Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій

Кафедра програмних систем і технологій

УДК 004.4

На правах рукопису

ВИПУСКНА КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА

Тема: “Розроблення онлайн сервісу для організації дозвілля та спілкування”

Спеціальність – 121 “Інженерія програмного забезпечення”

ПОЯСНЮВАЛЬНА ЗАПИСКА

БР.ІПЗ – 22.00.00.000

Студент

ІПЗ-44_____/Ярослав ДМИТРИК/

Науковий керівник

к. ф.-м. н., доц. _____/Ольга СУПРУН/

Консультант

з питань нормоконтролю

фахівець_____/Тамара ЧАПОВСЬКА/

Допускається до захисту

Завідувач кафедри

д.т.н., проф._____/Олексій БИЧКОВ/

Київ – 2021

Рішенням Екзаменаційної комісії

випускна кваліфікаційна робота студента

_____

захищена з оцінкою

_____

Голова Екзаменаційної комісії

д. т. н., проф. Віктор ВИШНІВСЬКИЙ

Київський національний університет імені Тараса Шевченка
Факультет інформаційних технологій
Кафедра програмних систем і технологій
Спеціальність 121 “Інженерія програмного забезпечення”

ЗАТВЕРДЖЕНО
Зав. кафедри програмних систем і технологій
_____(Олексій БИЧКОВ)
„_____”_____ 2021 р.

ЗАВДАННЯ
НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ
СТУДЕНТУ

Дмитрику Ярославу Івановичу
(прізвище, ім’я, по батькові)

1. Тема випускної кваліфікаційної бакалаврської роботи “Розроблення онлайн сервісу для організації дозвілля та спілкування”, керівник роботи Ольга СУПРУН, к. ф.-м. н., доц. затверджені на засіданні кафедри програмних систем і технологій, протокол №6 від «11» листопада 2020р.
2. Строк здачі студентом закінченої роботи _____
3. Вихідні дані до роботи: теоретичні відомості щодо розробки веб-додатків
4. Зміст пояснювальної записки (перелік питань, що їх належить розробити)

1. Аналіз і огляд предметної галузі.
2. Проєктування додатку.
3. Особливості реалізації додатку.
4. Опис здобутих результатів.

5. Перелік графічного матеріалу (з точним зазначенням обов’язковихкреслень)

1. Схематичне порівняння контейнеризації та віртуалізації (рис.1.1, стор. 22)
2. Схематичне зображення архітектури «Клієнт-сервера» (рис.1.2., стор. 25)
3. Схематичне зображення архітектури додатку (2.1., стор. 28)
4. Схема бази даних додатку (2.2., стор. 30)

6. Консультанти з роботи із зазначенням розділів роботи, що їх стосуються

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
РОЗДІЛ 1	Ольга СУПРУН		
РОЗДІЛ 2	Ольга СУПРУН		
РОЗДІЛ 3	Ольга СУПРУН		
РОЗДІЛ 4	Ольга СУПРУН		

7. Дата видачі завдання «05» листопада 2020 р.

Керівник _____ (Ольга СУПРУН)

Завдання прийняв до виконання _____(Ярослав ДМИТРИК)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назви етапів бакалаврської роботи	Термін виконання етапів роботи	Примітка
1	Огляд архітектурних підходів проектування веб-додатків	10.11.2020	виконано
2	Огляд мов програмування для реалізації серверної так клієнтської частини	24.12.2020	виконано
3	Огляд та вибір бази даних	16.01.2021	виконано
4	Проектування веб-додатку	05.02.2021	виконано
5	Реалізація веб-додатку	03.05.2021	виконано
6	Тестування веб-додатку	10.05.2021	виконано
7	Оформлення пояснювальної записки	06.06.2021	виконано

Студент – бакалавр_____ (Ярослав ДМИТРИК)

Керівник роботи _____ (Ольга СУПРУН)

АНОТАЦІЯ

Випускна кваліфікаційна бакалаврська робота: 61с., 10 рис., 1 додат., 14 джерел.

Тема: Розроблення онлайн сервісу для організації дозвілля та спілкування.

Об’єкт дослідження: технології для розробки онлайн сервісів.

Мета роботи: оптимізувати організаційну діяльність осіб, що здійснюють пошук людей для спільних занять спортом.

Предмет дослідження: онлайн сервіс для організації дозвілля та спілкування.

Результати дослідження:

Досліджено сучасні технології та підходи для розробки веб-додатків, а саме React (мова програмування TypeScript) для клієнтської частини, та Flask (мова програмування Python 3) для серверної частини, також було досліджено архітектурний підхід REST, який був взятий за основу сервісу.

Висновок:

Розроблено онлайн сервісу пошуку людей, для організації дозвілля та спілкування Додаток отримав простий та зрозумілий інтерфейс, яким зручно користуватись. В порівнянні з конкурентами, розроблений сервіс має мапу, та великий вибір фільтрів, що допомагає пришвидшити пошук.

ОНЛАЙН СЕРВІС, МОВА ПРОГРАМУВАННЯ PYTHON, МОВА СТРУКТУРОВАНИХ ЗАПИТІВ SQL, КОНТЕЙНЕРИЗАЦІЯ, DOCKER, REST API

АННОТАЦИЯ

Выпускная квалификационная бакалаврская работа: 61с., 10 рис., 1 прил., 14 источников.

Тема: Разработка онлайн сервиса для организации досуга и общения.

Объект исследования: технологии для разработки онлайн сервисов.

Цель работы: оптимизировать организационную деятельность лиц, осуществляющих поиск людей для совместных занятий спортом.

Предмет исследования: онлайн сервис для организации досуга и общения.

Результаты исследования:

Проведено исследование современных технологии и подходы для разработки веб-приложений, а именно React (язык программирования TypeScript) для клиентской части, и Flask (язык программирования Python 3) для серверной части, также было исследовано архитектурный подход REST, который был взят за основу сервису.

Вывод:

Разработан онлайн сервис поиска людей, для организации досуга и общения Приложение получил простой интерфейс, которым удобно пользоваться. По сравнению с конкурентами, разработанный сервис имеет карту, и большой выбор фильтров, которые помогает ускорить поиск.

ОНЛАЙН СЕРВИС, ЯЗЫК ПРОГРАММИРОВАНИЯ PYTHON, ЯЗЫК СТРУКТУРИРОВАННЫХ ЗАПРОСОВ SQL, КОНТЕЙНЕРИЗАЦИИ, DOCKER, REST API

ANNOTATION

Graduation qualifying bachelor's thesis: 61 pages, 10 images, 1 application, 14 sources.

Topic: Development of an online service for organizing leisure and communication.

Object of study: technologies for the development of online services.

The goal of the work: optimize the organizational activities of persons searching for people for joint sports.

Subject of study: development of an online service for organizing leisure and communication.

Results of the research:

Modern technologies and approaches for developing web applications were investigated, namely React (TypeScript programming language) for the frontend, and Flask (Python 3 programming language) for the backend; the architectural style REST was also investigated, which was taken as a basis for the service.

Conclusion:

An online service for finding people was developed for organizing leisure and communication. The application has a simple interface that is convenient to use. Compared to competitors, the developed application has a map and a large selection of filters that help speed up the search.

ONLINE SERVICE, PYTHON PROGRAMMING LANGUAGE, STRUCTURED QUERY LANGUAGE, SQL, CONTAINERATION, DOCKER, REST API

ЗМІСТ

ВСТУП

Актуальність роботи

Технології відіграють величезну роль у нашому повсякденному житті, починаючи від найпростіших програм і закінчуючи найбільш новаторськими винаходами. Кожен веб-сайт або програмне забезпечення, з яким ми стикаємось в Інтернеті, було створено веб-розробником.

Веб-розробка на сьогоднішній день є найпопулярнішим напрямом в світі технологій. На даний момент 55% розробників працюють в сфері веб-розробки [1], і даний показник постійно зростає.

Мета і задачі дослідження

Метою роботи є оптимізація організаційної діяльності осіб, що здійснюють пошук людей для спільних занять спортом. Для досягнення цієї мети потрібно здійснити огляд та аналіз наявних технологій та підходів до розробки веб-додатків, а також проєктування та реалізація онлайн сервісу для організації дозвілля та спілкування використовуючи сучасні технології та методи розробки.

Об’єктом дослідження є технології для розробки онлайн сервісів.

Предметом дослідження є розробка онлайн сервісу для організації дозвілля та спілкування.

Особистий внесок студента

Основним результатом є:

1. оптимізація алгоритму кластеризації використаного в Google Maps API для використання в умовах онлайн сервісу;

2. проєктування та розробка онлайн сервісу для організації дозвілля та спілкування;

Апробація результатів випускної кваліфікаційної бакалаврської роботи та публікації

За результатами досліджень та розробок, проведених у бакалаврській роботі, підготовлено доповідь для участі в 8-мій Східно-Європейський конференції «Математичні та програмні технології Internet of Everything», що відбулася в м. Києві 14 травня 2021 р. та опубліковано тези

РОЗДІЛ 1
АНАЛІЗ ТА ОГЛЯД ПРЕДМЕТНОЇ ГАЛУЗІ

1.1. Мова програмування Python

Мова програмування Python – це мова програмування високого рівня, розширена, розроблена та багатоплатформна. Python розроблений таким чином, щоб бути особливо читабельним, дана мова регулярно використовує англійські ключові слова, в яких, інші мови використовують розділові знаки, також вона має менше синтаксичних конструкцій, ніж інші мови [2].

Крім того, мова Python інтегрує дивовижне управління з надзвичайно чіткими та простими мовним синтаксисом або правилами програмування. Більше того, вона пропонує користувальницькі інтерфейси для багатьох системних дзвінків, вбудованих бібліотек та різноманітних віконних систем. Таким чином, можна сказати, що мова Python має можливість інтегруватися в себе C та C++ модулі.

З іншого боку, Python також може працювати як розширювана мова для програм, які потребують програмованого робочого інтерфейсу користувача. Python – це мова програмування високого рівня, яка пропонує багато можливостей для перенесення, оскільки вона працює на багатьох платформах серед яких Linux, macOS, Windows.

Особливості Python:

- інтерпретується Python під час виконання. Розробнику більше не потрібно перед запуском збирати додаток, так як в більшості інших мов;
- Python підтримує об'єктно-орієнтований стиль або підхід програмування, що інкапсулює код всередині;
- має велику стандартну бібліотеку – основна частина бібліотеки Python

може бути дуже портативною та сумісною на різних платформах в UNIX, Windows та macOS;

- інтерактивний режим – який дозволяє здійснювати інтерактивні випробування та налагодження фрагментів коду;
- портативність – Python може працювати на широкому типі апаратних структур і має однаковий інтерфейс на всіх платформах;
- розширюваність – розробник можете додавати модулі низького рівня до Python. Ці модулі дозволяють програмістам додавати та налаштовувати модулі, щоб бути більш ефективними;
- програмування графічного інтерфейсу – Python має пакетам графічного інтерфейсу, які можна використовувати для створення додатків з графічним інтерфейсом.

Розширюваність мови має велике значення, мова була задумана саме як розширювана. Це означає, що є можливість вдосконалення мови усіма бажаючими. Інтерпретатор написаний на C і вихідний код доступний для будь-яких маніпуляцій. У разі необхідності, можна вставити його в свою програму і використовувати як вбудовану оболонку. Або ж, написавши на C свої доповнення до Python і скомпілювавши програму, отримати «розширений» інтерпретатор з новими можливостями.

Наявність великого числа модулів, що підключаються до програми, які забезпечують різні додаткові можливості. Такі модулі пишуться на C і на самому Python і можуть бути розроблені усіма досить кваліфікованими програмістами.
Основним недоліком є порівняно невисока швидкість виконання Python-програми, що обумовлено її інтерпретованих. Однак, це з лишком окупається перевагами мови при написанні програм не критичних до швидкості виконання.

Відмінності мови програмування Python від інших мов:

- Python не вимагає опису змінних. Вони створюються в місці їх ініціалізації, тобто при першому присвоєнні змінної будь-якого значення. Значить, тип змінної визначається типом присвоюється значення.
- керування пам'яттю – цілком автоматичне, не потрібно хвилюватися щодо розподілу або звільнення пам'яті.

- типи зв'язані з об'єктами, а не зі змінними. Це означає, що змінній може бути призначене значення будь-якого типу, і що масив може містити об'єкти різних типів. Традиційні мови не надають такої можливості.
- операції зазвичай виконуються в більш високому рівні абстракції. Це частково результат того, як написана мова, і частково результат розширеної стандартної бібліотеки кодів, що поставляється разом з Python.
- тип змінної не є незмінним. Будь-яке присвоювання для неї значення буде коректним і це призводить до того, що типом змінної стає тип нового присвоюється значення.
- досить оригінальним є те, як в Python групуються оператори. В C++, Java використовуються фігурні дужки. У мові Python все набагато простіше: виділення блоку операторів здійснюється шляхом зсуву виділеної групи на одну табуляції вправо щодо заголовка конструкції.

