ensimmäiseksi osallistujat kirjautuisivat sisään pääsykoodilla, jonka jälkeen he voisivat kirjoittaa mitä tahansa kysymysten pohjalta. Osallistujat voisivat kirjoittaa myös mitä tahansa jonkin virikkeen, kuten kuvan, pohjalta. Lopuksi osallistujat täyttäisivät palautelomakkeen. Esimerkkeinä kysymystyypeistä, mitä työpajassa osallistujille esitettäisiin, listattiin monivalintatehtävät, tekstikenttätehtävät ja luokitukset asteikolla, esimerkiksi 1–5. Erikseen mainittiin myös ylläpitäjän kyky valita, minkä kysymyksien vastauksia voidaan äänestää.

Sovelluksen alustava rakenne (ks. kuvio 1) muodostui seuraavanlaiseksi. Keskimmäisenä on sovelluksen pääasiallinen toiminnallisuus backendissa, mistä on suora yhteys tietokantaan ja frontendiin. Backendistä on myös yhteys REST-rajapintaan ja mahdolliseen ulkoiseen todentamisjärjestelmään, jos sitä ei hoideta backendin sisällä. Rajapinnasta on myös suora yhteys sekä frontendiin että todentamisjärjestelmään, sillä rajapinnan kautta tieto liikkuu frontendin ja backendin välillä. Kun esimerkiksi osallistuja vastaa kysymykseen, joka esitetään hänelle frontendissä, hänen vastauksensa siirtyvät rajapinnan kautta backendiin, josta ne siirtyvät sitten tietokantaan. Osana frontendia on myös sovelluksen visualisointi ja pelillisyys. Nämä on kaaviossa kirjattu erillisiksi osioiksi, vaikka niiden tarkoitus oli olla frontendin sisällä. Pelillisyyttä ei vielä määritelty.

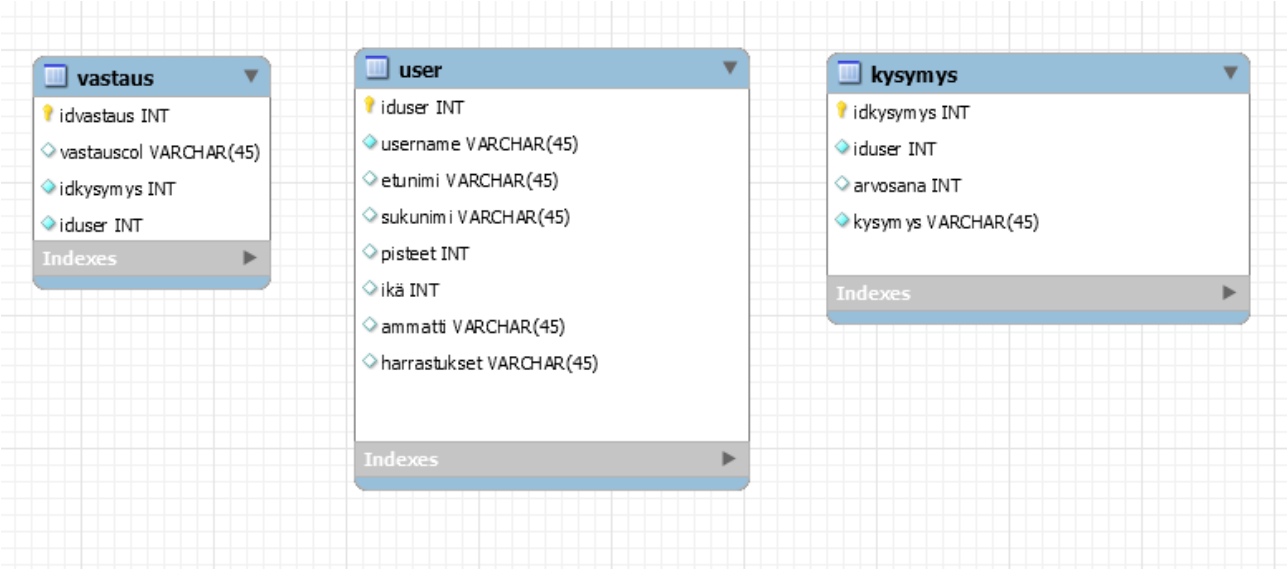


Kuvio 1. Sovelluksen rakenteen alkuperäinen suunnitelma.

4.2 Tietokanta

Sovelluksen tietokanta päätettiin toteuttaa MySQL-teknologialla. Projektin kehitysvaiheessa jokainen versio tietokannasta toimi lokaalisti kehitysympäristön sisällä. Ensimmäisessä luonnoksessa (ks. kuvio 2) laadittiin kolme taulua. Yksi taulu käyttäjätiedoille, yksi taulu kysymyksille ja yksi taulu

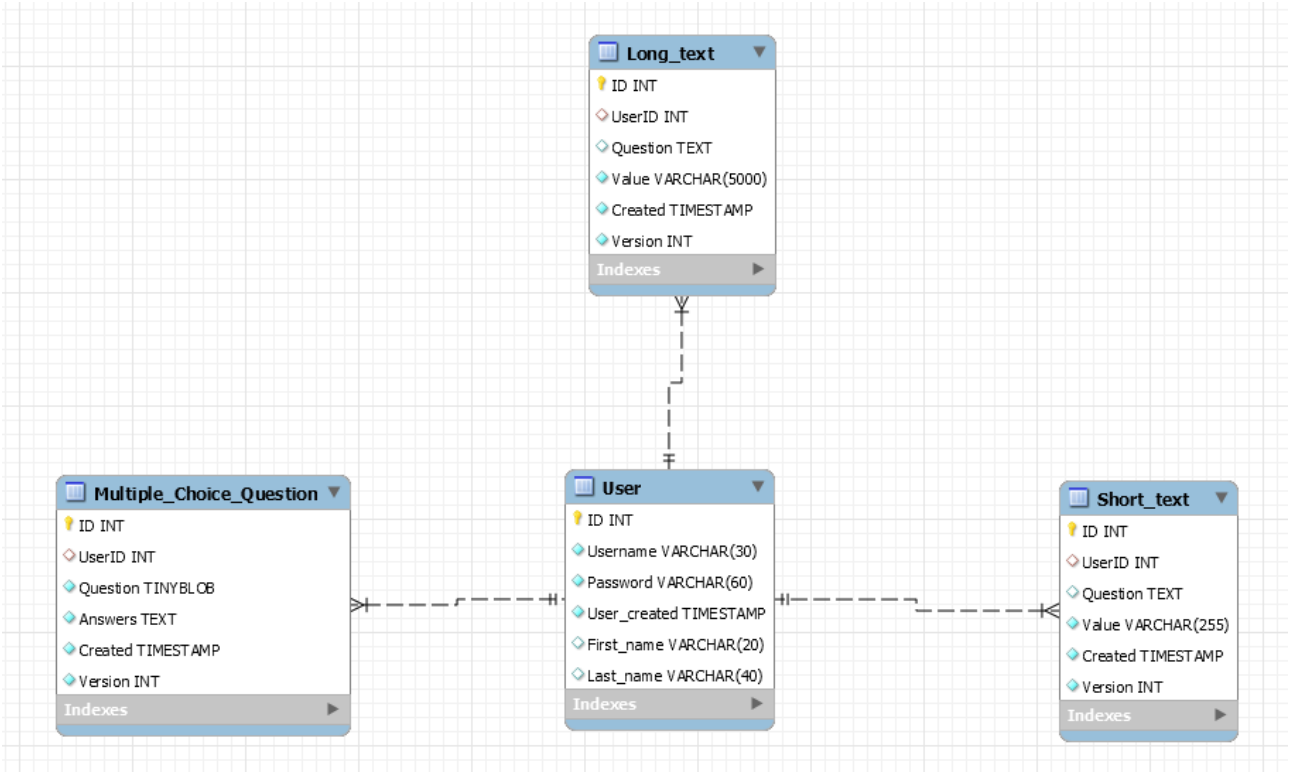
vastauksille. Kysymys- ja vastaustauluissa oli vain tarvittavat sarakkeet, ID, käyttäjäID ja arvo kysymykselle tai vastaukselle. Vastaustaulussa oli myös kysymysID, ja kysymystaulussa oli sarake arvosanalle. Suunnitteluvaiheessa oli kaavailtu ominaisuutta osallistujille antaa kysymyksille arvosana, josta tämä valinta juontui. Käyttäjätaulu oli näistä kolmesta laajin, siinä oli yhteensä 8 saraketta. Suunnitteluvaiheessa mahdollisista käyttäjistä kerättävistä henkilötiedoista listattiin nimen lisäksi osallistujien itse valitsema käyttäjänimi, ikä, ammatti ja harrastukset. Näistä jokaiselle oli oma sarake tietokantataulussa. Muut sarakkeet olivat käyttäjäID ja käyttäjän kerryttämä pistesaldo.



Kuvio 2. Tietokannan ensimmäinen luonnos.

Seuraavissa luonnoksissa (ks. kuvio 3) keskityttiin lisäämään useampia tauluja kysymystyyppien perusteella. Ensimmäisenä oli taulut lyhyille ja pitkille tekstivastauksille. Nämä kaksi taulua olivat muuten samanlaiset, mutta suurin sallittu merkkimäärä vastauksille oli 255 lyhyissä tekstivastauksissa ja 5000 pitkissä. Tässä luonnoksessa kysymykset ja vastaukset olivat mukana samassa taulussa. Uusina lisäyksinä olivat myös sarake aikaleimalle, josta näkee milloin merkintä tauluun on luotu, ja sarake versionumerolle, mistä näkisi montako kertaa kyseistä vastausta on muokattu. Kolmas uusi taulu oli monivalintakysymyksiä varten. Taulun rakenne oli muuten samanlainen tekstikysymystauluihin verrattuna, mutta vastaussarakkeen datatyyppi oli tekstin sijasta tinyblob. Monivalintakysymysten useammat vastausvaihtoehdot olisi ollut hankalaa listata tekstimuodossa, etenkin kun tavoitteena oli noutaa vastausdataa kannasta ja visualisoida sekä pelillistää tätä dataa myö-