1.2. Мова програмування TypeScript

TypeScript – це об’єктно-орієнтована мова з відкритим кодом, розроблена та підтримувана корпорацією Майкрософт, ліцензована за ліцензією Apache 2. TypeScript був розроблений під керівництвом Андерса Хейлсберга, який також керував створенням мови C #. Вперше TypeScript було випущено в жовтні 2012 року [3].

TypeScript розширює JavaScript, додаючи типи даних, класи та інші об’єктно-орієнтовані функції за допомогою перевірки типу. Це JavaScript з статичною типізацією, який компілюється до простого JavaScript.

JavaScript – це динамічна мова програмування без статичних типів. JavaScript надає примітивні типи, такі як рядок, число, об'єкт тощо, але не перевіряє призначені значення. Змінні JavaScript оголошуються за допомогою ключового слова var, і воно може вказувати на будь-яке значення. JavaScript не підтримує класи та інші об'єктно-орієнтовані функції. Отже, без статичних типів непросто

використовувати JavaScript для побудови складних додатків з великими командами, що працюють над тим самим кодом.

Система типів підвищує якість коду, читабельність та полегшує обслуговування та рефакторинг кодової бази. Що ще важливіше, помилки можуть бути виявлені під час компіляції, а не під час виконання.

Отже, причина використання TypeScript полягає в тому, що він виявляє помилки під час компіляції, так що ви можете виправити це перед запуском коду. Він підтримує об'єктно-орієнтовані функції програмування, такі як типи даних, класи, переліки тощо, що дозволяє використовувати JavaScript у масштабі.

TypeScript компілюється в простий JavaScript. Компілятор TypeScript також реалізований в TypeScript і може використовуватися з будь-якими браузерами або механізмами JavaScript, такими як Node.js. Для компіляції TypeScript потрібне сумісне середовище ECMAScript 3 або вище. Цю умову сьогодні виконують усі браузери та механізми JavaScript.

Особливості TypeScript:

- Крос-платформа: TypeScript працює на будь-якій платформі, на якій працює JavaScript. Компілятор TypeScript можна встановити в будь-якій операційній системі, такі як Windows, macOS та Linux;
- Об'єктно-орієнтована мова: TypeScript надає потужні функції, такі як класи, інтерфейси та модулі. Ви можете писати чистий об’єктно-орієнтований код як для клієнтської, так і для серверної розробки;
- Статична перевірка типу: TypeScript використовує статичний набір тексту. Це робиться за допомогою анотацій типу. Це допомагає перевірити тип під час компіляції. Таким чином, ви можете знайти помилки під час введення коду, не запускаючи сценарій кожного разу. Крім того, за допомогою механізму виведення типу, якщо змінна оголошена без типу, вона буде виведена на основі її значення;
- Необов’язкова статична типізація: статична типізація TypeScript не є обов’язковою;
- Маніпуляція DOM: Як і JavaScript, TypeScript можна використовувати для маніпулювання DOM.

Переваги TypeScript:

- TypeScript – це мова з відкритим кодом, яка постійно розробляється та підтримується компанією Microsoft;
- TypeScript працює на будь-якому браузері або механізмі JavaScript;
- TypeScript схожий на JavaScript і використовує однаковий синтаксис та семантику. Весь код TypeScript нарешті перетворюється на JavaScript. Це дозволяє пришвидшити навчання для розробників, які в даний час використовують JavaScript;
- Код TypeScript можна викликати з існуючого коду JavaScript. TypeScript також без проблем працює з існуючими фреймворками та бібліотеками JavaScript;
- Файл визначення TypeScript із розширенням .ts забезпечує підтримку існуючих бібліотек JavaScript;
- TypeScript підтримує найновіші функції JavaScript від ECMAScript 2015. Він включає функції ES6 та ES7, які можуть працювати в механізмах JavaScript рівня ES5, таких як Node.js. Це пропонує величезну перевагу використання функцій майбутніх версій JavaScript у поточних механізмах JavaScript.

1.3. Мова структурованих запитів SQL

SQL (англ. structured query language – мова структурованих запитів) – декларативна мова програмування для взаємодії користувача з базами даних, що застосовується для формування запитів, оновлення і керування реляційними БД, створення схеми бази даних та її модифікації, системи контролю за доступом до бази даних. Сама по собі SQL не є ані системою керування базами даних, ані окремим програмним продуктом.

На початку 1970-х років в одній з дослідницьких лабораторій компанії IBM була розроблена експериментальна реляційна СУБД System R, для якої потім була створена спеціальна мова SEQUEL, що дозволяла відносно просто керувати даними в цій СУБД. Абревіатура SEQUEL розшифровувалася як англ. Structured English Query Language – «структурована англійська мова запитів». Пізніше з юридичних міркувань мова SEQUEL була перейменована в SQL.

Метою розробки було створення простої непроцедурної мови, якою зміг би скористатися будь-який користувач, що навіть не має навичок програмування. Власне розробкою мови запитів займалися Дональд Чемберлін та Рей Бойс. Пет Селінджер (Pat Selinger) займалася розробкою вартісного оптимізатора, Реймонд Лорі займався компілятором запитів.

Перший офіційний стандарт мови SQL був прийнятий ANSI в 1986 і ISO (Міжнародною організацією зі стандартизації) в 1987 (так званий SQL-86), який був уточнений в 1989 році. Наступний розвиток мови постачальниками СУБД зажадав ухвалення в 1992 р. нового розширеного стандарту (ANSI SQL-92, або просто SQL2), в якому були визначені спеціальні темпоральні розширення в стандарті SQL. Наступним стандартом став SQL:1999 (SQL3). В наш час діє стандарт, який був прийнятий у 2003 році (SQL:2003), а надалі зазнав незначними модифікацій.

SQL складається з декількох частин:

- Data Definition Language (DDL) – робота зі структурою бази;
- Data Manipulation Language (DML) – робота з рядками;
- Data Control Language (DCL) – робота з правами;
- Transaction Control Language (TCL) – робота з транзакціями.

Data Definition Language – це сімейство мов, що використовуються в комп'ютерних програмах або користувачами баз даних для опису структури даних.

Data Control Language – комп'ютерна мова, також частина SQL, що використовуються в комп'ютерних програмах або користувачами баз даних для контролю доступу до даних в базах даних.

Приклади команди мови Data Control Language:

- GRANT – дозволити визначеним користувачам виконувати визначені маніпуляції;
- REVOKE – скасувати надані права.

В Oracle виконання DCL-команди створює примусовий неявний commit, тобто завершення транзакції. В PostgreSQL виконання DCL є частиною транзакції, що триває.

Transaction Control Language – мова управління транзакціями, що використовується для обробки транзакцій.

Приклади команд мови Transaction Control Language:

- BEGIN TRANSACTION – почати транзакцію;
- COMMIT – прийняти зміни в транзакції;
- ROLLBACK – відміна змін.

Транзакцію можна розглядати як перетворення одного логічно узгодженого стану БД в інше, причому в проміжних точках (тобто під час виконання транзакції) БД може перебувати в неузгоджену стані.

Наприклад, при переказі грошей на оплату покупки з рахунку продавця на рахунок покупця (будемо вважати, що вся інформація зберігається в одній БД) треба з одного рахунку списати зазначену суму, а на іншій – зарахувати. Якщо сума списана, але не зарахована (наприклад, через що стався в момент між цими операціями збою живлення сервера), то виникає внутрішнє протиріччя – гроші втрачаються. Тому така послідовність операцій повинна бути оформлена у вигляді транзакції.

Для вказівки рамок транзакції прикладна програма повинна дати команди на початок транзакції, збереження транзакції, відкат транзакції. Транзакція починається з оператора BEGIN. Закінчується транзакція виконанням оператора COMMIT або оператора ROLLBACK. Оператор COMMIT встановлює точку фіксації, яка позначає закінчення логічної одиниці роботи. Виконання оператора ROLLBACK повертає БД до попередньої точки фіксації.

Якщо в процесі виконання транзакції до виконання оператора COMMIT відбудеться збій електроживлення, після процедури відновлення БД буде повернута в стан, що передує початку транзакції.

Можливо, знадобиться зробити відкат транзакції і в інших випадках. Для цього, зокрема, може використовуватися умовний оператор IF, який визначає умови успішного завершення транзакції:

Транзакції володіють чотирма наступними властивостями (їх ще називають властивостями ACID – Atomicity, Consistency, Isolation, Durability):

- атомарність – транзакція є неподільною, виконуються або всі дії, або нічого;
- узгодженість – транзакція переводить БД з одного узгодженого стану узгоджену в інший, без дотримання обов'язкової підтримай узгодженості в проміжних точках;
- ізоляція – навіть якщо запущено кілька конкуруючих між собою транзакцій, будь-яке оновлення, виконане однією з транзакцій, буде приховано від інших, поки що зробила зміни транзакція не буде зафіксована;
- довговічність – коли транзакція виконана, її поновлення зберігаються, навіть якщо в наступний момент станеться збій системи.

Коли одночасно відбувається виконання декількох транзакцій, що використовують одні ті ж об'єкти БД, ці транзакції називаються паралельними або конкуруючими. Якщо нс зробити додаткових дій, спрямованих на забезпечення перерахованих вище властивостей ACID, поява конкуруючих транзакцій може призвести до таких проблем, як:

- проблема втраченого поновлення – кілька користувачів змінюють один і той самий рядок, грунтуючись на її початковому значенні, в результаті частина даних буде втрачено, так як кожна наступна транзакція перезапише зміни, зроблені попередньої;
- проблема «брудного читання» може виникнути при читанні транзакцією записів, які були змінені, але ще не збережена в БД (дані змінені ще не завершилася транзакцією, яка після буде скасована);
- проблема неповторюваного читання – при повторному читанні даних, вже прочитаних раніше, транзакція виявляє модифікації або видалення, викликані іншою завершеною транзакцією; подібна зміна може порушити логіку роботи транзакції;
- проблема читання «фантомів» з'являється, коли при повторному читанні даних транзакція виявляє нові рядки, додані або змінені інший транзакцією, завершеною після попереднього читання цього набору даних.

Data Manipulation Language – використовується для отримання та маніпулювання даними в реляційній базі даних.

Мови DML спочатку використовувалися лише комп'ютерними програмами, але з появою SQL вони стали доступними і для людей. DML мають свою функціональну здатність, організовану за початковим словом в запиті, яке майже завжди є дієсловом. У випадку з SQL ці дієслова – «SELECT», «INSERT», «UPDATE», «DELETE».

Мови DML можуть істотно розрізнятися у різних виробників СУБД. Існує стандарт SQL, встановлений ANSI, але виробники СУБД часто пропонують свої власні «розширення» до мови SQL.

Запит – найчастіше виконувана операція в SQL. Для здійснення запитів використовується декларативна інструкція SELECT. SELECT отримує дані з однієї чи більше таблиць, або виразів. Стандартна інструкція SELECT не здійснює постійних змін бази даних. Запити дозволяють користувачу отримати бажані дані, залишаючи базі даних створення плану запиту, його оптимізацію, і виконання фізичних операцій необхідних для отримання бажаних результатів.

До запиту входить список колонок, які повинні включатись в кінцевий результат, зазвичай одразу після ключового слова SELECT. Можна використати символ "*", щоб описати, що запит повинен повертати всі колонки всіх таблиць, з якими працює. SELECT – найскладніша інструкція SQL, з необов'язковими ключовими словами і пунктами, до яких входить пункт FROM, який задає таблиці, з яких треба отримати дані.

Переваги SQL:

- незалежність від конкретної СУБД. Не зважаючи на наявність діалектів і відмінностей в синтаксисі, більшість текстів SQL-запитів, що містять DDL і DML, можуть бути досить легко перенесені з однієї СУБД в іншу. Існують системи, розробники яких спочатку орієнтувалися на застосування щонайменше кількох СУБД (наприклад: система електронного документообігу Documentum може працювати як з Oracle, так і з Microsoft SQL Server та IBM DB2). Зрозуміло, що при застосуванні деяких специфічних для реалізації можливостей, такого рівня перенесення дуже важко досягти;

- наявність стандартів. Наявність стандартів і наборів тестів для виявлення сумісності та відповідності конкретній реалізації SQL загальноприйнятому стандарту тільки сприяє «стабілізації» мови;
- декларативність. За допомогою SQL програміст описує лише дані, які потрібно витягнути або модифікувати. Те, яким чином це зробити, вирішує СУБД безпосередньо при обробці SQL-запиту. Не слід вважати, що це повністю універсальний принцип – програміст описує набір даних для вибірки або модифікації, проте йому при цьому корисно уявляти, як СУБД інтерпретуватиме текст його запиту. Такі моменти стають особливо критичними при роботі з великими базами даних та зі складними запитами – чим складніше сконструйований запит, тим більше варіантів написання (різних за швидкістю виконання, але тих самих за набором даних) він припускає.

Недоліки SQL:

- явна вказівка порядку стовпчиків зліва направо;
- стовпчики без імені та імена стовпчиків, що дублюються;
- значна надлишковість;
- мову SQL було початково заплановано як засіб роботи кінцевого користувача, врешті-решт вона стала настільки складною, що перетворилася на інструмент програміста;

1.4. Контейнерна технологія Docker

Docker – це інструмент, призначений для спрощення створення, розгортання та запуску програм за допомогою контейнерів. Контейнери дозволяють розробнику пакувати додаток з усіма необхідними йому частинами, такими як бібліотеки та інші залежності, і розгортати його як один пакет. Тим самим, завдяки контейнеру, розробник може бути впевнений, що програма буде працювати на будь-якій іншій машині Linux незалежно від будь-яких налаштованих параметрів, які можуть мати машини, які можуть відрізнятися від машини, що використовується для написання та тестування коду [4].

Однією з цілей сучасної розробки програмного забезпечення є тримати додатки на одному хості або кластері ізольованими один від одного, щоб вони не надто перешкоджали роботі або обслуговуванню один одного. Це може бути складно, завдяки пакетам, бібліотекам та іншим програмним компонентам, необхідним для їх роботи. Одним з варіантів вирішення цієї проблеми були віртуальні машини, які підтримують додатки на одному і тому ж апаратному забезпеченні повністю відокремленими і зводять до мінімуму конфлікти між компонентами програмного забезпечення та конкуренцію за апаратні ресурси. Але віртуальні машини є громіздкими – кожна вимагає власної ОС, тому зазвичай розміром гігабайт є складна в обслуговуванні та модернізації.

Контейнери – ізолюють середовища виконання програм одне від одного, але поділяють базове ядро ОС. Зазвичай вони вимірюються в мегабайтах, використовують набагато менше ресурсів, ніж VM, і запускаються практично негайно. Контейнери забезпечують високоефективний механізм для комбінування програмних компонентів, необхідних на сучасному підприємстві, та для постійного оновлення та обслуговування цих компонентів програмного забезпечення.

Образ контейнера Docker – це легкий, окремий, виконуваний пакет програмного забезпечення, який включає все необхідне для запуску програми: код, час виконання, системні інструменти, системні бібліотеки та налаштування.

Контейнер Docker працює на будь-якій машині, яка підтримує середовище виконання контейнера. Програми не повинні прив'язуватися до хост-операційної системи, тому і середовище програми, і базове операційне середовище можуть бути чистими та мінімальними.

Наприклад, контейнер MySQL для Linux працює на більшості будь-якої системи Linux, яка підтримує контейнери. Усі залежності для програми зазвичай постачаються в одному контейнері.

Додатки на основі контейнерів можна легко переміщувати з надомних систем у хмарні середовища або з ноутбуків розробників на сервери, якщо цільова система підтримує Docker і будь-який із сторонніх інструментів, які можуть використовуватися з ним, наприклад Kubernetes.

Зазвичай образ контейнерів Docker повинні бути створені для певної платформи. Наприклад, контейнер Windows не працюватиме в Linux і навпаки. Раніше одним із способів цього обмеження було запуск віртуальної машини, яка запускала екземпляр необхідної операційної системи та запуск контейнера у віртуальній машині.

Однак команда Docker з тих пір розробила більш елегантне рішення, що називається маніфестами, які дозволяють зображенням для декількох операційних систем упаковуватись в одне зображення. Маніфести все ще вважаються експериментальними, але вони натякають на те, як контейнери можуть стати кросплатформним прикладним рішенням, а також крос-середовищем.

Певним чином Docker трохи схожий на віртуальну машину. Але на відміну від віртуальної машини, а не для створення цілої віртуальної операційної системи, Docker дозволяє програмам використовувати те саме ядро Linux, що і система, в якій вони запущені, і вимагає лише доставки додатків із речами, які вже не працюють на хост-комп'ютері. Це дає значне підвищення продуктивності та зменшує розмір програми. Схематичне порівняння контейнеризації та віртуалізації зображено на рис. 1.1.

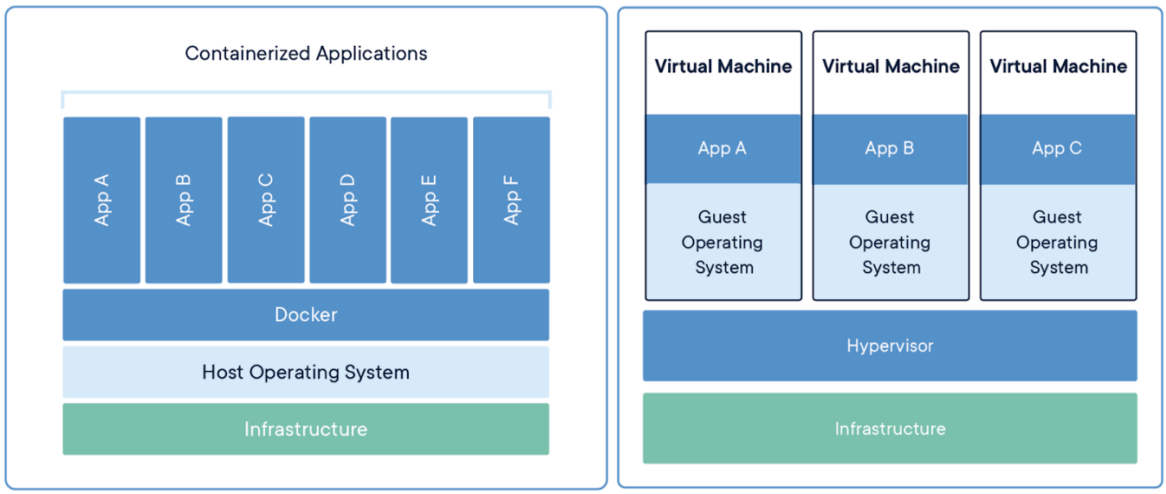

Рис. 1.1. Схематичне порівняння контейнеризації та віртуалізації

Контейнери – це абстракція на рівні програми, яка пакує код і залежності. Кілька контейнерів можуть працювати на одній машині та обмінюватися ядром ОС з іншими контейнерами, кожен працює як окремі процеси в просторі користувача.

Контейнери займають менше місця, ніж VM (зображення контейнерів, як правило, розміром десятки МБ), можуть обробляти більше додатків і вимагати меншої кількості віртуальних машин та операційних систем.

Віртуальні машини – це абстракція фізичного обладнання, що перетворює один сервер на багато серверів. Гіпервізор дозволяє на одній машині працювати декілька віртуальних машин. Кожна віртуальна машина включає в себе повну копію операційної системи, програми, необхідні бінарні файли та бібліотеки, всі ці залежності і дані займають десятки гігабайт.

1.5. Протоколи передачі даних IP, TCP та HTTP

IP (Інтернет протокол) – це головний протокол комунікації в наборі протоколів для передачі даних через мережу Інтернет, головною функцією якого я передача даних між двома комп’ютерами в мережі, даний протокол дає гарантію, що данні будуть доставлені адресату.

IP має завдання доставити пакети від відправника до цільового вузла виключно на основі IP-адрес у заголовках пакетів. Для цього IP визначає структуру пакетів, які інкапсулюють дані, що підлягають доставці [5].

Потік даних від відправника до місця призначення розбивається на малі пакети. Кожного разу, коли вузли в мережі надсилають або отримують пакет даних, також з цією інформацією іде велика кількість важливою інформації, сюди входить інформація, додана протоколом управління передачею даних (TCP).

TCP (протокол керування передачею) – відповідає за те, щоб усі дані, що надсилаються в потоці пакетів, рухалися з точки A в точку B у правильному порядку та неушкодженими. TCP повідомляє цільовому комп'ютеру, яка програма повинна отримувати дані. TCP, жертвує швидкістю, щоб забезпечити надійність передавання даних. Деякі форми передачі даних, такі як потокове відео, де ідеальна точність не має великого значення, а швидкість передачі має, краще використовувати інші протоколи такі як UPD, що оптимізують швидкість над точністю.

TCP використовує методику, відому як підтвердження з повторною передачею, вимагаючи прийому сигналу по успішну передачу пакету, щоб знати, які дані були отримані. Завдяки цьому відправник знає, які пакети надсилати далі, або, можливо, повторно, для підтримки бездоганного потоку даних. Отже, надіслані байти можуть точно відповідати отриманим байтам. Жодні дані не змінюються і не втрачаються під час подорожі через всесвітню павутину.

Хоча TCP містить інформацію про те, які дані отримані чи ще не отримані, HTTP містить конкретні інструкції щодо того, як читати та обробляти ці дані, як тільки вони надходять. Перед тим, як дані будуть надіслані з одного вузла в Інтернеті на інший, вони отримують інформацію, що деталізує походження запиту. Це робиться за допомогою HTTP або протоколу передачі гіпертексту.

Вводячи URL-адресу у свій веб-браузер, ми надсилаємо HTTP-запит на веб-сервер. Потім сервер дає відповідь, знову використовуючи протокол HTTP.

Два найпоширеніші приклади запитів HTTP: POST, що позначає, що вони містять дані, які потрібно зберегти на сервер та GET, з проханням отримати дані із сервера.

HTTP розташований сьомому рівні моделі взаємозв’язку відкритих систем TCP знаходиться на четвертому рівні.

Комп'ютер фіксує фізичну електрику, яка в певному сенсі «загортає» пакет невідчутних даних для транспорту, без TCP комп'ютер не знав би, до якої програми вказувати пакет. TCP вказає комп'ютеру куди направляти пакет.

Опинившись у такій програмі, як Firefox чи Google Chrome, зчитаються інструкції HTTP. Браузер дізнається природу вхідних даних і може нарешті правильно завантажити вміст веб-сторінки.

1.6. Клієнт-серверна архітектура

Архітектура клієнт-сервер є одним із архітектурних шаблонів програмного забезпечення та є домінуючою концепцією у створенні розподілених мережних

застосунків і передбачає взаємодію та обмін даними між ними. Вона передбачає такі основні компоненти:

- набір серверів, що приймають HTTP-запити від клієнтів, зазвичай веб-браузерів та видають їм HTTP-відповіді.
- набір клієнтів, які використовують сервіси, що надаються серверами;
- мережа, яка забезпечує взаємодію між клієнтами та серверами.

Сервери є незалежними один від одного. Клієнти також функціонують паралельно і незалежно один від одного. Немає жорсткої прив'язки клієнтів до серверів. Більш ніж типовою є ситуація, коли один сервер одночасно обробляє запити від різних клієнтів; з іншого боку, клієнт може звертатися то до одного сервера, то до іншого. Клієнти мають знати про доступні сервери, але можуть не мати жодного уявлення про існування інших клієнтів. Схематичне зображення клієнт-серверної архітектури зображено на рис. 1.2.

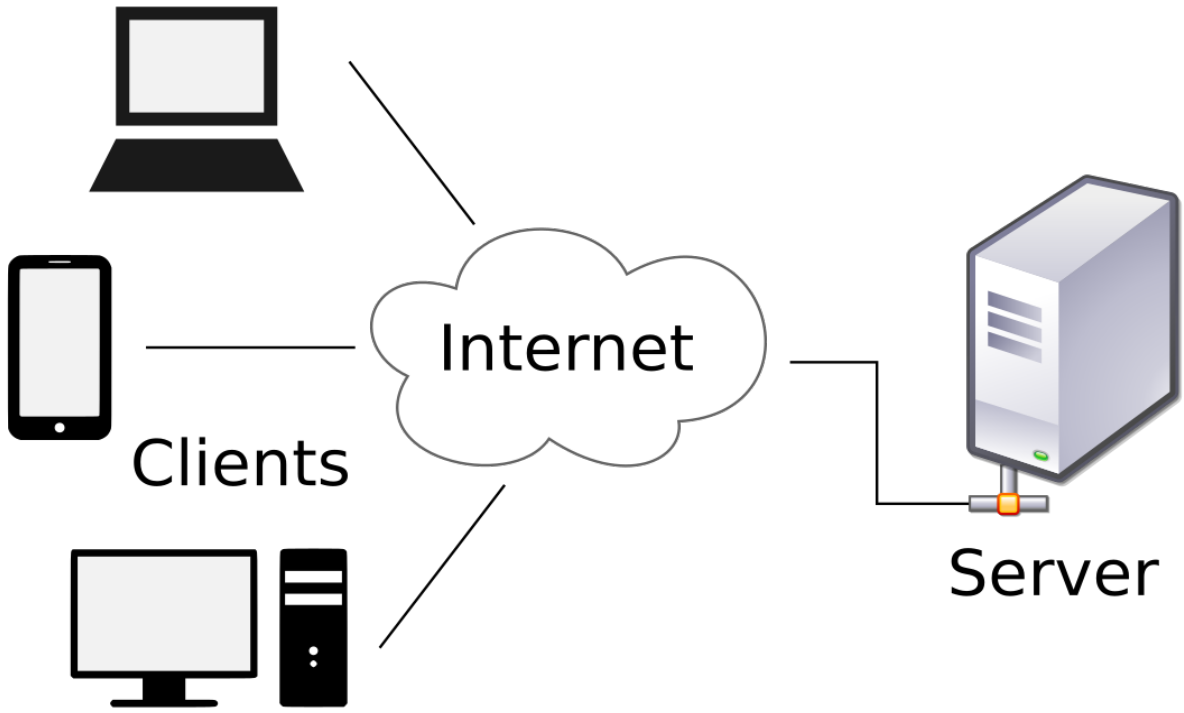

Рис. 1.2. Схематичне зображення архітектури «Клієнт-сервера»

Основна ідея архітектури «клієнт-сервер» полягає в поділі мережевого додатки на кілька компонентів, кожен з яких реалізує специфічний набір сервісів. Компоненти такого додатку можуть виконуватися на різних комп'ютерах, виконуючи серверні і/або клієнтські функції. Це дозволяє підвищити надійність, безпеку і продуктивність мережевих додатків і мережі в цілому.