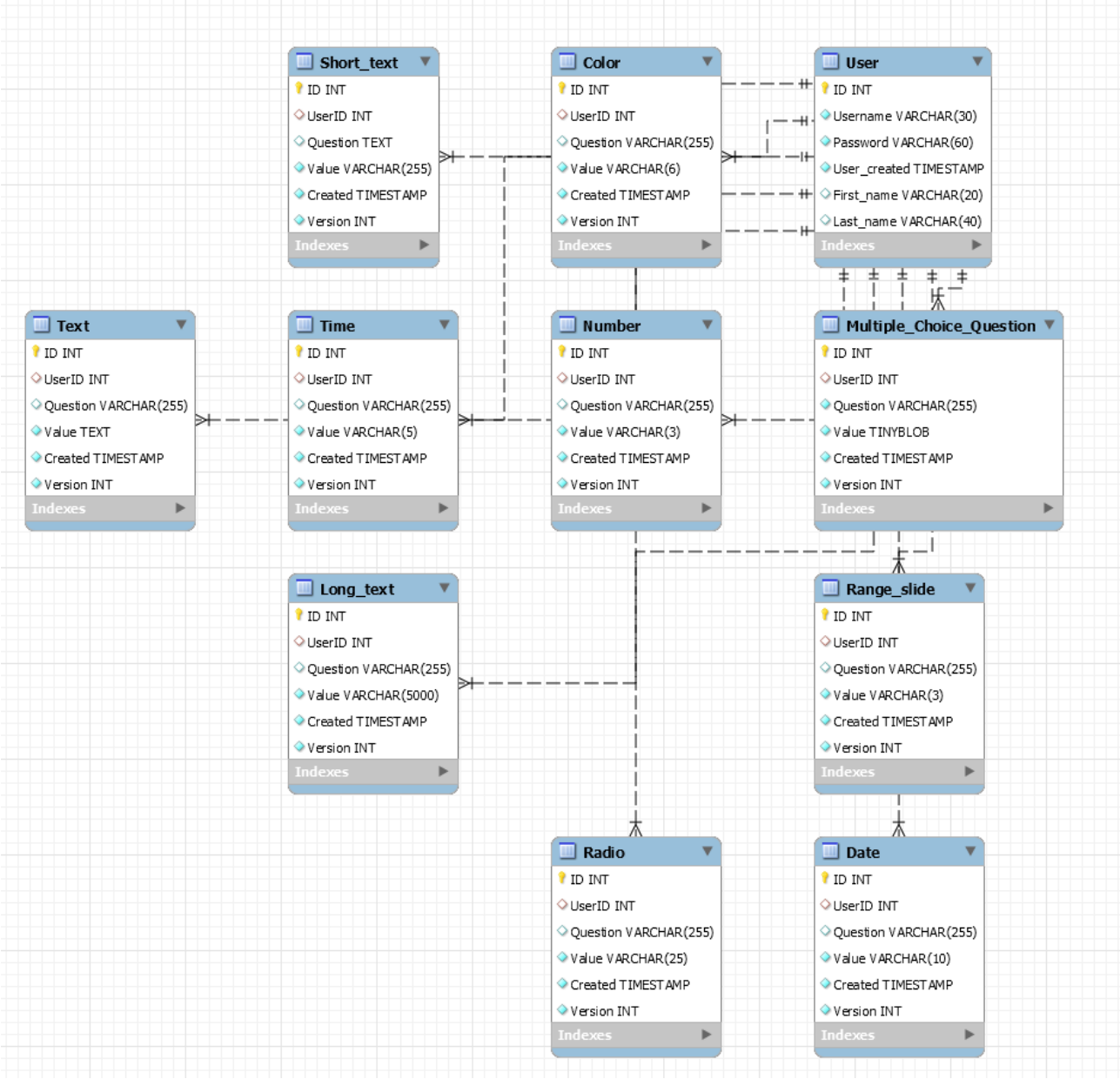
hemmässä vaiheessa. Tinyblobiin oli mahdollista tallentaa useita vastausvaihtoehtoja tekstimuodossa pilkulla erotettuina. Käyttäjätaulusta poistettiin suurin osa henkilötietosarakkeista. Jäljelle jäi sarakkeet etu- ja sukunimille sekä käyttäjänimelle käyttäjäID:n ja aikaleiman lisäksi. Uutena lisäyksenä oli sarake salasanalle. Suunnitelmana oli, että käyttäjät loisivat itselleen tunnukset omalla salasanallaan, jota he sitten käyttäisivät kirjautuakseen sovellukseen sisään työpajan alussa.



Kuvio 3. Tietokannan toinen luonnos.

Tietokannan viimeiseen luonnokseen (ks. kuvio 4), jonka pohjalta projektin kehittämisessä käytetty tietokanta rakennettiin, lisättiin runsaasti lisää tauluja. Toimeksiantaja halusi tietokannassa olevan jo olemassa olevien taulujen lisäksi taulut seuraaville kysymystyypeille: väri, numero, päivämäärä, radio, kellonaika ja liukuvalikko. Nämä pohjautuivat jo olemassa oleviin HTML-attribuutteihin. Sarakkeiltaan nämä taulut keskenään erittäin samanlaisia, suurimmat erot näkyivät arvosarakkeessa. Jokaisen kysymystyypin, pois lukien monivalintakysymykset, arvosarakkeen datatyyppi oli tekstimuodossa, mutta jokaisessa niistä oli eri maksimimerkkimäärä. Esimerkiksi väritaulussa vastausten arvon maksimimerkkimäärä oli 6, sillä HTML-värivalitsin listaa kaikki mahdolliset värit niiden heksadesimaalinumerolla, joka on aina 6 merkkiä pitkä merkkijono.

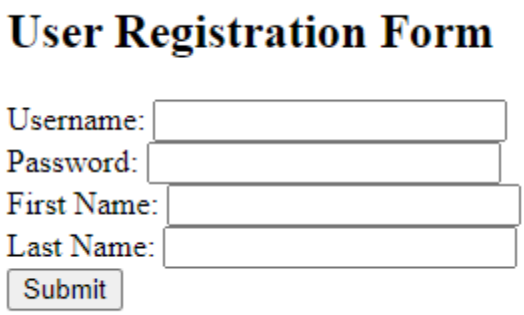
Kehityksen jatkuessa tultiin lopulta siihen tulokseen, että kysymysten säilyttäminen suoraan tietokannassa ei ole kannattava ratkaisu. Koska jokaisessa merkinnässä tietokantatauluun olisi vastauksen lisäksi aina sama kysymyskin, se aiheuttaisi turhaa sekaannusta ja toistoa. Kysymysten ylläpito olisi myös hankalaa, sillä joka kerta kun kysymyksiä tahdottaisiin muokata tai lisätä, se vaatisi tietokantapäivityksen ajamista. Tämän seurauksena jokaisesta taulusta poistettiin kysymyssarakkeet. Todentamisen suhteen ratkaisuksi päädyttiin ottamaan kertakäyttöinen salasana, joka on jokaiselle osallistujalle sama, joten oman salasanan luonti jokaiselle käyttäjälle ei ollut enää tarpeen. Tämän myötä käyttäjätaulusta poistettiin salasanasarake.



Kuvio 4. Tietokannan lopullinen luonnos.

4.3 Backend

Backendin totetukseen valittiin ohjelmointikieli Python ja sen sovelluskehys Flask. Kehittäminen alkoi yksinkertaisilla testeillä lisätä dataa tietokantaan flaskin kautta. Käyttäen flaskin sisäistä renderöintiä, yksinkertaisen HTML-lomakkeen (ks. kuvio 5) kautta käyttäjä pystyisi rekisteröimään uuden käyttäjätunnuksen. Backendissa oleva reitti (ks. kuvio 6) siirsi onnistuneesti HTML-lomakkeeseen syötetyt tiedot tietokantaan. Näitä reittejä tehtiin myös kysymystauluille (ks. kuvio 7). Kuviossa 7 näkyvät reitit ovat lyhyille ja pitkille tekstivastauksille, mutta muiden kysymystyyppien reitit olivat melko lailla identtisiä toiminnaltaan.



Kuvio 5. HTML-lomake uuden käyttäjän luomiseksi.

```
class User(db.Model):
    ID = db.Column(db.Integer, primary_key=True, autoincrement=True)
    Username = db.Column(db.String(30), nullable=False)
    Password = db.Column(db.String(60), nullable=False)
    User_created = db.Column(db.TIMESTAMP, nullable=False, default=datetime.utcnow)
    First_name = db.Column(db.String(20))
    Last_name = db.Column(db.String(40))
```

Kuvio 6. Ensimmäinen versio uuden käyttäjän luonnin reitistä.

```
class LongText(db.Model):
    ID = db.Column(db.Integer, primary_key=True, autoincrement=True)
    UserID = db.Column(db.Integer, db.ForeignKey('user.ID'), nullable=False)
    Question = db.Column(db.String(255))
    Value = db.Column(db.String(5000), nullable=False)
    Created = db.Column(db.TIMESTAMP, nullable=False, default=datetime.utcnow)
    Version = db.Column(db.Integer, nullable=False)

    user = db.relationship('User', backref=db.backref('long_texts', lazy=True))

class ShortText(db.Model):
    ID = db.Column(db.Integer, primary_key=True, autoincrement=True)
    UserID = db.Column(db.Integer, db.ForeignKey('user.ID'), nullable=False)
    Question = db.Column(db.TEXT)
    Value = db.Column(db.String(255), nullable=False)
    Created = db.Column(db.TIMESTAMP, nullable=False, default=datetime.utcnow)
    Version = db.Column(db.Integer, nullable=False)

    user = db.relationship('User', backref=db.backref('short_texts', lazy=True))
```

Kuvio 7. Ensimmäiset versiot tekstivastausten lisäämisreiteistä.

Koska projektiin kuitenkin oli tarkoitus kehittää myös frontend, flaskin sisäinen renderöinti ei kelvannut sovelluksen käyttötarkoituksiin. Tätä varten backendin koodia piti kirjoittaa uusiksi. Uudessa versiossa tietokantaan yhdistämisessä käytettiin SQLAlchemy-kirjaston sijaan mysql.connector-kirjastoa. Backendin alkuun lisättiin toiminta (ks. kuvio 8), joka käyttäjätaulun puuttuessa tietokannasta luo sellaisen automaattisesti.

```
try:
    cnx = mysql.connector.connect(**db_config)
    cursor = cnx.cursor()

    cursor.execute("""
        CREATE TABLE IF NOT EXISTS User (
            ID INT AUTO_INCREMENT PRIMARY KEY,
            Username VARCHAR(30) NOT NULL,
            Password VARCHAR(60) NOT NULL,
            User_created TIMESTAMP NOT NULL,
            First_name VARCHAR(20),
            Last_name VARCHAR(40)
        )
    """)

except mysql.connector.Error as err:
    if err.errno == errorcode.ER_ACCESS_DENIED_ERROR:
        print("Access denied. Check username and password.")
    elif err.errno == errorcode.ER_BAD_DB_ERROR:
        print("Database does not exist.")
    else:
        print(err)

finally:
    if 'cnx' in locals():
        cnx.close()
```

Kuvio 8. User-taulun puuttuessa sen automaattisesti luova koodi.

Myös käyttäjien lisäämistä varten tehty reitti (ks. kuvio 9) kirjoitettiin uudelleen. Kun POST-pyyntö tehdään, funktio hakee JSON-tiedot, jotka sisältävät käyttäjän tiedot, kuten käyttäjänimen, salasanan, etunimen ja sukunimen. Se muodostaa yhteyden tietokantaan käyttämällä db_config-kohdassa määritettyjä konfiguraatiotietoja ja lisää User-tauluun uuden käyttäjätietueen, jossa on annetut tiedot ja User_created-kentän sen hetken UTC-aikaleima. SQL-kysely, jota käytetään tietojen lisäämiseen, on parametrisoitu, mikä tarkoittaa, että kyselymerkkijonossa käytetään sijoitussymboleita (%s) ja todelliset arvot välitetään tupleina cursor.execute-metodille. Parametrisointi auttaa estämään SQL-injektiohyökkäykset varmistamalla, että arvot erotetaan asianmukaisesti ja niitä käsitellään datana. Kun tapahtuma on suoritettu ja tietokantakursori suljettu, funktio palauttaa JSON-vastauksen, joka osoittaa, että käyttäjän luominen on onnistunut. Jos prosessin aikana ilmenee virheitä, funktio palauttaa virheilmoituksen. On syytä huomata, että turvallisuuden vuoksi salasanat olisi hajautettava ennen tallentamista, mitä ei käsitellä tässä esimerkissä. Ainoa ero kuviossa 8 näkyvään reittiin lopullisesta versiosta on salasanan puuttuminen, sillä kehityksen aikana

yksityiset salasanat jokaiselle käyttäjälle todettiin tarpeettomiksi ja siirryttiin käyttämään kerta-käyttöistä jokaiselle osallistujalle yhteistä salasanaa, jonka työpajan ylläpitäjä pystyy asettamaan ja muokkaamaan.

```python
@app.route('/api/create_user', methods=['POST'])
def create_user():
    try:
        data = request.get_json()

        cnx = mysql.connector.connect(**db_config)
        cursor = cnx.cursor()

        query = """
            INSERT INTO User (Username, Password, User_created, First_name, Last_name)
            VALUES (%s, %s, %s, %s, %s)
        """

        cursor.execute(query, (data['Username'], data['Password'], datetime.utcnow(), data['First_name'], data['Last_name']))

        cnx.commit()
        cursor.close()

        return jsonify({"message": "User created successfully"})

    except Exception as e:
        return jsonify({"error": str(e)})
```

Kuvio 9. Lopullinen versio uuden käyttäjän lisäämisen reitistä.