Модель клієнт-серверної взаємодії визначається перш за все розподілом обов'язків між клієнтом та сервером. Можна відокремити три рівні операцій:

- рівень представлення даних, який по суті являє собою інтерфейс користувача і відповідає за представлення даних користувачеві і введення від нього керуючих команд;
- прикладний рівень, який реалізує основну логіку застосунку і на якому здійснюється необхідна обробка інформації;
- рівень управління даними, який забезпечує зберігання даних та доступ до них.

1.7. REST API

REST – скорочення від Representational State Transfer, що можна перекласти як «передача репрезентативного стану». Це стиль проектування розподілених систем. Центральною абстракцією в REST є ресурс [6].

Головні принципи REST API:

- клієнт-сервер – відокремлюючи проблеми користувальницького інтерфейсу від проблем зберігання даних, ми покращуємо портативність користувальницького інтерфейсу на кількох платформах та покращуємо масштабованість шляхом спрощення компонентів сервера;
- stateless – кожен запит від клієнта до сервера повинен містити всю інформацію, необхідну для розуміння запиту, стан сесії зберігає клієнт;
- кешованість – обмеження кешу вимагають, щоб дані у відповіді на запит неявно або явно позначали як кешовані або не кешовані. Якщо відповідь є

кешованою, то кеш клієнта надається право повторно використовувати ці дані відповіді для наступних, рівнозначних запитів;

- уніфікований інтерфейс – за допомогою застосування принципу загальної інженерії програмного забезпечення до компонентного інтерфейсу спрощується загальна архітектура системи та покращується видимість взаємодій. Щоб отримати єдиний інтерфейс, для керування поведінкою компонентів потрібно кілька архітектурних обмежень. REST визначається чотирма обмеженнями інтерфейсу: ідентифікація ресурсів, маніпулювання ресурсами через представництва, повідомлення з самоописом та гіпермедіа як двигун стану застосування;
- багаторівнева система – рівневий стиль системи дозволяє архітектурі складатися з ієрархічних рівнів, обмежуючи поведінку компонентів таким чином, щоб кожен компонент не міг «бачити» поза безпосереднім рівнем, з яким вони взаємодіють.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ДОДАТКУ

2.1. Проєктування архітектури додатку

В якості архітектури для додатку було обрано клієнт-серверну архітектуру. Крім стандартного набору компонентів які присутні в клієнт-серверній архітектурі (клієнт, сервер, база даних) також присутній такий елемент як Worker (робітник).

Архітектура додатку складається з чотирьох основних компонентів зображених на рис. 2.1.

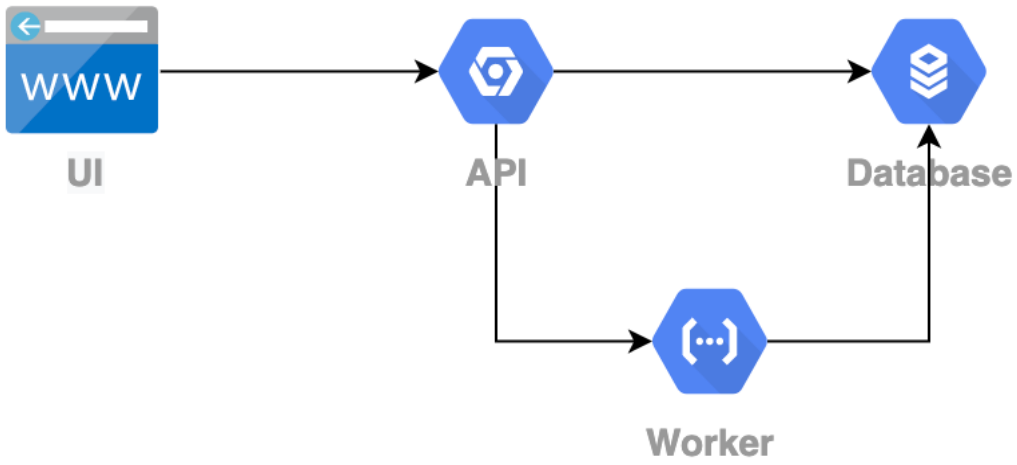

Рис. 2.1. Схематичне зображення архітектури додатку

В якості сервера виступає Python-додаток з використанням фреймворку Flask, який надає клієнту API, використовуючи який клієнт може отримати всі необхідні дані, а також виконувати всі необхідні операції.

Основні функції сервера:

- зберігання, доступ, захист даних;
- обробка клієнтського запиту;
- відправлення відповіді клієнту.

В якості клієнта виступає TypeScript-додаток з використанням бібліотеки React, який собою представляє зручний інтерфейс для взаємодії користувача з

додатком. Для взаємодії з сервером використовується протокол HTTPS, який гарантує цілісний та безпечний обмін даними.

Основні функції клієнта:

- надання користувальницького інтерфейсу;
- формулювання запиту до сервера і його відправка;
- отримання результатів запиту і відправка додаткових команд (запитів на додавання, оновлення або видалення даних).

В якості сховища даних використовується реляційна база даних PostgreSQL. Для обміну даними з сервером та робітником (Worker) використовується протокол TCP/IP, який гарантує цілісність даних, а також не містить рівня який займається безпекою передачі даних. В даному випадку нам не потрібен даний рівень, так як доступ до бази даних будуть здійснювати тільки API та Worker, а сама база даних буде розміщена за Firewall.

Основні фікції бази даних:

- збереження та обробка даних;
- додавання даних;
- редагування даних;
- упорядкування та перегляд;
- надання спільного доступу до даних декільком серверам (API, Worker).

В якості робітника (Worker) виступає Python-скрипт, який автоматично запускається та виконується кожні 5 хвилин, даний скрипт, виконує дві функції:

- отримує з бази даних всі події які мають статус «Planned» та час початку події менше поточного часу, це означає що подія уже розпочалась, тому робітник змінює статус даних подій на «Started»;
- отримує з бази даних всі події які мають статус «Started» та час закінчення події менше поточного часу, це означає що подія уже закінчилась, тому робітник змінює статус даних подій на «Finished».

Всі компоненти додатку мають свій власний Docker Image, що дозволяє розгорнути додаток використовуючи будь-якого постачальника хмарних послуг.

2.2. Проєктування бази даних

Для збереження та подальшої оброки інформації про користувачів, подій та інших даних було використано реляційну базу даних під управлінням PostgreSQL.

Загалом база даних містить тринадцять таблиць. Повна схема бази даних додатку зображена на рис. 2.2.

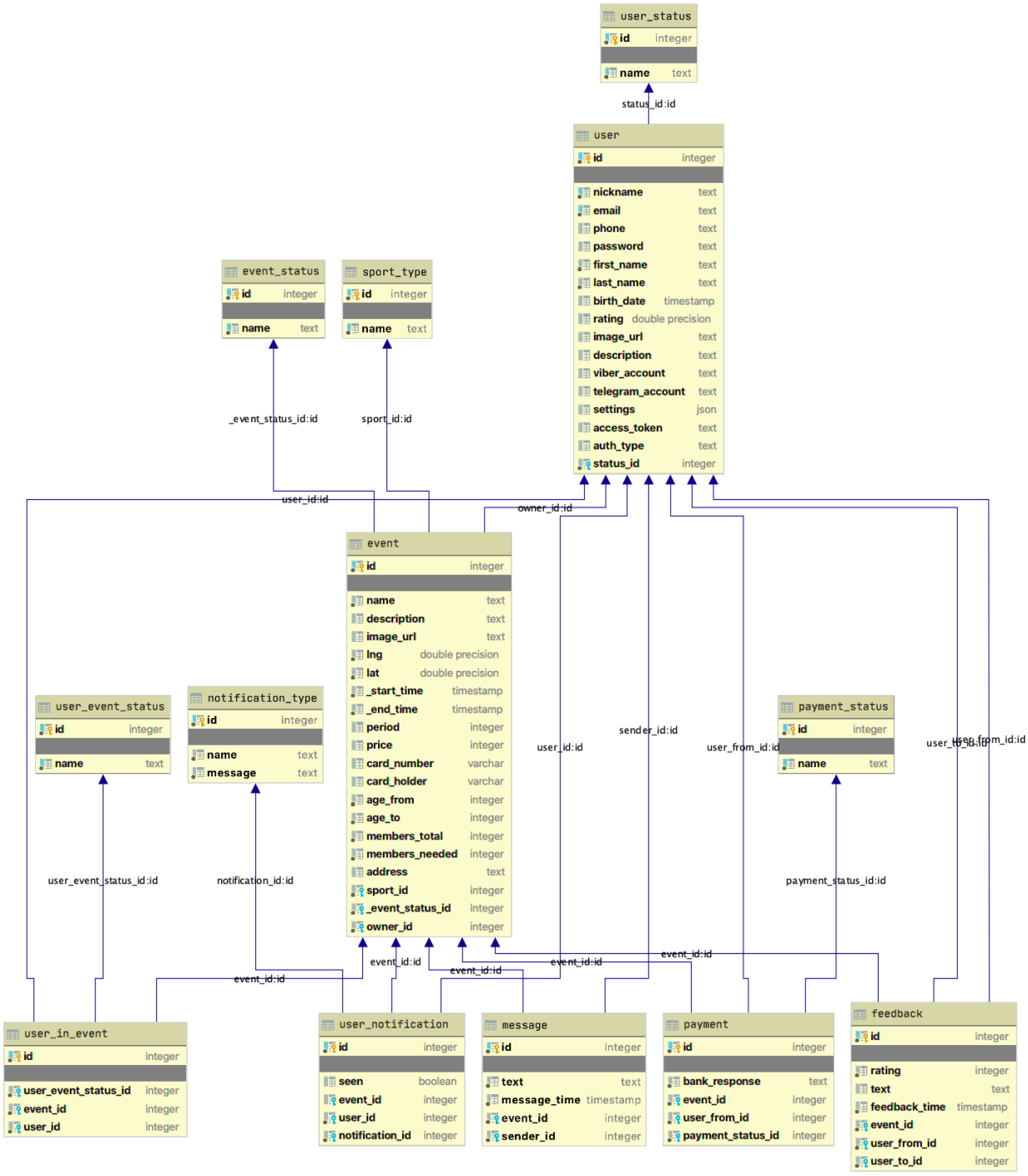

Рис. 2.2. Схема бази даних додатку

В додатку існує дві головні сутності це користувачі, а також події. Спочатку було створено таблицю для збереження даних про користувача.

Таблиця «User» має наступні атрибути:

- id – первинний ключ (тип даних integer);
- nickname – ім’я користувача яке видно для всіх користувачів (тип даних text);
- email – електронна пошта, необхідна під час реєстрації а також, я отримання повідомлень про статус події (тип даних text);
- phone – мобільний номер телефону, необхідний для отримання повідомлень про статус події (тип даних text);
- password – хеш-пароль, необхідний для входу в систему (тип даних text);
- first_name – ім’я користувача події (тип даних text);
- last_name – прізвище користувача (тип даних text);
- birth_date – дата народження користувача (тип даних timestamp);
- rating – рейтинг користувача, який рахується на базі отриманих оцінок в подіях (тип даних double);
- image_url – посилання на зображення яке використовується я зображення профілю користувача (тип даних text);
- settings – налаштування (тип даних json);
- status_id – статус профілю користувача в додатку (зовнішній ключ на таблицю «User Status»).

Атрибут «settings» використовується для збереження налаштувань користувача, які включать в себе, типи повідомлення та спосіб отримання повідомлень. Атрибут «nickname» використовується для відображення в системі, якщо користувач не хоче що його справжні ім’я та прізвище були не помітні для інших користувачів.

Для зберігання даних про подію використовується таблиця «Event».

Таблиця «Event» має наступні атрибути:

- id – первинний ключ (тип даних integer);

- name – назва події (тип даних text);
- description – короткий опис події (тип даних text);
- image_url – посилання на зображення яке використовується я зображення події (тип даних text);
- lng – довгота, для коректного розміщення маркера події на карті (тип даних double);
- lat – широта, для коректного розміщення маркера події на карті (тип даних double);
- start_time – дата та час початку події (тип даних timestamp);
- end_time – дата та час закінчення події (тип даних timestamp);
- price – ціна участі в події (тип даних integer);
- age_from – вік з якого дозволено брати участь в події (тип даних integer);
- age_to – вік до якого дозволено брати участь в події (тип даних integer);
- members_total – максимально необхідна кількість учасників (тип даних integer);
- members_needed – кількість необхідних учасників (тип даних integer);
- sport_id – ідентифікатор типу спорту (зовнішній ключ на таблицю «Sport Type», тип даних integer);
- event_status_id – ідентифікатор статусу події (зовнішній ключ на таблицю «Event Status», тип даних integer);
- owner_id – ідентифікатор організатора події (зовнішній ключ на таблицю «User», тип даних integer).

Атрибути «age_from» та «age_to» використовуються для обмежень доступу до події по віку, але це не є обов’язковим, організатор події має можливість обирати самостійно. Також присутні атрибут «price» він також є необов’язковим, і не використовується, цей атрибут потрібний більш зручного розширення в майбутньому.

Одна подія може мати багато учасників, а один користувач може приймати участь у багатьох подіях, така реляція між таблицями називається «багато до багатьох». В такому випадку слід використовувати допоміжну таблицю.

Було створено таблицю «User_in_event», яка має такі атрибути:

- id – первинний ключ (тип даних integer);
- event_id – ідентифікатор події (зовнішній ключ на таблицю «Event», тип даних integer);
- user_id – ідентифікатор учасника (зовнішній ключ на таблицю «User», тип даних integer);
- user_event_status_id – ідентифікатор події (зовнішній ключ на таблицю «User_event_status», тип даних integer).

Так як користувач додатку подає заявку на участь в події, а потім організатор події має можливість відбирати учасників, тому потенційний учасник події має свій статус в події, для збереження цього статусу використовується атрибути «user_event_status_id».

Після зміни свого статусу в події користувач отримує відповідне оповіщення, щоб користувач міг можливість отримувати та переглядати оповіщення було створено таблицю «User_notification».

Таблиця «User_notification» має наступні атрибути:

- id – первинний ключ (тип даних integer);
- seen – булева змінна, яка має значення «True» якщо користувач уже переглянув це оповіщення, «False» – користувач ще не переглянув дане оповіщення (тип даних boolean);
- user_id – ідентифікатор користувача якому адресоване оповіщення (зовнішній ключ на таблицю «User», тип даних integer);
- event_id – ідентифікатор події, до якої відноситься дане оповіщення (зовнішній ключ на таблицю «Event», тип даних integer);
- notification_id – ідентифікатор типу оповіщення (зовнішній ключ на таблицю «Notification_type», тип даних integer).

Користувачі додатку мають можливість спілкуватись між собою в чаті, тому для збереження повідомлень було створено таблицю, вона буде називатись «Message». Таблиця буде зберігати саме повідомлення, а також всі необхідні метадані.

Таблиця «Message» має наступні атрибути:

- id – первинний ключ (тип даних integer);
- text – тіло повідомлення (тип даних text);
- message_time – дата та час відправлення повідомлення (тип даних timestamp);
- event_id – ідентифікатор події (зовнішній ключ на таблицю «Event», тип даних integer);
- sender_id – ідентифікатор відправника (зовнішній ключ на таблицю «User», тип даних integer).

Оскільки повідомленнями можуть обмінюватись тільки учасники однієї події, тому слід зберігати ідентифікатор події в метаданих для повідомлення, щоб користувачі інших подій не мали доступу до повідомлень, для цього використовується атрибут «event_id».

Схема бази даних має додаткові таблиці для збереження метаданих. Серед таких таблиць можна знайти наступні:

- «User_status» – місить всі можливі статуси профілю в додатку («activated», «deactivated», «removed», «created»);
- «Event_status» – містить всі можливі дані статуси події («created», «pending», «finished», «canceled»);
- «Sport_type» –містить всі можливі типи спорту, на основі яких можуть бути створені події;
- «User_event_status» – містить всі можливі статуси користувача в відповідній події («waiting_approval», «accepted», «declined»);
- «Notification_type» – містить всі можливі типи оповіщень, які отримує користувач, в залежності від зроблених дій в додатку.

Всі ці таблиці мають одинакові атрибути: «id» (первинний ключ, тип даних integer), а також «name» (зберігає необхідні метадані). Всі дані в цих таблицях використовується як зовнішні посилання з інших таблиць, користувач додатку не має доступу до цих таблиць. Дані в ці таблиці добавляються автоматично при розгортанні системи, при потребі ці дані можуть бути змінені або розширені.

Після завершення події всі учасники мають можливість залишити відгук по інших учасників та виставити рейтингові бали, бля збереження цих даних було створено таблицю «Feedback». Таблиця «Feedback» має такі атрибути:

- id – первинний ключ (тип даних integer);
- rating – рейтинг отриманий користувачем (тип даних integer);
- text –текст відгуку (тип даних text);
- feedback_time – дата та час створення відгуку (тип даних timestamp);
- event_id – ідентифікатор події в якій брали участь користувачі (зовнішній ключ на таблицю «Event», тип даних integer);
- user_from_id – ідентифікатор користувача, який залишив відгук (зовнішній ключ на таблицю «User», тип даних integer);
- user_to_id – ідентифікатор користувача, про якого залишили відгук (зовнішній ключ на таблицю «User», тип даних integer).

Схема бази даних має ще декілька таблиць, які приховані від користувача, і потрібні для наступних версій додатку, в яких буде можливість для створення платних подій. Для даного функціоналу будуть використані наступні таблиці:

- «Payment_status» – буде використана для, збереження статусу оплати учасника;
- «Payment» – для збереження платежів, а також всіх необхідних метаданих, необхідних для підтвердження платежу.

2.3. Опис допоміжних бібліотек для розробки системи

Для розробки серверної частини, а саме серверу з API, за основу було обрано бібліотеку Flask.

Flask – це веб-фреймворк, який представляє колекцію бібліотек та модулів, що дозволяє розробнику веб-додатків розробляти не турбуючись про деталі низького рівня.

Flask часто називають мікрофреймворком. Він спрямований на те, щоб ядро програми було простим, але розширюваним. Flask не має вбудованого рівня абстракції для обробки баз даних, а також не має форми перевірки валідації. Натомість Flask підтримує розширення для додавання такої функціональності до програми.

Переваги Flask:

- Можливість розширювати базовий функціонал сторонніми бібліотеками;
- Використання WSGI. Web Server Gateway Interface (Інтерфейс шлюзу веб-сервера) – прийнятий як стандарт для розробки веб-додатків Python. WSGI – це специфікація універсального інтерфейсу між веб-сервером та веб-додатками;
- Використання Jinja2. Jinja2 – це популярний механізм шаблонування для Python. Система веб-шаблонів поєднує шаблон із певним джерелом даних для відображення динамічних веб-сторінок.

Для розробки клієнтської частини за основу було обрано бібліотеку React.

React – це бібліотека JavaScript для побудови компонованих користувальницьких інтерфейсів. Це заохочує створення багаторазових компонентів інтерфейсу, які представляють дані, які змінюються з часом. React абстрагує від розробників DOM, пропонуючи простішу модель програмування та кращу продуктивність. React також може відображатись на сервері за допомогою Node.

Особливості React:

- JSX – це розширення синтаксису JavaScript. Не потрібно використовувати JSX при розробці React, але це рекомендується;
- Односпрямований потік даних. React реалізує односторонній потік даних, що полегшує реалізацію додатків, а також підтримує високий рівень читабельності коду.

Переваги React:

- React використовує віртуальний DOM, який є об'єктом JavaScript. Це

покращує продуктивність додатків, оскільки віртуальний DOM JavaScript швидший за звичайний DOM;

- може використовуватися на стороні клієнта та сервера, а також з іншими фреймворками;
- компоненти та шаблони даних покращують читабельність, що допомагає підтримувати великі програми;
- можливість інтеграції з TypeScript, що дозволяє підтримувати великі додатки.

Для комунікації з базою даних було обрано ORM (Object Relational Mapper) SQLAlchemy.

SQLAlchemy – це повний набір інструментів для роботи з базами даних для мови програмування Python. Дана бібліотека має кілька різних областей функціональності, які можна використовувати як окремо, так і комбінувати разом. Її основні компоненти проілюстровані на рис. 2.3, архітектура бібліотеки складається з декількох шарів:

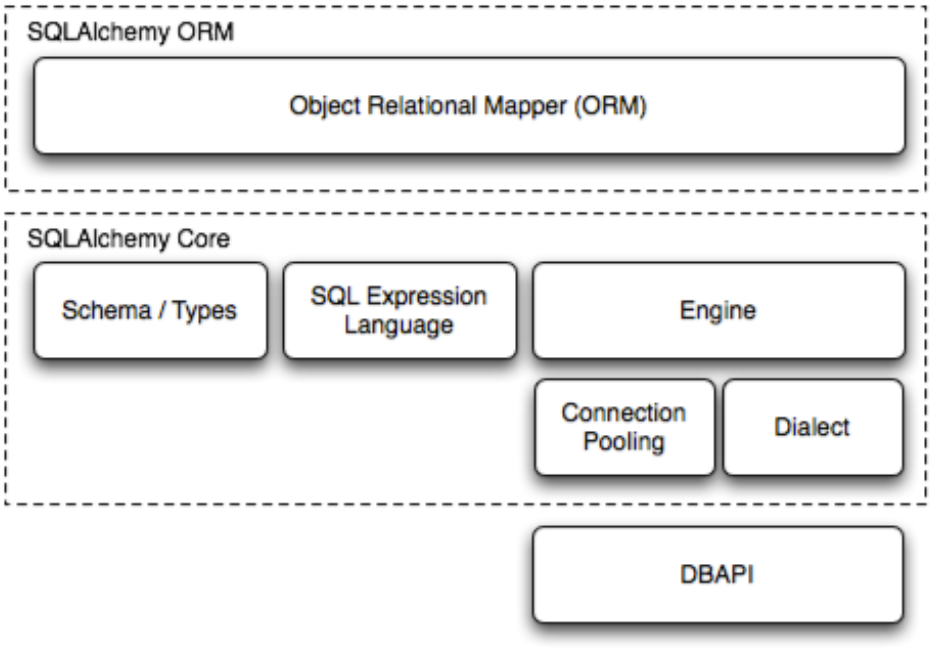

Рис 2.3. Архітектура бібліотеки SQLAlchemy

Двома найбільш значущими частинами SQLAlchemy є рівні ORM та Core.

Core містить служби для роботи з SQL і інтеграції та опис баз даних SQLAlchemy, найвизначніша частина яких – SQL Expression Language (мова виразів SQL).

Мова виразів SQL – це власний набір інструментів, незалежний від пакета ORM, який забезпечує систему побудови виразів SQL, представлених складаними об'єктами, які потім можна виконати в цільовій бази даних у межах конкретної транзакції, повертаючи набір результатів. Вставки, оновлення та видалення (тобто DML) досягаються шляхом передачі об'єктів виразів SQL, що представляють ці оператори, разом зі словниками, що представляють параметри, які будуть використовуватися з кожним оператором.

ORM базується на Core, щоб забезпечити засоби роботи з об'єктною моделлю домену, зіставленою зі схемою бази даних. При використанні ORM оператори SQL будуються здебільшого так само, як і при використанні Core, однак завдання DML, яке тут стосується збереження бізнес-об'єктів у базі даних, автоматизується за допомогою шаблону, званого одиницею роботи, який перекладає зміни стану щодо змінних об'єктів у конструкції INSERT, UPDATE та DELETE, які потім викликаються з точки зору цих об'єктів. Оператори SELECT також доповнюються специфічними для ORM автоматизацією та об'єктно-орієнтованими можливостями запитів.

Переваги використання SQLAlchemy:

- лаконічність – ORM дозволяє користувачам використовувати обрану мову програмування з більш стислим і простим синтаксисом, ніж використання необроблених запитів SQL, зменшуючи кількість необхідного коду;
- оптимальна – ORM також дозволяє користувачам скористатися перевагами оптимізації об'єктно-орієнтованого програмування, такими як успадкування, інкапсуляція та абстракція, представляючи записи в базі даних як об'єкти;
- гнучкість – за допомогою ORM користувачі можуть легко переключатися між різними системами управління базами даних, не маючи необхідного знання цих систем, а також мови SQL.

Недоліки використання SQLAlchemy:

- час – оскільки ORM сторонній набір інструментів то користувачам доведеться витратити час на навчання та ознайомлення з цими інструментами;
- менше контролю – використовуючи ORM, користувачі матимуть менше контролю бази даних.