Toimeksiantaja oli toivonut kehitettävän työkalu ylläpitäjälle, jolla kysymyksiä voisi lisätä tietokantaan. Tässä vaiheessa kuitenkin siirryttiin pois mallista, jossa kysymykset säilytettiin tietokannassa, joten tälle työkalulle ei ollut enää tarvetta. Kysymykset asetettiin erillisen python-tiedoston sisään, josta varsinainen backend pystyi ne noutamaan ja esittämään frontendissä. Kysymykset oli muotoiltu tiedostossa dictionaryn eli sanakirjan sisään (ks. kuvio 10), missä kysymyksen ID on avain ja kysymys itsessään on arvo. Monivalintakysymyksissä arvo oli aseteltu aaltosulkujen sisään, mikä mahdollisti suuremman jaksottelun kysymyksen otsikkoon ja vastausvaihtoehtoihin, milloin myös vastausvaihtoehdot monivalintakysymyksissä pystyttiin pitämään kysymystiedoston sisällä, josta ne saatiin esitettyä frontendissa.

```
radio_questions = {
    1: {
        "question": "What is your favorite color?",
        "options": ["Red", "Blue", "Green", "Other"]
    },
    2: {
        "question": "What is your favorite season?",
        "options": ["Summer", "Spring", "Autumn", "Winter"]
    },

    # Add more questions as needed
}
```

Kuvio 10. Sanakirja sisältäen radiokysymykset vastausvaihtoehtoineen kysymystiedostossa.

Flaskin sisäisestä renderöinnistä ulkoiseen frontendiin siirtyminen tarkoitti sitä, että myös muut reitit datan lisäämiseksi tauluihin piti kirjoittaa uudelleen. Toiminnaltaan lähes jokainen reitti datan lisäämiseen on identtinen, mutta käytän esimerkkinä reittiä radiokysymysten vastausten lisäämiseen (ks. kuvio 11). Funktio hakee JSON-tiedot pyynnön rungosta, joka sisältää question_id- ja answer_value -parit. Se muodostaa yhteyden tietokantaan käyttämällä db_config-kohdassa määritettyjä konfiguraatiotietoja ja iteroi datan läpi lisätäkseen kunkin vastauksen Radio-tauluun ja liittääkseen sen kovakoodattuun UserID:hen (joka olisi korvattava todellisessa toteutuksessa todellisella käyttäjätunnuksella). Jokainen tietue sisältää myös sen hetken UTC-aikaleiman ja versionumeron. Funktio sitouttaa tapahtuman muutosten tallentamiseksi ja sulkee kursorin. Jos virheitä ilmenee, funktio palauttaa virheilmoituksen. Jos se onnistuu, se vastaa vahvistusviestillä.

```python
# Submit radio answers to the database
@app.route('/api/submit_radio_answers', methods=['POST'])
def submit_radio_answers():
    try:
        data = request.get_json()

        cnx = mysql.connector.connect(**db_config)
        cursor = cnx.cursor()

        for question_id, answer_value in data.items():
            query = """
                INSERT INTO Radio (UserID, QuestionID, Value, Created, Version)
                VALUES (%s, %s, %s, %s, %s)
            """

            cursor.execute(query, (1, question_id, answer_value, datetime.utcnow(), 1))  # Replace 1 with actual user ID

        cnx.commit()
        cursor.close()

        return jsonify({"message": "Radio answers submitted successfully"})

    except Exception as e:
        return jsonify({"error": str(e)})
```

Kuvio 11. Reitti radiovastausten lisäämiseen.

Merkittävin ero oli checkbox-monivalintakysymysten vastausten lisäämisen reitissä. Radiokysymykset ovat monivalintakysymyksiä, missä käyttäjä pystyy valitsemaan vain yhden vastausvaihtoehdoista, mutta checkbox-kysymyksissä käyttäjät voivat valita vastausvaihtoehdoista niin monta kuin tahtovat. Tämän vuoksi on tärkeää varmistaa, että reitti lisää datan tietokantaan sopivassa muodossa. Checkbox reitissä (ks. kuvio 12) jokainen valittu arvo liitetään yhteen yhdeksi merkkijonoksi pilkulla erotettuina.

```python
# Submit checkbox answers to the database
@app.route('/api/submit_checkbox_answers', methods=['POST'])
def submit_checkbox_answers():
    try:
        data = request.get_json()

        cnx = mysql.connector.connect(**db_config)
        cursor = cnx.cursor()

        for question_id, selected_values in data.items():
            # Join selected values into a single string
            combined_values = ', '.join(selected_values)

            query = """
                INSERT INTO Checkbox (UserID, QuestionID, Value, Created, Version)
                VALUES (%s, %s, %s, %s, %s)
            """

            cursor.execute(query, (1, question_id, combined_values, datetime.utcnow(), 1))  # Replace 1 with actual user ID

        cnx.commit()
        cursor.close()

        return jsonify({"message": "Checkbox answers submitted successfully"})

    except Exception as e:
        return jsonify({"error": str(e)})
```

Kuvio 12. Reitti checkbox-vastausten lisäämiseen.

Kun POST-rajapinnat oli saatu toimiviksi, seuraavaksi siirryttiin kehittämään GET-rajapintaa datan noutamiseksi tietokannasta ja sen esittämiseksi frontendissa. Tärkeimmät reitit olivat reitti hakea käyttäjätietoja sekä reitti hakea yksittäisen käyttäjän vastauksia kaikkiin kysymyksiin. Käyttäjätietojen haun reitissä (ks. kuvio 13), kun GET-pyyntö tehdään, funktio luo yhteyden tietokantaan käyttämällä db_config-kohdassa määritettyjä konfiguraatiotietoja ja luo kursorin SQL-kyselyjen suorittamista varten. Jos pyyntö sisältää parametrin käyttäjätunnus, se etsii LIKE-kyselyllä käyttäjiä, joiden käyttäjätunnus vastaa annettua parametria, ja mahdollistaa osittaiset osumat. Jos käyttäjätunnusparametria ei anneta, se hakee kaikki käyttäjätietueet User-taulusta. Tämän jälkeen tulokset haetaan ja muotoillaan sanakirjaluetteloksi, joista kukin edustaa käyttäjää avaimilla kuten ID, Username, Password, User_created, First_name ja Last_name. Kursorin sulkemisen jälkeen funktio palauttaa muotoillut käyttäjätiedot JSON-vastauksena. Jos prosessin aikana ilmenee virheitä, funktio palauttaa virheilmoituksen. Tämä funktio takaa joustavan tiedonhaun, sillä se mahdollistaa joko tietyn käyttäjän haun tai kaikkien käyttäjätietueiden haun.

```python
@app.route('/api/user_data', methods=['GET'])
def get_user_data():
    try:
        cnx = mysql.connector.connect(**db_config)
        cursor = cnx.cursor()

        if 'username' in request.args:
            # Search for a specific user by username
            query = "SELECT * FROM User WHERE Username LIKE %s"
            cursor.execute(query, ('%' + request.args['username'] + '%',))
        else:
            # Fetch all users
            query = "SELECT * FROM User"
            cursor.execute(query)

        data = cursor.fetchall()

        user_data = [{'ID': row[0], 'Username': row[1], 'Password': row[2], 'User_created': row[3], 'First_name': row[4], 'Last_name': row[5]} for row in data]

        cursor.close()

        return jsonify(user_data)

    except Exception as e:
        return jsonify({"error": str(e)})
```

Kuvio 13. Reitti käyttäjätietojen hakemiseen.

Seuraava kehitetty GET-rajapintareitti oli yksittäisen käyttäjän kaikkien vastausten haku (ks. kuvio 14). Kun GET-pyyntö tehdään, funktio hakee käyttäjätunnusparametrin käyttäjän syöttämästä merkkijonosta ja muodostaa yhteyden tietokantaan db_configin avulla. Funktio tarkistaa ensin, onko käyttäjä olemassa, tekemällä kyselyn User-taulusta käyttäjän tunnuksen löytämiseksi annetun käyttäjänimen perusteella. Jos käyttäjää ei löydy, se palauttaa 404-virheilmoituksen, jossa lukee ”Käyttäjää ei löydy”. Jos käyttäjä on olemassa, se suorittaa kyselyn, jolla haetaan vastaukset sekä Radio- että Checkbox-taulukoista käyttäjän tunnuksen osalta, ja yhdistää tarvittaessa tulokset QuestionID:n avulla. Tulokset muotoillaan sanakirjaluetteloksi, joista jokainen sisältää QuestionID:n, RadioAnswerin ja CheckboxAnswerin. Kun kursori on suljettu, funktio palauttaa käyttäjän vastaukset JSON-muodossa. Jos prosessin aikana ilmenee virheitä, funktio palauttaa virheilmoituksen. Tämä funktio yhdistää tehokkaasti käyttäjän vastaukset kahdesta eri taulukosta ja hoitaa käyttäjän olemassaolon validoinnin. Tässä vaiheessa funktio yhdistelee vastaukset vain kahdesta eri taulusta, mutta siihen on helppo myöhemmin lisätä lisää tauluja.

```python
@app.route('/api/user_answers', methods=['GET'])
def get_user_answers():
    try:
        username = request.args.get('username')

        cnx = mysql.connector.connect(**db_config)
        cursor = cnx.cursor()

        # Check if the user exists
        cursor.execute("SELECT ID FROM User WHERE Username = %s", (username,))
        user_id = cursor.fetchone()

        if not user_id:
            # User not found, return 404 status
            return jsonify({"error": "User not found"}), 404

        #
        query = """
            SELECT r.QuestionID, r.Value AS RadioAnswer, c.Value AS CheckboxAnswer
            FROM Radio r
            LEFT JOIN Checkbox c ON r.QuestionID = c.QuestionID
            WHERE r.UserID = %s OR c.UserID = %s
        """

        cursor.execute(query, (user_id[0], user_id[0]))
        data = cursor.fetchall()

        user_answers = [{'QuestionID': row[0], 'RadioAnswer': row[1], 'CheckboxAnswer': row[2]} for row in data]

        cursor.close()

        return jsonify(user_answers)

    except Exception as e:
        return jsonify({"error": str(e)})
```

Kuvio 14. Reitti yksittäisen käyttäjän kaikkien vastausten hakuun.

Kolmas kehitetty GET-reitti (ks. kuvio 15), jota kuitenkaan ei käytetty toteutuksessa frontendin kanssa, mahdollisti jokaisen vastausarvon noutoa tietystä taulusta ja tietyllä QuestionID:llä käyttäjän syöttämien parametrien perusteella. Kun GET-pyyntö tehdään, funktio poimii taulu- ja QuestionID-parametrit käyttäjän syöttämistä merkkijonoista. Jos jompikumpi parametreista puuttuu, funktio palauttaa 400-virheilmoituksen asianmukaisen viestin kera. Jos molemmat parametrit ovat olemassa, funktio muodostaa yhteyden tietokantaan db_configin avulla ja suorittaa parametrisoidun SQL-kyselyn noutaakseen Value-kentän määritetystä taulusta, jossa QuestionID vastaa annettua ID:tä. Haetut tulokset käsitellään sitten arvoluetteloksi, joka palautetaan JSON-vastauksena. Tietokantavirheen esiintyessä, funktio kirjaa virheilmoituksen ja palauttaa tietokantaongelmasta kertovan 500-virheilmoituksen. Lopuksi funktio varmistaa, että tietokantakursori ja -yhteys suljetaan asianmukaisesti tuloksesta riippumatta.

```python
@app.route('/values', methods=['GET'])
def get_values():
    table = request.args.get('table')
    question_id = request.args.get('questionId')

    if not table or not question_id:
        return jsonify({'error': 'Table name and Question ID are required.'}), 400

    try:
        conn = mysql.connector.connect(**db_config)
        cursor = conn.cursor()
        query = f"SELECT Value FROM {table} WHERE QuestionID = %s"
        cursor.execute(query, (question_id,))
        results = cursor.fetchall()
        values = [row[0] for row in results]
        return jsonify(values)
    except mysql.connector.Error as err:
        print(f"Error: {err}")
        return jsonify({'error': 'Error fetching values from database.'}), 500
    finally:
        cursor.close()
        conn.close()
```

Kuvio 15. Reitti vastausten etsimiseen taulun ja kysymys ID:n perusteella.

Koska kysymykset päätettiin siirtää pois tietokannasta ja erilliseen tiedostoon, backendiin tarvitsi kirjoittaa reitit (ks. kuvio 16), jotka noutavat kysymykset erillisestä tiedostosta. Kaikki kysymysten noutamiseen tehdyt reitit toimivat samalla tavalla. GET-pyynnön tapahtuessa funktio tuo kysymystiedostosta reitissä määritellyn nimen mukaisen sanakirjan sisällön, joka sisältää kysymysID ja kysymysarvo avain- arvoparit. Sanakirja palautetaan sen jälkeen JSON-vastauksena.

```python
# Checkbox table route
@app.route('/api/checkbox_questions', methods=['GET'])
def get_checkbox_questions():
    # Fetch questions from a separate Python file (questions.py)
    from questions import checkbox_questions
    return jsonify(checkbox_questions)

# Short text table route
@app.route('/api/short_text_questions', methods=['GET'])
def get_short_questions():
    # Fetch questions from a separate Python file (questions.py)
    from questions import short_text_questions
    return jsonify(short_text_questions)

# Long text table route
@app.route('/api/long_text_questions', methods=['GET'])
def get_long_questions():
    # Fetch questions from a separate Python file (questions.py)
    from questions import long_text_questions
    return jsonify(long_text_questions)
```

Kuvio 16. Osa kysymysten noudon reiteistä.

Viimeisenä backendiin kehitettiin reitti, jolla halutusta tietokantataulusta pystyy hakemaan vastausdatan kysymysID:n perusteella (ks. kuvio 17). Projektin puitteissa tehtiin reitti vain radiotaulun kysymysID 1 vastausten datan noudolle, mutta reitin muokkaaminen halutun taulun tai kysymysID:n saamiseksi on varsin yksinkertaista. Kun GET-pyyntö tehdään, funktio muodostaa yhteyden tietokantaan käyttämällä db_config-kohdassa määritettyjä konfiguraatiotietoja ja luo kursorin SQL-kyselyjen suorittamista varten. Sen jälkeen funktio suorittaa kyselyn, jolla valitaan Radio-taulun Value-kenttä, jossa QuestionID on 1. Kyselyn tulokset haetaan ja arvot poimitaan luetteloksi. Kursorin sulkemisen jälkeen funktio palauttaa tämän arvoluettelon JSON-vastauksena. Jos prosessin aikana ilmenee virheitä, funktio palauttaa virheilmoituksen JSON-vastauksena.

```python
#Route to fetch all values from the radio table with the question ID of 1
@app.route('/api/radio_data', methods=['GET'])
def get_radio_data():
    try:
        cnx = mysql.connector.connect(**db_config)
        cursor = cnx.cursor()

        # Fetch data from the "Radio" table for question ID 1
        query = "SELECT Value FROM Radio WHERE QuestionID = 1"
        cursor.execute(query)
        data = cursor.fetchall()

        # Extract values from the fetched data
        values = [row[0] for row in data]

        cursor.close()

        return jsonify(values)

    except Exception as e:
        return jsonify({"error": str(e)})
```

Kuvio 17. Reitti radiotaulun kysymysID 1 vastausdatan noutoon.

Sovelluksen suunnittelussa yksi haluttu ominaisuus oli todennus, jonka avulla työpajan osallistujat pääsisivät kirjautumaan sisään sovellukseen. Kehitystä tehtiin todennusjärjestelmän toteuttamiseksi alkuperäistä suunnitelmaa mukaillen, mutta kun käyttäjäkohtaisista salasanoista päätettiin luopua, aika ei enää riittänyt uuden login-funktion kehittämiseen. Näin ollen todennus jäi valmiista projektista puuttumaan.

4.4 Frontend

Sovelluksen frontendiin valittiin JavaScriptin Vue.js-sovelluskehys. Toimeksiantaja toivoi toteutuksen olevan selaimen sisällä toimiva kokonaisuus, jossa osallistujat vastaisivat yksittäisiin kysymyksiin ja kysymysten välillä olisi visualisoitua vastausdataa ja pelillistettyjä ominaisuuksia. Kehityksen aikana testattiin muutamaa eri menetelmää saavuttaa halutut tulokset, kuten yhden sivun ratkaisua (Single Page Application), jossa kaikki sovelluksen komponentit olisivat saman tiedoston sisällä

olevia uudelleenkäytettäviä osia. Ratkaisu, johon lopulta päädyttiin, koostui useista sivuista. Frontendin rakenne koostui yhdestä päätiedostosta, joka ohjaa kaikkia sovelluksen sivuja ja niiden välillä navigointia, sekä jokaisesta yksittäisestä sivusta, joista jokainen on myös oma tiedostonsa.

Frontendin päätiedostossa (ks. kuvio 18) on ”Edellinen” ja ”Seuraava” -painikkeet eri sivujen välillä navigoimista varten, ja ne kytkeytyvät pois päältä nykyisen sivunumeron mukaan, jotta navigointi ensimmäisen tai viimeisen sivun ulkopuolelle ei ole mahdollista. CurrentPageComponent-laskenta-ominaisuus palauttaa dynaamisesti oikean komponentin currentPage-arvon perusteella, jolloin kukin sivunumero yhdistetään tiettyyn komponenttiin, kuten Home, CreateUser, About ja muut. NavigateToPreviousPage- ja navigateToNextPage-metodit päivittävät currentPage-arvon rajojen sisällä, mikä mahdollistaa navigoinnin sivujen välillä. Komponentti hallinnoi yhtätoista sivua, vaihtelee niiden välillä dynaamisesti ja varmistaa eri komponenttien turvallisen navigoinnin ja renderöinnin. Sivujen järjestystä voi muuttaa yksinkertaisesti vaihtamalla switch-casen numero halutun komponentin kohdalla halutuksi numeroksi. Sivujen lisääminen myöhemmässä vaiheessa onnistuu yksinkertaisesti lisäämällä uusi case switch-caseen ja muokkaamalla totalPages-muuttujan luku vastaamaan todellista sivujen lukumäärää.

```
export default {
  data() {
    return {
      currentPage: 1,
      totalPages: 11, // Adjust based on the total number of pages
    };
  },
  computed: {
    currentPageComponent() {
      switch (this.currentPage) {
        case 1:
          return Home;
        case 2:
          return CreateUser;
        case 3:
          return Login;
        case 4:
          return Search;
        case 5:
          return RadioQuestions;
        case 6:
          return Checkbox;
        case 7:
          return SearchAnswers;
        case 8:
          return About;
        case 9:
          return PieChart;
        case 10:
          return ShortText;
        case 11:
          return QuestionAnswers;
        default:
          return Home;
      }
    },
  },
  methods: {
    navigateToPreviousPage() {
      if (this.currentPage > 1) {
        this.currentPage -= 1;
      }
    },
    navigateToNextPage() {
      if (this.currentPage < this.totalPages) {
        this.currentPage += 1;
      }
    },
  },
```

Kuvio 18. Frontendin navigointijärjestelmä.

Projektin aikana frontendiin kehitettiin seuraavat komponentit: CreateUser uusien käyttäjien luomista varten, Search käyttäjien datan hakemiseen tietokannasta, RadioQuestions radiokysymyksiin vastaamista varten, Checkbox monivalintakysymyksiin vastaamista varten, SearchAnswers käyttäjien vastausten hakemista varten, PieChart käyttäjien vastausdatan visualisointia varten, ShortText lyhyisiin tekstikysymyksiin vastaamista varten ja QuestionAnswers tekstivastausten hakemista ja esittämistä varten. Aiemmin mainitut Home- ja About-komponentit jäivät sisällöiltään kokonaan

tyhjiksi. Login-komponenttia todennusta varten alettiin kehittää myös, mutta ajan puutteen vuoksi sitä ei ehditty saada valmiiksi.

CreateUser (ks. kuvio 19) on lomake uuden käyttäjän luomista varten, jossa on kentät käyttäjätunnukselle, salasanalle, etunimelle ja sukunimelle, jotka on sidottu formData-objektiin v-modelin avulla. Lomakkeen lähettäminen hoidetaan createUser-metodilla, joka estää lomakkeen lähettämisen oletuskäyttäytymisen ja puhdistaa käyttäjän syötteen DOMPurify-menetelmällä XSS-hyökkäysten estämiseksi. Aluksi tehdään GET-pyyntö /api/create_user-päätepisteeseen CSRF-tunnisteen luomisen käynnistämiseksi. Tämän jälkeen samaan päätepisteeseen lähetetään POST-pyyntö, jossa on puhdistetut lomaketiedot JSON-muodossa. Vastauksen saatuaan hälytys näyttää joko onnistumis- tai virheilmoituksen, ja jos vastaus on onnistunut, lomakkeen kentät nollataan. Komponentti sisältää perustyylit lomakkeen esitystapaa varten ja varmistaa, että käyttäjätiedot käsitellään turvallisesti ja lähetetään backendiin käyttäjän luomista varten.

```
export default {
  data() {
    return {
      formData: {
        Username: '',
        Password: '',
        First_name: '',
        Last_name: '',
      },
      csrfToken: '',  // Add this line to store CSRF token
    };
  },
  methods: {
    createUser() {
      // Sanitize user input using Dompurify
      const sanitizedFormData = {
        Username: DOMPurify.sanitize(this.formData.Username),
        Password: DOMPurify.sanitize(this.formData.Password),
        First_name: DOMPurify.sanitize(this.formData.First_name),
        Last_name: DOMPurify.sanitize(this.formData.Last_name),
      };

      // Make a dummy request to the create_user endpoint to trigger CSRF token generation
      fetch('http://127.0.0.1:5000/api/create_user', {
        method: 'GET',
      })
        .then(() => {
          // Make the actual POST request with the generated CSRF token
          return fetch('http://127.0.0.1:5000/api/create_user', {
            method: 'POST',
            headers: {
              'Content-Type': 'application/json',
            },
            body: JSON.stringify(sanitizedFormData),
          });
        })
        .then((response) => response.json())
        .then((data) => {
          alert(data.message || data.error);
          if (!data.error) {
            this.formData.Username = '';
            this.formData.Password = '';
            this.formData.First_name = '';
            this.formData.Last_name = '';
          }
        })
        .catch((error) => {
          console.error('Error:', error);
        });
```

Kuvio 19. CreateUser komponentti.

Search-komponentti (ks. kuvio 20) on käyttöliittymä käyttäjähallintaan, joka mahdollistaa käyttäjätietojen hakemisen ja näyttämisen tietokannasta backendin kautta. Malli sisältää painikkeen kaikkien käyttäjien hakemiseen, syöttökentän ja painikkeen tietyn käyttäjän etsimiseen käyttäjänimen perusteella sekä osion tulosten näyttämiseen. Kun ”Get All Users” -painiketta napsautetaan, getAllUsers-metodi hakee käyttäjätiedot /api/user_data-päätepisteestä ja puhdistaa jokaisen käyttäjänimen DOMPurify-menetelmällä XSS-hyökkäysten estämiseksi, ennen kuin käyttäjälista päivitetään. SearchUser-metodi, joka käynnistyy napsauttamalla ”Search User”-painiketta, noutaa tietoja samasta päätepisteestä käyttäen käyttäjän syötettä kyselyparametrina, puhdistaa palautetun käyttäjänimen ja määrittää ensimmäisen tuloksen searchedUserille tyhjentäen samalla käyttäjälistan. Komponentti näyttää luettelot kaikista käyttäjistä tai haetusta käyttäjästä haettujen tietojen perusteella. Muotoilua sovelletaan siistin ulkoasun varmistamiseksi, ja virheenkäsittely on sisällytetty lokiin tietojen hakuoperaatioiden aikana ilmenevien ongelmien kirjaamiseksi.

```
export default {
props: ['pageNumber'],
name: 'SearchPage',
data() {
  return {
    users: [],
    username: '',
    searchedUser: null,
  };
},
methods: {
  getAllUsers() {
    fetch('http://127.0.0.1:5000/api/user_data')
      .then(response => response.json())
      .then(data => {
        // Sanitize HTML content using Dompurify before assigning to users
        this.users = data.map(user => ({
          ...user,
          Username: DOMPurify.sanitize(user.Username),
          // Sanitize other properties if necessary
        }));
        this.searchedUser = null;
      })
      .catch(error => console.error('Error:', error));
  },
  searchUser() {
    if (this.username.trim() === '') {
      return;
    }

    fetch(`http://127.0.0.1:5000/api/user_data?username=${encodeURIComponent(this.username)}`)
      .then(response => response.json())
      .then(data => {
        if (data.length > 0) {
          // Sanitize HTML content using Dompurify before assigning to searchedUser
          this.searchedUser = {
            ...data[0],
            Username: DOMPurify.sanitize(data[0].Username),
            // Sanitize other properties if necessary
          };
          this.users = [];
        } else {
          this.searchedUser = null;
        }
      })
      .catch(error => console.error('Error:', error));
  },
},
```

Kuvio 20. Search-komponentti.