РОЗДІЛ 3
ОСОБЛИВОСТІ РЕАЛІЗАЦІЇ СИСТЕМИ

3.1. Опис процесу обміну повідомленнями між клієнтом та сервером

Серверна частина додатку розроблена з використання архітектурного підходу REST API. Однією з ключових переваг розробки серверу з REST API є те, що сервер забезпечить високу гнучкість. Дані не прив'язані до ресурсів або методів, тому REST може обробляти кілька типів викликів, повертати різні формати даних і навіть структурно змінюватись за допомогою правильної реалізації. Ця гнучкість дозволяє створеному серверу задовольняти потреби різних типів клієнтів.

В архітектурі REST сервер не зберігає ніяких даних про стан клієнта. Сесії повинен зберігати клієнт. Це означає, якщо сервер отримав два різних запити від одного клієнта, вони не повинні впливати один на одного. Всю інформацію, потрібну серверу для здійснення дії, клієнт повинен відправляти відразу.

Для авторизації та ідентифікації використовується JSON Web Token (JWT). JWT – це відкритий стандарт, який визначає компактний та автономний спосіб безпечної передачі інформації між сторонами. Цю інформацію можна перевірити та довіряти їй, оскільки вона має цифровий підпис. JWT можна підписати, використовуючи секрет або пару відкритих-приватних ключів, використовуючи RSA або ECDSA. Підписаний токен можна перевірити на цілісність вимог, що містяться в ньому, тоді як зашифровані токени приховують ці вимоги від інших сторін. Коли токени підписуються за допомогою пар відкритого / приватного ключів, підпис також засвідчує, що лише сторона, яка тримає закритий ключ, та, що підписала його.

Після того як клієнт надіслав коректні дані для входу в систему (логін та пароль), API повертає JWT токен який містить всю необхідну інформацію про клієнта. Клієнт зберігає ці дані у себе, в даному випадку в якості клієнта виступає браузер, який зберігає отриманий токен в cookies, для подальшого використання.

Кожен наступний раз коли клієнт відправляє запит до серверу, він читає JWS з cookies і додає його до запиту.

Сервер в свою чергу бере отриманий токен від клієнта, та валідує його, якщо токен пройшов етап валідації, тобто сервер зміг підтвердити його цілісність, а також даний токен не наявний в чорному списку. Після етапу автентифікації, сервер також перевіяє чи має даний клієнт право доступу до даних, котрі він хоче отримати від серверу (даний етап називається етапом авторизації). Якщо JWT токен успішно пройшов ці два етапи, то в такому випадку сервер надає всі необхідні дані клієнту.

Термін життя JWT токену встановлено до 10 днів, після того як в токена закінчився термін придатності, клієнту буде необхідним пройти повторну автентифікацію, після чого він отримає новий токен.

Якщо на етапі автентифікації сервер не зміг підтвердити цілісність токену, то клієнт отримає HTTP відповідь зі статус-кодом 401, та проханням, пройти повторну автентифікацію. Якщо етап автентифікації пройдено успішно, а на етапі авторизації виникла помилка, то в такому випадку клієнт отримає HTTP відповідь зі статус-кодом 403, що означає що клієнт не має доступу до запитуваних даних.

3.2. Використання Docker та Docker Compose

Додаток має архітектуру яка складається з чотирьох компонентів:

- API – дозволяє взаємодіяти клієнтам з сервером;
- UI – графічний інтерфейс користувача;
- Worker – використовується для зміни статусу подій;
- База даних – використовується для збереження даних.

Кожен компоненти архітектури має Dockerfile, який описує кроки для створення Docker Image, це означає що всі компоненти додатку мають можливість для розгортання в будь-якому середовищі, в незалежності від платформи. Docker гарантує, що додаток буде працювати ідентично на всіх платформах. Також дана особливість пришвидшує розробку на декількох комп’ютерах одночасно, так як додаток запускається в ізольованому середовищі, та не має зовнішніх залежностей.

Для пришвидшення локальної розробки було використано Docker Compose. Docker Compose – це інструмент для визначення та запуску багатоконтейнерних додатків. Для роботи з Docker Compose було створено файл «docker-compose.yaml» який містить в собі налаштування служб Docker додатків. Після чого за допомогою однієї команди можна створити та запустити всі служби з файлу конфігурації.

Переваги використання Docker Compose:

- можливість взаємодіяти з декількома контейнерами одночасно, використовуючи прості та зручні команди;
- можливість мати кілька ізольованих середовищ на одному хості;
- можливість збереження даних (volume data) при створенні контейнерів, особливо актуально для контейнера який містить в собі базу даних, так як необхідно зберігати дані, навіть після аварійної зупинки контейнера;
- можливість перебудовувати тільки ті контейнери котрі змінились, що зменшує необхідну кількість затраченого часу на перебудову контейнерів.

РОЗДІЛ 4
ОПИС ЗДОБУТИХ РЕЗУЛЬТАТІВ

Головна сторінка додатку зображена на рис. 4.1. Головна сторінка складається з декількох частин.

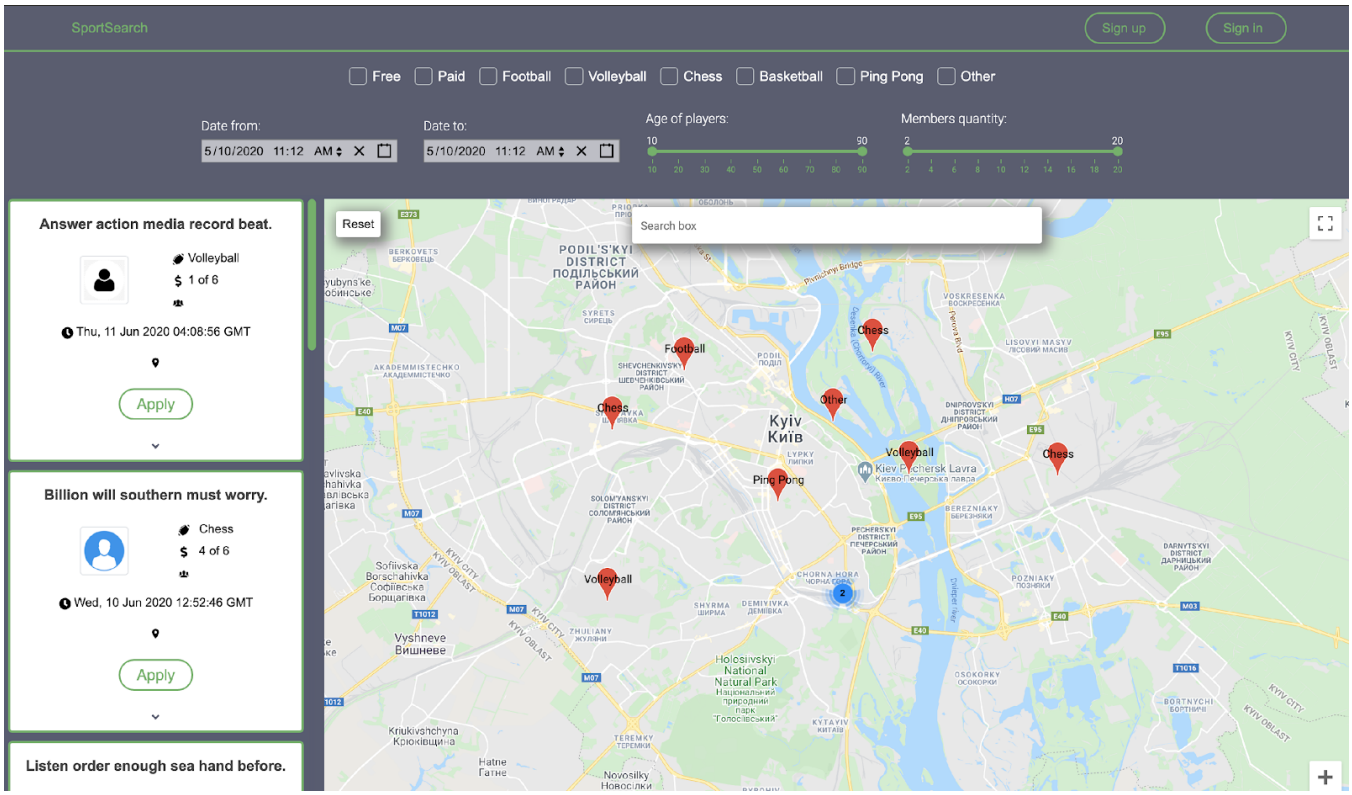

Рис. 4.1. Головна сторінка веб-додатку

Зверху розташована навігаційна панель, яка в залежно від того чи користувач увійшов до системи відображає різні елементи керування, в даному випадку користувач неавторизований, тому відображаються кнопки для реєстрування та входу до системи. Нижче знаходиться форма для фільтрування подій в залежності від заданих параметрів, можна фільтрувати за: видом спорту, віком та кількістю учасників, часом початку та кінця. Ліворуч розташована бокова панель яка відображає події в залежності від обраних фільтрів в картковому вигляді. Картка містить в собі коротку інформацію про подію, а також кнопку для подачі заявки на участь. Більшу частину головної сторінки займає мапа, на якій відображені події в

залежності від обраних фільтрів. Подія відображається у вигляді маркеру червоного кольору, а також містить надпис відповідного виду спорту. Для більш зручної взаємодії з картою, на карті присутня кластеризація (якщо в одному місці знаходиться багато подій то вони будуть об’єднані в один маркер синього кольору). При натисканні на маркер на мапі в боковій панелі буде відображено відповідну подію.

Для того щоб створити подію, або подати заявку на участь в обраній події потрібно створити профіль та авторизуватись в системі. Сторінка створення профілю зображена на рис. 4.2.

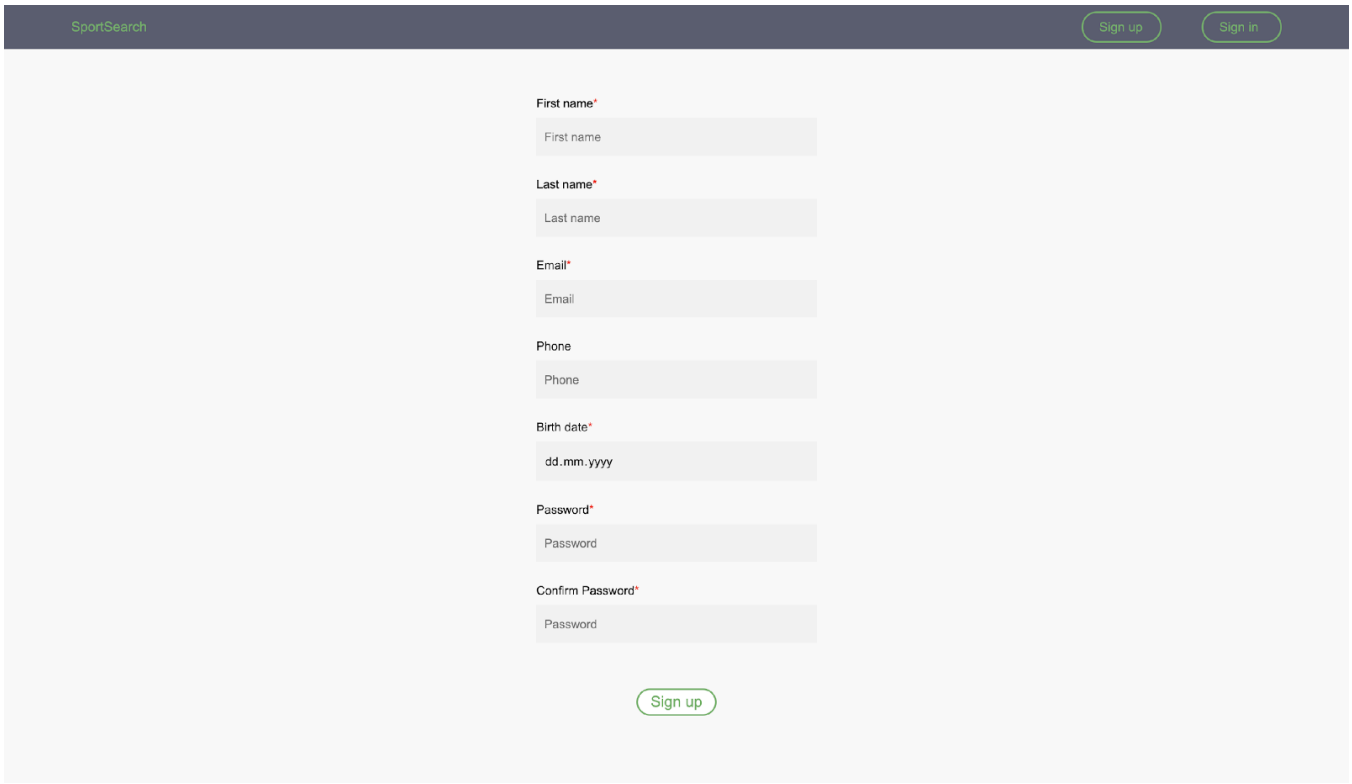

Рис. 4.2. Сторінка створення профілю

Після авторизації всі можливості передбачені в веб-додатку, стають доступними для користувача.