RadioQuestions-komponentti (ks. kuvio 21) on käyttöliittymä radiokysymysten näyttämiseen ja lähettämiseen. Malli renderöi luettelon kysymyksistä, joista kukin sisältää omat vaihtoehtonsa, valintapainikkeina. Data-funktio alustaa questions- ja selectedOptions-objektit, joihin tallennetaan haetut kysymykset ja käyttäjän valitsemat vastaukset. Kun komponentti mountataan, se kutsuu fetchQuestions-kutsua noutaakseen kysymykset backendista ja täyttääkseen questions-olion. Käyttäjä voi valita vastauksia kuhunkin kysymykseen, ja nämä valinnat sidotaan selectedOptions-olioon v-modelin avulla. Kun painiketta ”Submit Answers” (Lähetä vastaukset) napsautetaan, submitAnswers-metodi kerää valitut vastaukset, lähettää ne POST-pyynnöllä backendiin ja käsittelee vastauksen näyttämällä hälytyksen, jossa on backendin viesti tai virhe. Lähettämisen jälkeen valitut vaihtoehdot tyhjennetään ja answers-submitted-tapahtuma lähetetään. Checkbox-komponentti monivalintakysymyksille toimii suurimmilta osin samalla tavalla kuin RadioQuestions.

```
export default {
  methods: {
    fetchQuestions() {
      fetch("http://127.0.0.1:5000/api/radio_questions")
        .then((response) => response.json())
        .then((data) => {
          this.questions = data;
        })
        .catch((error) => {
          console.error("Error fetching questions:", error);
        });
    },
    submitAnswers() {
      const answerData = {};

      Object.keys(this.selectedOptions).forEach((questionID) => {
        const selectedOption = this.selectedOptions[questionID];

        if (selectedOption !== null) {
          answerData[questionID] = selectedOption;
        }
      });

      fetch("http://127.0.0.1:5000/api/submit_radio_answers", {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify(answerData),
      })
        .then((response) => response.json())
        .then((data) => {
          // Handle the response from the backend if needed
          alert(data.message || data.error);
        })
        .catch((error) => {
          console.error("Error submitting answers:", error);
        });

      // Optionally, emit an event to notify the parent component about the submission
      this.$emit("answers-submitted", answerData);

      // Clear selected options after submission
      this.clearSelectedOptions();
    },
    clearSelectedOptions() {
      Object.keys(this.selectedOptions).forEach((questionID) => {
        this.selectedOptions[questionID] = null;
      });
```

Kuvio 21. RadioQuestions-komponentti.

SearchAnswers-komponentti (ks. kuvio 22) on käyttöliittymä, jonka avulla voidaan etsiä ja näyttää käyttäjän vastaukset radio- ja monivalintakysymyksiin. Mallissa on syöttökenttä käyttäjänimen syöttämistä varten ja painike haun suorittamiseksi. Hakutuloksen mukaan se näyttää erilaisia viestejä: ”Käyttäjää ei löytynyt”, ”Käyttäjä löytyi, mutta vastauksia ei löytynyt” tai luettelo käyttäjän vastauksista, jos sellaisia löytyy. Data-funktio alustaa muuttujat searchUsername, userAnswers ja searchStatus komponentin tilan hallintaa varten. SearchUserAnswers-metodi tekee GET-pyynnön backendiin syötetyllä käyttäjänimellä ja päivittää searchStatus-tilan vastauksen perusteella: 'not-found', jos käyttäjää ei ole olemassa (404-tilan käsittely), 'found-no-answers', jos vastauksia ei löydy, tai 'found-with-answers', jos vastauksia on haettu. Sanitize-metodi käyttää DOMPurify-menetelmää vastausten sisällön puhdistamiseen, mikä estää XSS-hyökkäykset. Näin varmistetaan, että dynaaminen sisältö renderöidään turvallisesti käyttäjän selaimessa.

```
export default {
  data() {
    return {
      searchUsername: '',
      userAnswers: [],
      searchStatus: '', // 'not-found', 'found-no-answers', or 'found-with-answers'
    };
  },
  methods: {
    searchUserAnswers() {
      fetch(`http://127.0.0.1:5000/api/user_answers?username=${this.searchUsername}`)
        .then((response) => {
          if (response.status === 404) {
            this.searchStatus = 'not-found';
            throw new Error('User not found.');
          } else if (!response.ok) {
            throw new Error('Error fetching user answers.');
          }
          return response.json();
        })
        .then((data) => {
          if (data.length === 0) {
            this.searchStatus = 'found-no-answers';
          } else {
            this.searchStatus = 'found-with-answers';
            this.userAnswers = data;
          }
        })
        .catch((error) => {
          console.error("Error fetching user answers:", error);
          this.userAnswers = [];
        });
    },
    sanitize(content) {
      return DOMPurify.sanitize(content);
    },
  },
};
```

Kuvio 22. SearchAnswers-komponentti.

PieChart-komponentti (ks. kuvio 23) renderöi piirakkakaavion, joka näyttää tulokset kysymyksestä, jonka kysymysID on 1. Malli käyttää Chart.js-kirjastoa kaavion renderöintiin. Mallissa piirakkakaavion kohteeksi annetaan canvas-elementti. Kun komponentti mountataan, se hakee tietoja backendin kautta tietokannasta. Käyttäjän vastauksia edustavia tietoja käsitellään kunkin vastausarvon esiintymien laskemiseksi. Näitä lukuja käytetään sitten piirakkakaavion merkintöjen ja arvojen määrittämiseen. Chart-olio luodaan näiden parametrien avulla, ja piirakkakaavio renderöidään canvas-elementissä. #chart-container-div on muotoiltu siten, että kaavion leveys ja korkeus ovat kiinteät. Asetuksella varmistetaan, että kaavio on responsiivinen ja säilyttää kuvasuhteensa säiliön koon muutoksista riippumatta.

```
import { ref, onMounted } from 'vue';
import Chart from 'chart.js/auto';
export default {
  name: 'PieChart',
  setup() {
    const chart = ref(null);

    onMounted(() => {
      fetch('http://127.0.0.1:5000/api/radio_data')
        .then(response => {
          if (!response.ok) {
            throw new Error('Network response was not ok');
          }
          return response.json();
        })
        .then(data => {
          const colorCounts = {};
          data.forEach(color => {
            colorCounts[color] = (colorCounts[color] || 0) + 1;
          });

          const labels = Object.keys(colorCounts);
          const values = Object.values(colorCounts);
          const backgroundColors = ['red', 'blue', 'green', 'purple', 'yellow']; // Add more colors as needed

          const ctx = document.getElementById('pie-chart').getContext('2d');
          chart.value = new Chart(ctx, {
            type: 'pie',
            data: {
              labels: labels,
              datasets: [{
                data: values,
                backgroundColor: backgroundColors
              }]
            },
            options: {
              responsive: true,
              maintainAspectRatio: false
            }
          });
        })
        .catch(error => {
          console.error('Error fetching radio data:', error);
        });
    });
    return {
      chart
```

Kuvio 23. PieChart-komponentti.

ShortText-komponentti (ks. kuvio 24) huolehtii tekstipohjaisten kysymysten näyttämisestä ja lähettämisestä. Malliosio sisältää luettelon kysymyksistä, joissa jokaisessa on tekstialue käyttäjän syöttöä varten ja merkkilaskuri. Käyttäjän syöttö on rajoitettu 240 merkkiin. Skriptiosassa määritellään komponentin tiedot, joihin kuuluvat kysymykset, textAnswers ja characterCount. Watch-ominaisuudella varmistetaan, että kun kysymyksiä päivitetään, textAnswers- ja characterCount-kohtien vastaavat merkinnät alustetaan. Mountingin yhteydessä fetchQuestions-metodi hakee kysymykset backendista ja määrittää ne kysymyksiin. submitAnswers-metodi kerää ja lähettää käyttäjän vastaukset toiseen backend-API-päätepisteeseen ja näyttää vastausviestin sisältävän hälytyksen. ClearTextAnswers-metodi nollaa kaikki tekstialueet ja laskurit lähettämisen jälkeen. UpdateCharacterCount-metodi seuraa kuhunkin kysymykseen syötettyjen merkkien määrää. Lisäksi sanitize-metodi käyttää DOMPurify-menetelmää XSS-hyökkäysten estämiseksi puhdistamalla käyttäjän syötteet ja kysymystekstin. Komponenttiä pitkille tekstivastauksille ei erikseen kehitetty, mutta se olisi helppoa tehdä muokkaamalla tähän komponenttiin oikeat backendin reitit ja tietokannan taulut sekä suurimman sallitun merkkimäärän.

```
export default {
  name: 'TextQuestions',
  data() {
    return {
      questions: {},
      textAnswers: {},
      characterCount: {},
    };
  },
  watch: {
    questions: {
      immediate: true,
      handler(newQuestions) {
        this.textAnswers = Object.fromEntries(
          Object.keys(newQuestions).map((questionId) => [questionId, ''])
        );
        this.characterCount = Object.fromEntries(
          Object.keys(newQuestions).map((questionId) => [questionId, 0])
        );
      },
    },
  },
  mounted() {
    this.fetchQuestions();
  },
  methods: {
    fetchQuestions() {
      fetch("http://127.0.0.1:5000/api/short_text_questions")
        .then((response) => response.json())
        .then((data) => {
          this.questions = data;
        })
        .catch((error) => {
          console.error("Error fetching questions:", error);
        });
    },
    submitAnswers() {···
    },
    clearTextAnswers() {···
    },
    updateCharacterCount(questionId) {
      this.characterCount[questionId] = this.textAnswers[questionId].length;
    },
    sanitize(content) {
      return DOMPurify.sanitize(content);
    },
  },
```

Kuvio 24. ShortText-komponentti.

Viimeinen kehitetty komponentti oli tekstikysymysten noutamiseen, näyttämiseen ja äänestämiseen tarkoitettu QuestionAnswers (ks. kuvio 25). QuestionAnswers, näyttää luettelon tiettyyn kysymystunnukseen liittyvistä arvoista ja antaa käyttäjien äänestää näistä arvoista. Esimerkissä haetaan kysymyksiä ID:llä 4, mutta toteutuksessa tämä on helppo muokata tarpeen mukaan. Komponentti noutaa arvot API-päätepisteestä luomisen yhteydessä ja alustaa äänten laskentamäärän, jolla seurataan kunkin arvon ääniä. Malliosio sisältää säiliön, jossa kukin arvo luetellaan tyylitellyssä laatikossa ja sen hetkinen äänimäärä. Jokaista laatikkoa voi napsauttaa, ja laatikkoa napsauttamalla rekisteröidään kyseisen arvon ääni. Äänestysmenetelmä varmistaa, että vain yksi arvo voidaan valita kerrallaan, tarkistamalla, jos samaa laatikkoa napsautetaan uudelleen, ja päivittämällä äänimäärän vastaavasti. Valittu laatikko erottuu visuaalisesti eri taustavärillä ja kehyksellä. Komponentti käyttää muotoilussa scoped CSS:ää, mikä varmistaa, että tyylit on eristetty vain tähän komponenttiin. Äänestystoiminto on kuitenkin vajavainen toiminnaltaan, sillä vastausten saamat äänet eivät tallennu minnekään vaan katoavat, kun sovellus suljetaan.

```
export default {
  name: 'QuestionAnswers',
  data() {
    return {
      tableName: 'short_text',  // Specify your table name here
      questionId: 4,  // Specify your Question ID here
      values: [],
      votes: [],  // Array to store votes for each value
      selectedValueIndex: null  // Index of the selected value
    };
  },
  methods: {
    fetchValues() {
      fetch(`http://127.0.0.1:5000/values?table=${this.tableName}&questionId=${this.questionId}`)
        .then(response => response.json())
        .then(data => {
          this.values = data;
          this.votes = Array(data.length).fill(0);  // Initialize votes array
        })
        .catch(error => {
          console.error('Error fetching values:', error);
        });
    },
    vote(index) {
      if (this.selectedValueIndex === index) {
        return;  // Do nothing if the same box is clicked again
      }
      if (this.selectedValueIndex !== null) {
        this.votes[this.selectedValueIndex] -= 1;  // Decrement the previous vote
      }
      this.selectedValueIndex = index;
      this.votes[index] += 1;  // Increment the vote count for the selected value
    }
  },
  created() {
    this.fetchValues();
  }
};
```

Kuvio 25. QuestionAnswers-komponentti.

4.5 Docker

Sovelluksen kehityksen viimeinen vaihe oli kontitus dockerilla. Projektin rakenne juurikansiossa muokattiin niin, että kaikki backendin tiedostot olivat samassa kansiossa nimeltä backend ja kaikki frontendin tiedostot samassa kansiossa nimeltä frontend. Molempiin kansioihin luotiin omat Dockerfilet ja projektin juureen luotiin docker-compose.yml-tiedosto (ks. kuvio 26) molempien palveluiden hallintaa varten. Backend-palvelu rakentaa kuvansa ./backend-hakemistosta ja yhdistää isännän portin 5000 kontin porttiin 5000. Se asettaa FLASK_ENV-ympäristömuuttujan arvoksi development ja asentaa ./backend-hakemiston isännältä kontin /app-osioon, mikä mahdollistaa koodin muutokset reaaliaikaisesti. Frontend-palvelu rakentaa kuvansa ./frontend-hakemistosta, liittää isännän portin 8080 kontin porttiin 8080 ja asentaa vastaavasti ./frontend-hakemiston isännän portista /app konttiin. Tämä asetelma mahdollistaa web-sovelluksen kehittämisen erillisillä frontend- ja backend-palveluilla, jotka kumpikin toimivat omassa kontissaan ja mahdollistavat reaaliaikaiset koodipäivitykset.

```yaml
services:
  backend:
    build: ./backend
    ports:
      - "5000:5000"
    environment:
      - FLASK_ENV=development
    volumes:
      - ./backend:/app

  frontend:
    build: ./frontend
    ports:
      - "8080:8080"
    volumes:
      - ./frontend:/app
```

Kuvio 26. Docker Compose -tiedosto.

Backendin Dockerfile (ks. kuvio 27) on suunniteltu luomaan Docker-kontti Python Flask -sovellukselle. Se alkaa määrittämällä python:3.9-slim peruskuvaksi, joka tarjoaa kevyen Python 3.9 -ympä-

ristön. Kontin sisällä olevaksi työhakemistoksi asetetaan /app, jossa sovelluskoodi sijaitsee. Aiemmin luotu requirements.txt-tiedosto, jossa luetellaan sovelluksen vaatimat Python-riippuvuudet, kopioidaan tähän hakemistoon, ja riippuvuudet asennetaan käyttämällä pip install --no-cache-dir -r requirements.txt, mikä varmistaa puhtaan asennuksen ilman välimuistia, jotta image-koko pysyy pienenä. Tämän jälkeen loput sovelluskoodista kopioidaan /app-hakemistoon. Kontti määritetään avaamaan portti 5000, jossa Flask-sovellus suoritetaan. Ympäristömuuttujat asetetaan Flask-sovelluksen aloituspisteen (new.py) määrittämiseksi, palvelimen saamiseksi saataville mistä tahansa IP-osoitteesta (0.0.0.0) ja ympäristön asettamiseksi kehitystilaan. Lopuksi määritetään oletuskomento, jolla sovellus käynnistetään: flask run. Tämän konfiguraation avulla Flask-sovellus voidaan helposti rakentaa, ottaa käyttöön ja ajaa Docker-kontissa.

```
# Use the official Python image from the Docker Hub
FROM python:3.9-slim

# Set the working directory
WORKDIR /app

# Copy the requirements.txt file and install the dependencies
COPY requirements.txt requirements.txt
RUN pip install --no-cache-dir -r requirements.txt

# Copy the rest of the application code
COPY . .

# Expose the port on which the app runs
EXPOSE 5000

# Set the environment variables if any
ENV FLASK_APP=new.py
ENV FLASK_RUN_HOST=0.0.0.0
ENV FLASK_ENV=development
# Run the application
CMD ["flask", "run"]
```

Kuvio 27. Backendin Dockerfile.

Frontendin Dockerfile on suunniteltu luomaan Docker-kontti Node.js-sovellukselle. Se käyttää pohjakuvana node:16-alpinea, joka tarjoaa kevyen Node.js 16 -ympäristön. Kontin sisällä olevaksi työhakemistoksi asetetaan /app, jossa sovelluskoodi sijaitsee. Tähän hakemistoon kopioidaan

package.json- ja package-lock.json-tiedostot, joissa luetellaan projektin riippuvuudet, ja riippuvuudet asennetaan npm install -ohjelmalla. Sitten loput sovelluskoodista kopioidaan /app-hakemistoon. Sovellus rakennetaan npm run build -ohjelmalla, jolloin se valmistellaan tuotantoa varten. Kontti määritetään avaamaan portti 8080, jossa sovellus suoritetaan. Lopuksi määritetään oletuskomento sovelluksen palvelemiseen npm run serve, joka käynnistää sovelluksen käyttämällä package.json-tiedostossa määriteltyä serve-skriptiä. Tämä asetus varmistaa, että Node.js-sovellus voidaan helposti rakentaa, ottaa käyttöön ja ajaa Docker-kontissa.

```
# Use the official Node.js image from the Docker Hub
FROM node:16-alpine

# Set the working directory
WORKDIR /app

# Copy the package.json and package-lock.json files
COPY package*.json ./

# Install the dependencies
RUN npm install

# Copy the rest of the application code
COPY . .

# Build the application
RUN npm run build

# Expose the port on which the app runs
EXPOSE 8080

# Serve the application
CMD ["npm", "run", "serve"]
```

Kuvio 28. Frontendin Dockerfile.

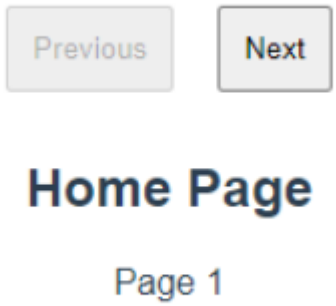
Dockerilla kontitettu sovellus käynnistyy yhdellä komennolla docker-compose up –build. Tämä komento rakentaa docker-compose.yml-tiedostossa määritellyt Docker imaget palveluille ja käynnistää kontit näiden imagejen pohjalta. Lokaaliympäristössä kehittäessä tietokantaan yhteydenottamiseen käytettyyn db_configiin (ks. kuvio 29) piti muokata DB_HOST-muuttujan osoite paikallisesta IP-osoitteesta muotoon host.docker.internal, sillä tietokantaan ei muuten pystytty saamaan yhteyttä.

```
DB_HOST=host.docker.internal
DB_USER=username
DB_PASSWORD=password
DB_DATABASE=db_name
```

Kuvio 29. DB_CONFIG .env-tiedostossa.

5 Tulokset

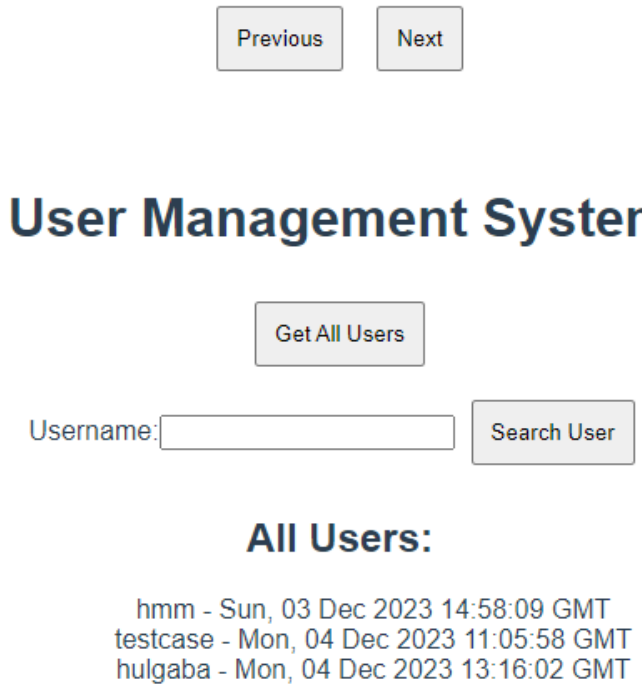
Yksi projektin tavoitteista oli kehittää putki, jonka läpi osallistujat pelaavat. Tämä onnistuttiin toteuttamaan. Esimerkkitoteutuksessa jokaista kysymystyyppiä on käytetty vain kerran, mutta todellisessa toteutuksessa frontendin komponentteja pystytään vapaasti yhdistelemään haluttua lopputulosta varten. Sovelluksen käynnistämisen jälkeen se pyörii selaimen sisällä, jossa osallistujat pystyvät vuorovaikuttamaan sivuilla olevan sisällön kanssa. Putki alkaa kotisivusta (ks. kuvio 30) ja koska tämä on putken ensimmäinen sivu, ”Edellinen”-painike on poistettu käytöstä. Seuraavalla sivulla on lomake uuden käyttäjän luontiin (ks. kuvio 31). Käyttäjänimi on asetettu pakolliseksi kentäksi täyttää, muuten lomake ei salli uuden käyttäjän lisäystä tietokantaan. Jos käyttäjänimi on jo käytössä, selain näyttää hälytyksen, jonka viestissä ilmoitetaan käyttäjänimen olevan varattu.



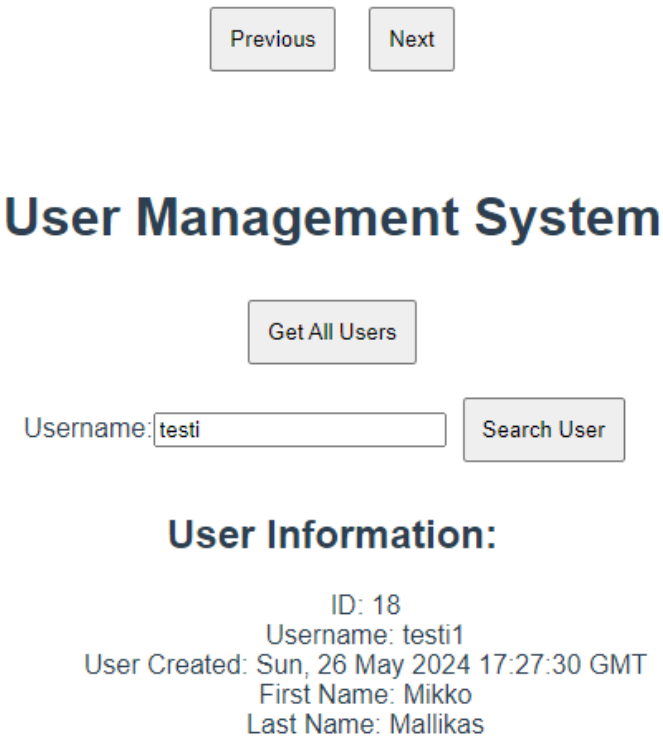
Kuvio 30. Kotisivu.

Kuvio 31. Lomake uuden käyttäjän luontiin.