Щоб створити подію користувач повинен перейти на сторінку створення подій (відповідне посилання можна знайти в навігаційний панелі, після успішної авторизації). Сторінка створення нової події зображена на рис. 4.3.

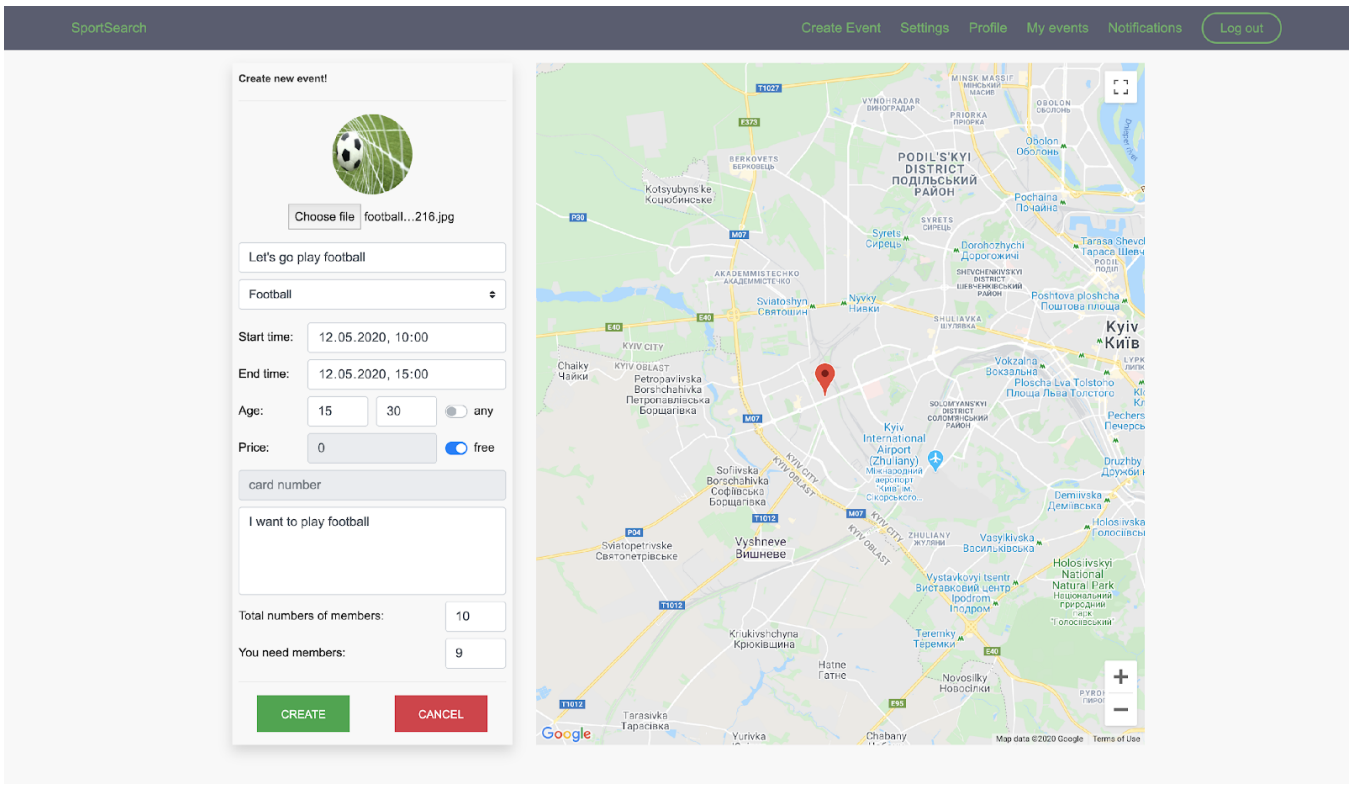

Рис. 4.3. Сторінка створення нової події

Сторінка створення нової події складається з двох частин: форма для заповнення відповідних даних та мапи.

При створені події потрібно обов’язково вказати коротку, описову назву, вид спорту якому буде присвячена подія, дата та час початку та закінчення, обмеження по віковій групі, короткий опис події та максимальну кількість учасників, які можуть приймати участь. Серед необов’язкових параметрів присутня можливість добавити фото (логотип).

Мапа містить червоний маркер який можна переміщати по карті, маркер відображає місце проведення нової події, за замовчуванням маркер буде встановлено та поточне місцезнаходження користувача, який створює подію, за умови що користувач надав доступ веб-додатку до поточного місцезнаходження.

Що зберегти подію потрібно натиснути на кнопку «CREATE», після чого дані будуть збережені, а користувач буде перенаправлений на сторінку тільки, що створеної події. Сторінка події зображена на рис. 4.4.

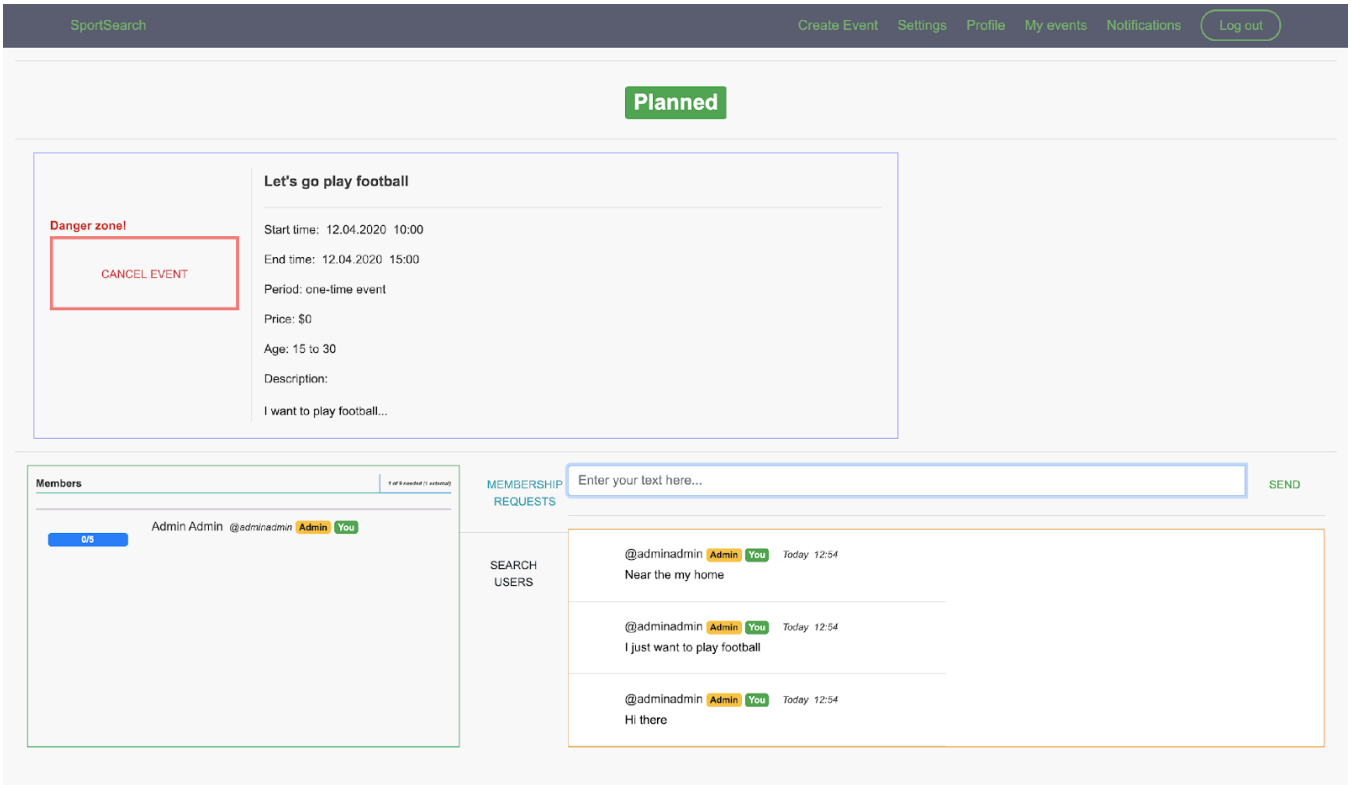

Рис. 4.4. Сторінка події

Сторінка події складається з декількох частин: вся інформація про подію відображена в верхній частині сторінки, також там знаходиться кнопка для відміни події.

Панель для керування учасниками події, розташована в нижньому лівому куту. Дана панель за замовчуванням відображає всіх учасників які були добавлені до події адміністратором (користувач який створив подію). За допомогою кнопок які розташовані праворуч від панелі можна змінювати призначення панелі. Одним з варіантів використання панелі може бути перегляд та підтвердження заявок на участь від інших користувачів. Також панель керування учасниками можна використовувати для пошуку нових учасників.

Праворуч від панелі керування учасниками розташована панель чату, за допомогою якого всі учасники події можуть спілкуватися між собою.

Для перегляду всіх події в який користувач брав участь існує сторінка «Мої події», посилання на яку можна знайти в навігаційному барі, за умови що користувач авторизований в системі. Сторінка «Мої події» зображена на рис. 4.5.

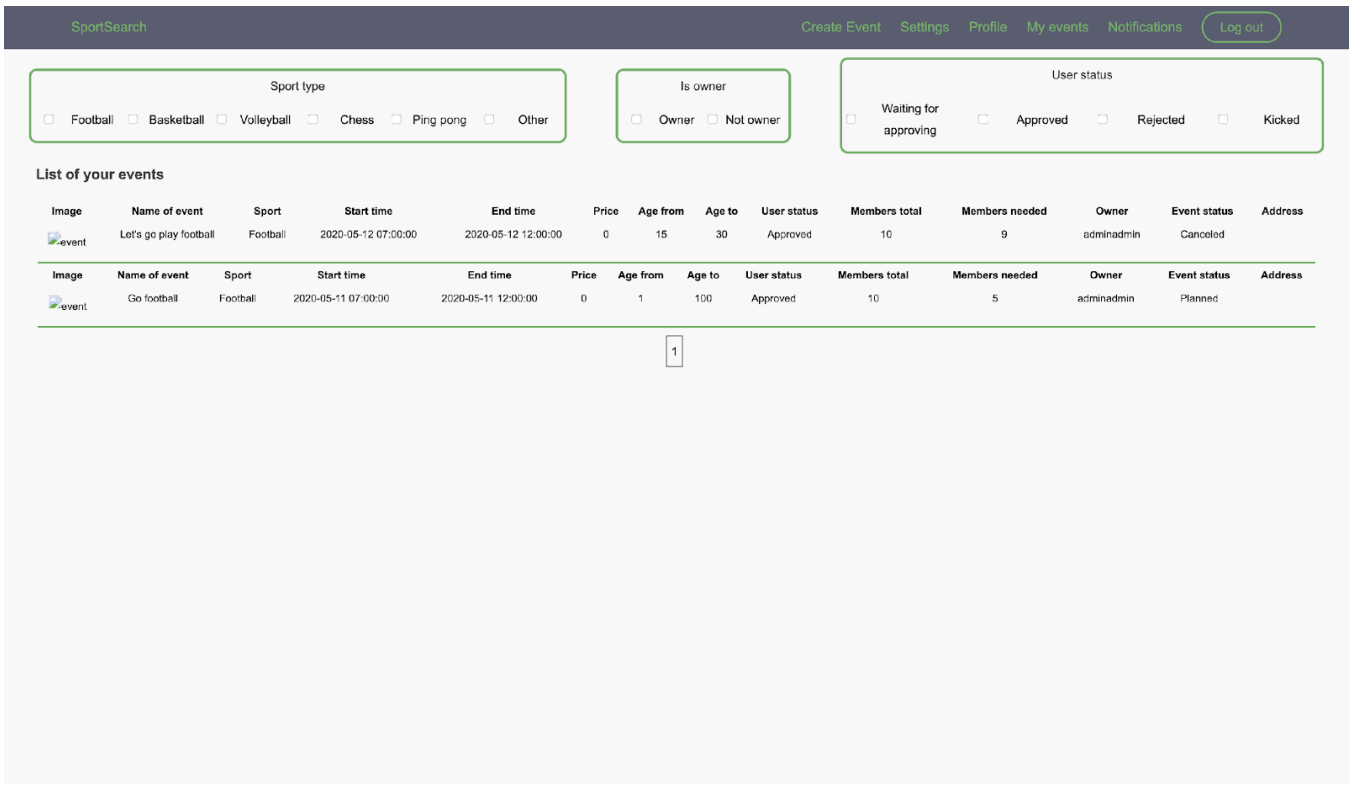

Рис. 4.5. Сторінка «Мої події»

Сторінка «Мої події» складається з двох частин: форма фільтрування, список відфільтрованих подій. Форма фільтрування має всі необхідні компоненти для фільтрування подій за: видом спорту, за статусом користувача в відповідній події, а також.