Seuraavalla sivulla on järjestelmä käyttäjätietojen hakuun tietokannasta. Painamalla ”Get All Users”-painiketta (ks. kuvio 32), sovellus hakee jokaisen käyttäjän tiedot ja esittää ne listassa käyttäjälle. Esitetyt tiedot sisältävät käyttäjänimen ja käyttäjän luomisen ajankohdan. Tekstikenttään kirjoittamalla ja painamalla ”Search User”-painiketta (ks. kuvio 33), käyttäjä pystyy hakemaan yhden tietyn käyttäjän tietoja. Kysely palauttaa ensimmäisen listassa olevan käyttäjänimen, joka vastaa käyttäjän syötettä. Jos esimerkiksi tietokannassa on käyttäjät ”testi1” ja ”testi2”, kirjoittamalla hakukenttään ”testi”, käyttäjälle palautetaan käyttäjän ”testi1” tiedot. Yksittäisen käyttäjän haussa esitetyt tiedot ovat yksityiskohtaisemmat kuin kaikkien käyttäjien haussa. Nämä tiedot sisältävät käyttäjän ID:n, käyttäjänimen, luomisen ajankohdan sekä etu- ja sukunimen. Jos käyttäjää ei löydy tietokannasta, sovellus ei palauta mitään. Tässä olisi ollut hyvä olla jokin virheilmoitus, mutta se jäi erehdyksen vuoksi kehittämättä.



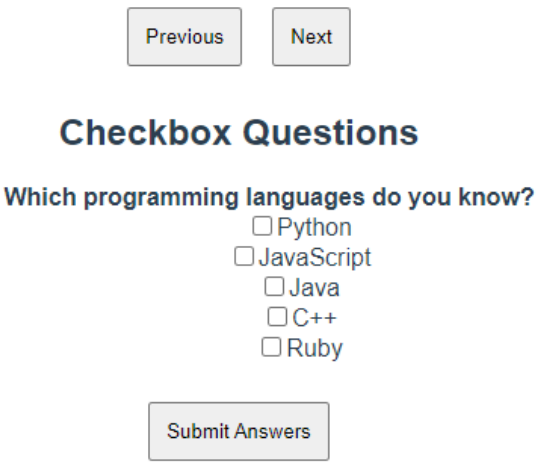
Kuvio 32. Kaikkien käyttäjien haku.



Kuvio 33. Yhden käyttäjän haku.

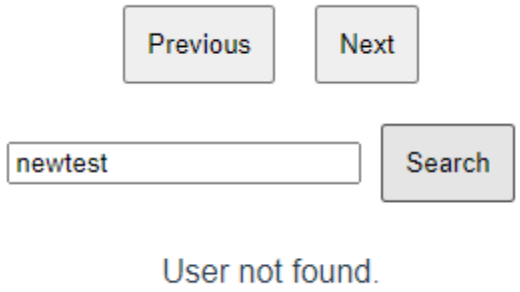
Seuraavalla sivulla on kaksi radiokysymystä ja sitä seuraavalla sivulla on yksi monivalintakysymys. Radiokysymyksissä (ks. kuvio 34) käyttäjä voi valita yhden vaihtoehdon kysymystä kohden. ”Submit Answers”-painiketta painaessa vastaukset lähetetään tietokantaan ja valitut vaihtoehdot tyhjennetään selaimesta. Monivalintalomake (ks. kuvio 35) toimii muuten samalla tavalla, mutta käyttäjä pystyy valitsemaan niin monta vastausvaihtoehtoa kuin tahtoo.

Kuvio 34. Radiokysymyslomake.

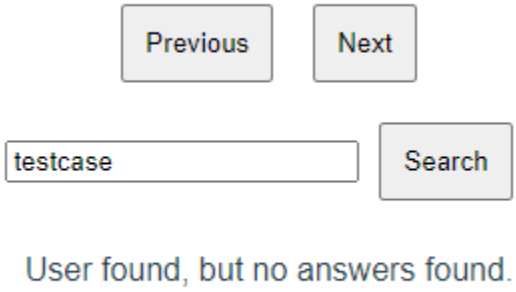


Kuvio 35. Monivalintakysymyslomake.

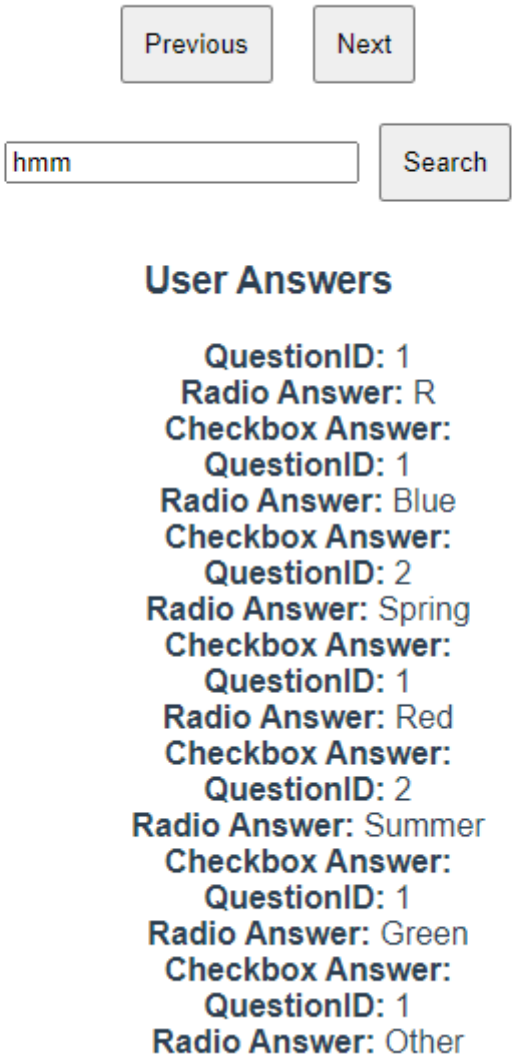
Tämän jälkeen käyttäjälle esitetään järjestelmä käyttäjien vastauksien hakemiseen. Käyttäjä syöttää tekstikenttään merkkijonon ja ”Search”-painiketta painaessa sovellus hakee käyttäjiä syötteen perusteella. Tässä lomakkeessa käyttäjänimen pitää olla tismalleen niin kuin se on kirjattuna tietokannassa. Jos käyttäjä syöttää hakukenttään ”testi”, lomake palauttaa viestin ”User not found” (ks. kuvio 36), vaikka ”testi1” olisikin tietokannassa. Jos käyttäjänimi löytyy, mutta yhtäkään vastausta kysymyksiin ei löydy, sovellus palauttaa ”User found, but no answers found” (ks. kuvio 37). Jos käyttäjänimi löytyy ja käyttäjällä on myös vastauksia kysymyksiin, sovellus palauttaa listassa kysymysID:n ja vastauksen arvon jokaisesta vastatusta kysymyksestä (ks. kuvio 38).



Kuvio 36. Jos käyttäjää ei löydy tietokannasta.

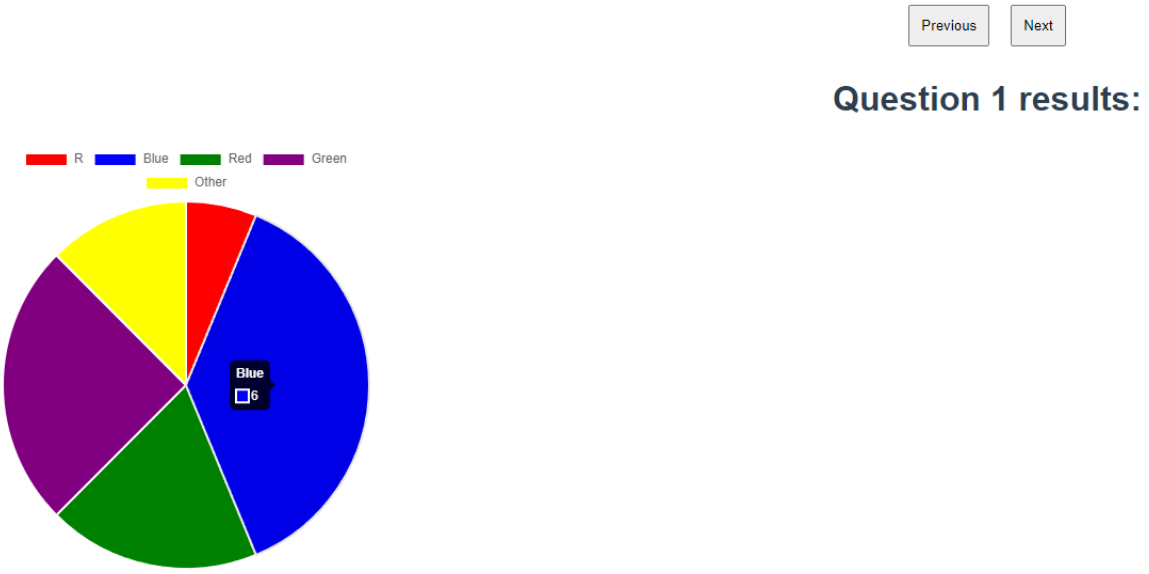


Kuvio 37. Jos käyttäjä löytyy ilman vastauksia.



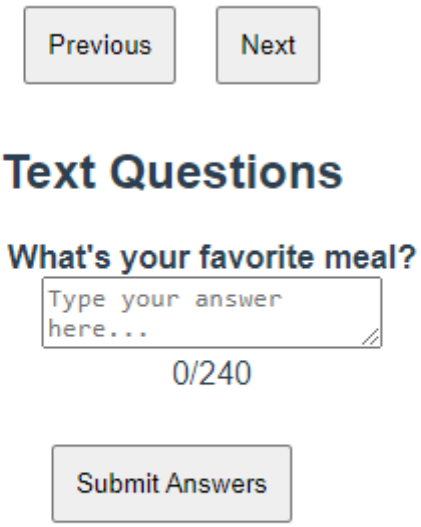
Kuvio 38. Käyttäjän kaikki vastaukset radio- ja monivalintatauluissa.

Seuraavalla sivulla käyttäjälle näytetään vastausdatasta visualisoitu piirakkakaavio (ks. kuvio 39). Piirakkakaavio listaa, kuinka monta kertaa kukin vastausvaihtoehto esiintyy tietokannassa. Esimerkissä kaavio visualisoi ensimmäisen radiokysymyksen vastaukset. Siirtämällä hiiren kursorin piirakan sektoreiden päälle, käyttäjälle näytetään kyseisen sektorin nimi ja kuinka monta esiintymää sillä on tietokannassa. Todellisessa toteutuksessa tämä komponentti on erittäin joustava ja uudelleenkäytettävä. KysymysID:n ja halutun taulun nimen muokkaamalla koodissa on mahdollista saada se esittämään haluttua kysymystä. Sektoreiden värejä on mahdollista lisätä ja muokata tarpeen mukaan.

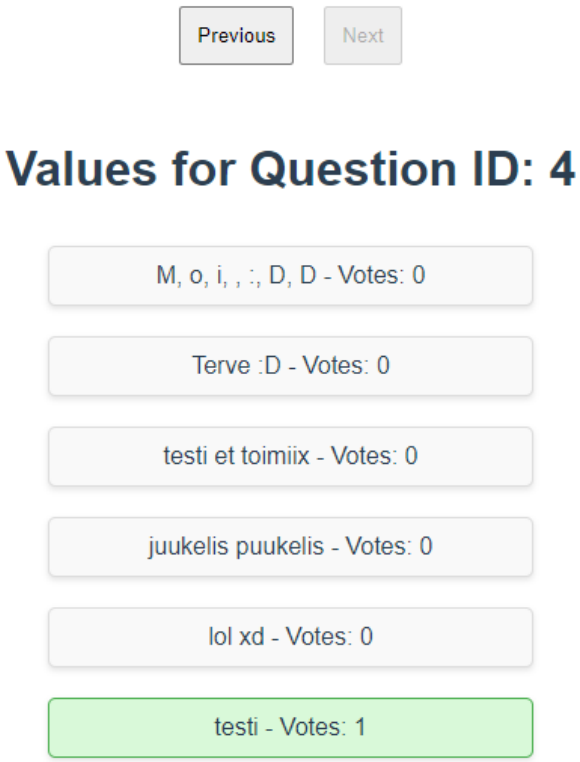


Kuvio 39. Piirakkakaavio kysymysID 1:n vastauksista.