ВИСНОВКИ

В кваліфікаційній роботі було розглянуто сучасні метод та технологій які використовуються для розробки веб-додатків. На сьогоднішній день розробка веб-додатків є найпопулярнішою сферою програмування, яка все ще активно набирає популярність, а також активно розвивається. Було проведено ознайомлення з особливості проєктування та реалізації веб-додатків, були використані сучасні технології які на даний момент є найзатребуванішими в сфері веб-розробки.

Було спроєктовано та реалізовано веб-додаток пошуку людей для спільних занять різними видами спорту. Додаток спроєктовано на базі клієнт-серверної архітектури, для розробки серверної частини використано інтерпретовану мову програмування Python, з використанням бібліотеки Flask, для клієнтської частини – TypeScript, а саме бібліотеку React. Завдяки використанню цих двох мов вдалось пришвидшити розробку додатку, в порівнянні з іншими мовами та технологіями. Крім класичних архітектурних компонентів додаток має компонент «Worker», який відповідає за оновлення статусу подій. Для зберігання даних використано реляційну базу даних під управлінням системи керування базою даних PostgreSQL.

Додаток має простий та зрозумілий інтерфейс, яким зручно користуватись. В порівнянні з конкурентами, розроблений додаток має мапу, та великий вибір фільтрів, дані особливості допомагають пришвидшити пошук спортивних активностей, що й являється головною функцією додатку. Веб-додаток має ввесь необхідний функціонал для організації сумісної спортивної активності.

Слід зазначити що система спроектована таким чином, що може бути легко розширена для організації будь-яких інших групових активностей.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. https://insights.stackoverflow.com/survey/2020#developer-profile-developer-type-all-respondents
2. Python 3.7.7 Documentation [Електронний ресурс] : [Веб-сайт]. – Режим доступу: https://docs.python.org/3.7/
3. https://www.typescriptlang.org/docs/
4. Docker Documentation [Електронний ресурс] : [Веб-сайт]. – Режим доступу: https://docs.docker.com
5. https://en.wikipedia.org/wiki/Internet_Protocol
6. https://restfulapi.net/rest-architectural-constraints/
7. https://reactjs.org/docs/getting-started.html
8. https://flask.palletsprojects.com/en/2.0.x/
9. https://stackoverflow.com/
10. https://www.tutorialspoint.com/system_analysis_and_design/system_design.htm
11. https://towardsdatascience.com/system-design-101-b8f15162ef7c
12. Доусон М. Программируем на Python / Доусон М. – 3-е изд. – СПб. : Питер, 2014. – 416 с.
13. Бхамидипати К., SQL. Справочник программиста. – М.: Издательство ЭКОМ, 2010. – 364 с. – С. 37-45.
14. PostgreSQL 12.2 Documentation [Електронний ресурс] : [Веб-сайт]. – Режим доступу: https://www.postgresql.org/docs/12/index.html

ДОДАТОК А

Лістинг модуля роботи з учасниками події

```
from datetime import date, datetime

from flask import Response, request
from flask_socketio import join_room, leave_room
from sqlalchemy import or_

from api.models import db, Feedback, User, UserInEvent
from . import socketio
from .notification_service import send
from .useful_decorators import error_func, visitor_allowed_socket

NAMESPACE = '/members'



@socketio.on('join', namespace=NAMESPACE)
@visitor_allowed_socket
def join(*args, **kwargs):
    event = args[0]
    join_room(event.id)



@socketio.on('leave', namespace=NAMESPACE)
@visitor_allowed_socket
def leave(*args, **kwargs):
    event = args[0]
    leave_room(event.id)
```

```
@socketio.on('get_active_members', namespace=NAMESPACE)
@visitor_allowed_socket
def get_event_members(*args, **kwargs):
    """
    Function that returns a list of event members
    :return:
    """
    event = args[0]
    user = args[1]
    data = args[2]

    try:

        # get all members that event has
        members = db.session.query(User).filter(
            or_(*(User.id == q_user.user_id
                  for q_user in event.users
                  # user status has to be 'approved'
                  if q_user.user_event_status_id == 2))
        ).order_by(
            User.first_name,
            User.last_name
        ).all()

        socketio.emit(
            'receive_active_members',
            {
                'members': [
```

```
                {
                    'id': member.id,
                    'name': member.first_name,
                    'surname': member.last_name,
                    'nickname': member.nickname,
                    'rating': member.rating,
                    'request_user_rating':
                        getattr(db.session.query(Feedback).filter(
                            Feedback.user_from_id == user.id,
                            Feedback.user_to_id == member.id,
                            Feedback.event_id == event.id,
                        ).first(), 'rating', 0) if member.id != user.id else 0,
                    # if there are no such object, return 0
                    # also, it is the turn of a request user, there is no need to query the database,
so return 0
                    'image_url': member.image_url,
                } for member in members
            ],
        },
        room=event.id if data.get('update_all_users') else request.sid,
        namespace=NAMESPACE,
    )

except Exception:
    return error_func()


@socketio.on('kick_member', namespace=NAMESPACE)
@visitor_allowed_socket
def kick_user(*args, **kwargs):
```

```
"""
Function that kicks a user from an event.
Required parameters:
    - kick_user_id : (int) { required } : id of user function need to kick from an event
:return:
"""

event = args[0]
user = args[1]
data = args[2]
user_is_owner = user.id == event.owner_id

if not user_is_owner:
    return error_func(error_status=403,
                      error_description='Only event owner can kick users.',
                      error_message='USER_NOT_OWNER_FORBIDDEN', )
if event.event_status_id != 1:
    return error_func(error_status=400,
                      error_description='Users can be kicked only from active events.',
                      error_message='EVENT_NOT_PLANNED', )

try:
    kick_id = data.get('kick_user_id')

    if not kick_id:
        return error_func(error_status=400,
                          error_description='Kick user id was not provided correctly.',
                          error_message='USER_ID_NOT_FOUND', )

    if user.id == kick_id:
```

```
            return error_func(error_status=403,
                              error_description='Admin/owner can not kick themselves.',
                              error_message='USER_OWNER_FORBIDDEN', )

        kick_in_event = db.session.query(UserInEvent).filter(
            UserInEvent.user_id == kick_id,
            UserInEvent.event_id == event.id,
        ).first()

        kick_in_event.user_event_status_id = 4
        send(3, user_id=kick_id, event_id=event.id)  #user kick notification
        db.session.commit()

    except Exception:
        return error_func(error_status=400,
                          error_description='You provided incorrect data.',
                          error_message='INCORRECT_DATA_PROVIDED', )

    return Response(status=200)



@socketio.on('grant_user', namespace=NAMESPACE)
@visitor_allowed_socket
def grant_request_member(*args, **kwargs):
    """
    Function that kicks a user from an event or accepts for an event.
    Required parameters:
        - target_user_id : (int) { required } : id of user that will be granted
        - target_user_status: (int) { required } : status of user that will be granted to him
    :return:
```

```
    """

    event = args[0]  #event id
    user = args[1]  #owner id
    data = args[2]  #{'event_id': '19', 'user_id': 17, 'target_user_id': 20, 'target_user_status':
2}
    user_is_owner = user.id == event.owner_id

    if not user_is_owner:
        return error_func(error_status=403,
                          error_description='Only event owner can accept or decline users.',
                          error_message='USER_NOT_OWNER_FORBIDDEN', )
    if event.event_status_id != 1:
        return error_func(error_status=400,
                          error_description='Users can be only accepted to an planned event.',
                          error_message='EVENT_NOT_PLANNED', )

    try:
        target_id = data.get('target_user_id')
        target_status = data.get('target_user_status')

        if not (target_id and target_status):
            return error_func(error_status=400,
                              error_description='Some of required fields missed.',
                              error_message='INCORRECT_DATA_PROVIDED', )

        if target_status not in (2, 3):
            return error_func(error_status=400,
                              error_description='User status can be changed only to 2 or 3.',
                              error_message='INCORRECT_STATUS_PROVIDED', )
```

```
    if user.id == target_id:
        return error_func(error_status=403,
                          error_description='Admin/owner can not grand themselves.',
                          error_message='USER_OWNER_FORBIDDEN', )

    target_user = db.session.query(User).filter(
        User.id == target_id,
    ).first()

    today = date.today()
    user_age = today.year - target_user.birth_date.year - \
           ((today.month, today.day) < (target_user.birth_date.month,
target_user.birth_date.day))
    if not (event.age_from <= user_age <= event.age_to):
        return error_func(error_status=400,
                          error_description='User does not fit age restrictions.',
                          error_message='USER_AGE_RESTRICTED', )

    target_in_event = db.session.query(UserInEvent).filter(
        UserInEvent.user_id == target_id,
        UserInEvent.event_id == event.id,
    ).first()

    if not target_in_event:
        return error_func(error_status=404,
                          error_description='User does not have request for this event.',
                          error_message='USER_NOT_REQUESTED_EVENT', )

    target_in_event.user_event_status_id = target_status
```

```
        # User notification
        if target_status == 2:
            send(1, target_id, event.id)    #request has been approved
        elif target_status == 3:
            send(2, target_id, event.id)    #request has been rejected

        db.session.commit()

    except Exception:
        return error_func(error_status=400,
                          error_description='You provided incorrect data.',
                          error_message='INCORRECT_DATA_PROVIDED', )

    return Response(status=200)

@socketio.on('get_request_users', namespace=NAMESPACE)
@visitor_allowed_socket
def get_request_members(*args, **kwargs):
    """
    Function that returns list of requests to participate in event.
    :return:
    """

    event = args[0]
    user = args[1]
    user_is_owner = user.id == event.owner_id

    if not user_is_owner:
        return error_func(error_status=403,
                          error_description='Only admin/owner can get membership requests.',
```

```
                error_message='USER_NOT_OWNER_FORBIDDEN', )
try:
    # check initial filters
    # we need to do this, because if filters will be empty,
    # query will return all users in database (filter() behaviour)
    filters = [*(User.id == q_user.user_id
                for q_user in event.users
                # user status has to be 'waiting for approve'
                if q_user.user_event_status_id == 1)]
    if filters:
        # get all requested members
        members = db.session.query(User).filter(
            or_(*(User.id == q_user.user_id
                  for q_user in event.users
                  # user status has to be 'waiting for approve'
                  if q_user.user_event_status_id == 1))
        ).order_by(
            User.first_name,
            User.last_name
        ).all()
    else:
        members = []

    today = date.today()

    socketio.emit(
        'receive_request_members',
        {
            'members': [
                {
```

```
                'id': member.id,
                'name': member.first_name,
                'surname': member.last_name,
                'nickname': member.nickname,
                'rating': member.rating,
                'age': today.year - member.birth_date.year -\
                    ((today.month, today.day) < (member.birth_date.month,
member.birth_date.day)),
                'image_url': member.image_url,
            } for member in members
        ],
    },
    room=request.sid,
    namespace=NAMESPACE
  )
 except Exception:
  error_func()

@socketio.on('rate_user', namespace=NAMESPACE)
@visitor_allowed_socket
def rate_user(*args, **kwargs):
 """
 Function that create a feedback for the user.
 Required parameters:
  - target_user_id : (int) { required } : id of user function need to kick from an event
  - mark : (int) { required } : mark that request user gave to the tarhet user
  - comment : (str) { optional } : comment that user can leave
 :return:
 """
```

```
event = args[0]
user = args[1]
data = args[2]

try:
    target_user_id = data.get('target_user_id')
    mark = data.get('mark')
    comment = data.get('comment')

    if not target_user_id:
        return error_func(error_status=400,
                          error_description='User id ws not found.',
                          error_message='USER_ID_NOT_FOUND', )
    if not mark:
        return error_func(error_status=400,
                          error_description='Rate for user is empty.',
                          error_message='USER_RATE_NOT_FOUND', )
    if target_user_id == user.id:
        return error_func(error_status=400,
                          error_description='User can not rate themselves.',
                          error_message='USER_RATE_FORBIDDEN', )
    if event.event_status_id != 2:
        return error_func(error_status=400,
                          error_description='Only when event was finished feedback can be left.',
                          error_message='EVENT_NOT_FINISHED', )

    feedback = db.session.query(Feedback).filter(
        Feedback.event_id == event.id,
        Feedback.user_from_id == user.id,
        Feedback.user_to_id == target_user_id,
```

```
    ).first()

    if not feedback:
        feedback = Feedback(
            rating=mark,
            text=comment if comment else None,
            feedback_time=datetime.utcnow(),
            event_id=event.id,
            user_from_id=user.id,
            user_to_id=target_user_id,
        )

        db.session.add(feedback)

    else:
        feedback.rating = mark
        feedback.text = comment or None

    send(6, user_id=target_user_id) #user notification (received_feedback)

    db.session.commit()

    User.update_rating(target_user_id)

except Exception as ex:
    return error_func(error_status=400,
                      error_description='You provided incorrect data.',
                      error_message='INCORRECT_DATA_PROVIDED')

return Response(status=200)
```