Seuraavalla sivulla käyttäjä voi vastata tekstikysymykseen. Kyseessä on lyhyt tekstikysymys ja suurin sallittu merkkimäärä on 240. Tekstilaatikon alapuolella on reaaliajassa syötettyjen merkkien määrää päivittävä laskuri (ks. kuvio 40). ”Submit Answers”-painiketta painaessa vastaus lähetetään tietokantaan ja tekstilaatikko tyhjennetään. Viimeisellä sivulla käyttäjälle esitetään laatikoissa edellisen sivun kysymyksen kaikki vastaukset (ks. kuvio 41). Käyttäjä pystyy äänestämään vastauksista yhtä klikkaamalla jotakin laatikkoa. Äänestetty laatikko on selvästi korostettu muista. Jokaisessa laatikossa on tekstiarvon jälkeen laskuri sen saaneille äänille. Nämä äänestykset eivät kuitenkaan tallennu minnekään, joten niillä ei tässä vaiheessa ole mitään hyödyllistä käyttöä. Koska tämä sivu on viimeinen, ”Next”-painike on poistettu käytöstä.



Kuvio 40. Lomake tekstikysymyksille.



Kuvio 41. Tekstikysymys ID:n 4 vastaukset esitettynä vuorovaikutteisina laatikoina.

6 Pohdinta

Opinnäytetyön pääasiallisiin tavoitteisiin päästiin vain osittain. Valmiiksi saatiin kokonainen putki, jonka läpi osallistujat voivat pelata. Pois lukien Login, kaikki toteutuneet komponentit frontendissa ja niitä vastaavat reitit backendissa toimivat toivotulla tavalla. Projekti saatiin myös onnistuneesti kontitettua, tehden pystytyksestä uusissa ympäristöissä helppoa. Käyttövalmiin sovelluksen sijasta projektissa saavutettiin ns. ”proof of concept”-malli. Esimerkkiputki ei kuvaa kovinkaan hyvin, miltä mahdollinen lopullinen työpaja voisi näyttää. Esimerkin tarkoitus oli lähinnä demonstroida jokaisen kehitetyn komponentin toiminnallisuutta. Käyttäjä- ja kysymystietojen hakemisten mallin ei kuuluisi varsinaisessa toteutuksessa olla kysymysten joukossa, vaan sovellukseen luultavasti pitäisi kehittää useita eri putkia, joista käyttäjä pystyy valitsemaan päävalikossa. Ensimmäisenä esiin nousevana puutteena on todennusjärjestelmän poisjääminen. Nykytilassa sovelluksesta puuttuu kaikkein perustavanlaatuisin tietoturvaominaisuus, mikä periaatteessa mahdollistaa kenen tahansa ulkopuolisenkin pääsevän käsiksi järjestelmään. Käytännössä tämä ei kuitenkaan ole hirveän suuri riski, sillä sovellusta ei koskaan aiottu julkaista julkisessa verkossa ollenkaan. Järjestetyssä työpajassa sovellus pyörisi työpajan järjestäjän sisäverkossa vain työpajan ajan. Todennus olisi silti hyvä olla lisäturvan vuoksi ja sen kehittäminen valmiiksi onkin jatkokehityksen ensimmäinen prioriteetti.

Toinen esiin nouseva asia on työn nimestä huolimatta työskentelyn pelillistämisen vähäisyys. Toimeksiantaja ei erikseen määritellyt, mitä he pelillistämisellä varsinaisesti tarkoittivat, vaan sen keksiminen oli yksi minun työtehtävistäni. Toiminnallisuuksien kehittämiseen kuitenkin kului huomattavasti enemmän aikaa kuin osattiin odottaa, joten luovien ratkaisujen keksiminen pelillisyyteen jäi taka-alalle, eikä siihen kehitetty paljoa. Ainoat toteutuneet elementit tästä ovat vastausdataa visualisoiva piirakkakaaviomalli ja tekstivastausten äänestämisen mahdollistava malli, vaikka näistä jälkimmäinen jäikin vajavaiseksi. Nykyisessä muodossaan sovellus ei merkittävästi eroa jo olemassa olevista kyselylomakkeiden luontiin käytettävistä ohjelmista. Jatkokehityksessä pelillisyyttä on syytä kehittää osallistujille innostavammaksi ja vuorovaikutteisemmaksi. Tekstivastausten äänestämisen toimivaksi saaminen on hyvä aloituskohde.

Kolmas puute ei liity suoraan sovelluksen toiminnallisuuteen, mutta sovelluksesta puuttuu tyylittelyt lähes kokonaan. Tämän vuoksi käyttäjän selaimessa näkemät lomakkeet ovat hyvin yksinkertai-

sia ulkonäöltään. Toimeksiantajalla oli suunnitelmia ja toiveita, miten he halusivat sovelluksen tyyliteltävän ja keskustelua käytiin mm. erilaisten CSS-kirjastojen implementoinnista projektin loppuvaiheessa. Kaikki aika kuitenkin kului toiminnallisuuksiin, joten tyylittelystä oli valitettavasti pakko luopua aikarajoitteiden vuoksi.

Tietokanta olisi ollut hyvä siirtää lokaaliympäristöstä pyörimään jollekin ulkoiselle palvelimelle, johon sovellus ottaisi yhteyden. Jos sovellusta lähdettäisiin pystyttämään kehitysympäristön ulkopuolella olevaan ympäristöön, tietokanta pitäisi erikseen luoda, jotta sovellusta voitaisiin käyttää. Tietokanta olisi voitu pystyttää esimerkiksi verkkoon, tai tietokannasta olisi voitu tehdä oma Docker-kontti. Jatkokehityksessä nämä on hyvä ottaa huomioon.

Frontendin nykyinen toteutus on huomattavan kömpelö ja raskas. Pienimuotoisessa toteutuksessa siitä ei ole haittaa, että jokainen sivu täytyy olla oma tiedostonsa, mutta se tekee sovelluksesta hyvin huonosti skaalautuvan. Jos sovelluksella tahdottaisiin toteuttaa esimerkiksi 50 sivun työpaja, frontendin tiedostojen ylläpito olisi erittäin hankalaa ja sovelluksesta tulisi myös hyvin raskas tiedostokoon puolesta. Jatkokehityksessä on syytä tutkia kevyempiä ja yksinkertaisempia ratkaisuja useamman sivun toteutukselle, kuten esimerkiksi mallikomponenttien uudelleenkäyttäminen dynaamisesti sivun tietoja muokaten tarpeen mukaan kovakoodattujen komponenttien sijasta.

Lähteet

Butler, B. 2013. PaaS Primer: What is platform as a service and why does it matter? Artikkeli Networkworldin verkkosivuilla. Viitattu 1.3.2024. https://www.networkworld.com/article/669129/cloud-computing-paas-primer-what-is-platform-as-a-service-and-why-does-it-matter.html.

Cromwell, V. 2016. Evan You. Evan Youn haastattelu, julkaistu Between The Wires uutistoimiston verkkosivuilla. Viitattu 29.2.2024. https://web.archive.org/web/20170603052649/https://betweenthewires.org/2016/11/03/evan-you/.

Docker frequently asked questions (FAQ). N.d. Usein kysytyt kysymykset Dockerista Dockerin omilla sivuilla. Viitattu 1.3.2024. https://docs.docker.com/engine/faq/#what-does-docker-technology-add-to-just-plain-lxc.

Flask 2.0.x Documentation. N.d. Flaskin dokumentaatio Pallets Projectsin omilla verkkosivuilla. Viitattu 4.2.2024. https://flask.palletsprojects.com/en/2.0.x/.

Flask 3.0.x Documentation. N.d. Flaskin dokumentaatio Pallets Projectsin omilla verkkosivuilla. Viitattu 4.2.2024. https://flask.palletsprojects.com/en/3.0.x/.

Introduction. 2024. Esittely Vue.js:n ominaisuuksista Vuen omilla sivuilla. Viitattu 29.2.2024. https://vuejs.org/guide/introduction.html.

JavaScript Documentation. 2024. JavaScriptin dokumentaatio Mozillan verkkosivuilla. Viitattu 28.2.2024. https://developer.mozilla.org/en-US/docs/Web/JavaScript.

Kuchling, A.M. 2000. PEP 206 – Python Advanced Library. PEP-dokumentti jossa ehdotetaan parannuksia Python-ohjelmointikieleen. Viitattu 4.2.2024. https://peps.python.org/pep-0206/.

Kuhlman, D. 2012. A Python Book: Beginning Python, Advanced Python, and Python Exercises. Pythonin itseopiskeluun tarkoitettu kirja. Viitattu 4.2.2024. https://web.archive.org/web/20120623165941/http:/cutter.rexx.com/~dkuhlman/python_book_01.html.

Meet the Team. N.d. Vue.js:n kehitystiimin esittely Vuen omilla verkkosivuilla. Viitattu 29.2.2024. https://vuejs.org/guide/introduction.html.

MySQL 8.0 Reference Manual. 2024. MySQL:n dokumentaatio MySQL:n omilla Oraclen ylläpitämillä verkkosivuilla. Viitattu 4.2.2024. https://dev.mysql.com/doc/refman/8.0/en/.

O'Gara, M. 2013. Ben Golub, Who Sold Gluster to Red Hat, Now Running dotCloud. Artikkeli, julkaistu kirjoittajan omilla verkkosivuilla. Viitattu 1.3.2024. https://web.archive.org/web/20190913100835/http://maureenogara.sys-con.com/node/2747331.

Peterson, B. 2020. Python 2.7.18, the last release of Python 2. Blogiteksti, julkaistu Python Insider -blogissa. Viitattu 4.2.2024. https://pythoninsider.blogspot.com/2020/04/python-2718-last-release-of-python-2.html.

Python 3.12.2 Documentation. 2024. Pythonin 3.12.2 version dokumentaatio Pythonin omilla verkkosivuilla, tarkkaan ottaen usein kysytyt kysymykset -osio. Viitattu 4.2.2024. https://docs.python.org/3/faq/general.html.

Ronacher, A. 2010. The Pocoo Project. Pocoo-projektin omat verkkosivut. Viitattu 4.2.2024. https://web.archive.org/web/20180313200311/http:/www.pocoo.org/

Ronacher, A. 2011. Opening the Flask. Flaskin kehittäjän kirjoittama PDF-tiedosto. Viitattu 4.2.2024. https://web.archive.org/web/20160604162342/http:/mitsuhiko.pocoo.org/flask-pycon-2011.pdf

Ronacher, A. 2016. Hello Pallets Users. Flaskin kehittäjän kirjoittama blogiteksti, julkaistu Pallets Projectsin omilla verkkosivuilla. Viitattu 4.2.2024. https://palletsprojects.com/blog/hello/.

Rossum, G. 2009. A Brief Timeline of Python. Blogiteksti, joka listaa Pythonin versioiden aikajanan. Viitattu 4.2.2024. https://python-history.blogspot.com/2009/01/brief-timeline-of-python.html.

Usage statistics of JavaScript as client-side programming language on websites, 2024. Tilastokokoelma JavaScriptiä käyttävistä verkkosivuista. Viitattu 28.2.2024. https://w3techs.com/technologies/details/cp-javascript.

What is a Container? N.d. Esittely Dockerille ja konttiteknologialle Dockerin omilla verkkosivuilla. Viitattu 1.3.2024. https://www.docker.com/resources/what-container/.

What is Vue.js? N.d. Verkkosivu oppimistarkoituksiin, joka selittää Vue.js:n ominaisuuksia. Viitattu 29.2.2024 https://www.w3schools.com/whatis/whatis_vue.asp.

You, E. 2014. First Week of Launching Vue.js. Blogikirjoitus Evan Youn omalla blogisivustolla. Viitattu 29.2.2024. https://blog.evanyou.me/2014/02/11/first-week-of-launching-an-oss-project/index.html.