НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено

Завідувач кафедри

_____ О.В. Коваль
(підпис) (ініціали, прізвище)

“___”_____2019р.

ДИПЛОМНА РОБОТА

на здобуття ступеня бакалавра

з напряму підготовки 6.050103 “ Програмна інженерія “

на тему розробка спеціалізованого інтернет-сервісу «Відкритий спортмайданчик з е-сервісами» (back-end).

Виконав: студент 4 курсу, групи ТВ-51

Голопапа Ярослав Григорович _____ _____
(прізвище, ім’я, по батькові)
(підпис)

Керівник доц. Ковальчук Артем Михайлович _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент_____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший , бакалаврський

Напрям підготовки 6.050103 “ Програмна інженерія”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.В. Коваль
(підпис)
”___”_____2019р.

ЗАВДАННЯ

на дипломну роботу студенту
Голопапи Ярослава Григоровича
(прізвище, ім’я, по батькові)

1. Тема роботи розробка спеціалізованого інтернет-сервісу «Відкритий спортмайданчик з е-сервісами» (back-end).

керівник роботи___доц. Ковальчук Артем Михайлович_____
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”__”___ 201_р. № __

2. Строк подання студентом роботи _____7 червня 2019 р._____

3. Вихідні дані до роботи __Мова програмування JavaScript, середовище розробки WebStorm

4.Зміст розрахунково-пояснювальної записки (перелік завдань, які потрібно розробити) _Аналіз існуючих сервісів. Проектування системи та розробка

архітектури, розробка розширеного функціоналу для авторизованого користувача.

5. Перелік ілюстративного матеріалу
Архітектура системи, робота клієнта у системі, способи зберігання даних, використані технології

6. Дата видачі завдання ”01” грудня 2018р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Вивчення та аналіз задачі	01.12.18-12.01.19	
2	Розробка архітектури та загальної структури системи	12.01.19-10.02.19	
3.	Розробка структур окремих підсистем	10.02.19-15.02.19	
4.	Програмна реалізація системи	15.02.19-12.03.19	
5.	Оформлення пояснювальної записки	13.03.19-30.05.19	
6.	Захист програмного продукту	17.05.19	
7.	Передзахист	31.05.19	
8.	Захист		

Студент _____ **Голопапа Я.Г.**
(підпис) (прізвище та ініціали,)

Керівник роботи _____ **Ковальчук А.М**
(підпис) (прізвище та ініціали,)

АНОТАЦІЯ

Останні роки набувають все більшого поширення різноманітні системи бронювання. Сфера їх застосування розростається з кожним днем. Від бронювання ресторанів до бронювання квитків в кіно.

Ці системи націлені на економію часу людей і гарантію того, що бронювання буде успішне і людина не витратить свій час майно.

Ціль роботи – створити сервіс, який би допомагав спортсменам і простим людям бронювати спортивний майданчик для заняття різними видами спорту.

Тому, було запропоновано дослідити можливість застосування алгоритмів та програмних засобів для автоматизації пропуску людей на спортивний майданчик с ціллю підвищення швидкості обігу людей на спортивному майданчику.

Записка містить 51 сторінки 28 рисунків.

ABSTRACT

In recent years, the various types of booking systems are becoming increasingly popular. The scope of their application grows day by day. From restaurant reservations to movie tickets.

These systems are aimed at saving people's time and guaranteeing that the booking will be successful and the person will not spend their time on the property.

The purpose of the work is to create a service that would help athletes and ordinary people to book a sports ground for various kinds of sports.

Therefore, it was proposed to investigate the possibility of using algorithms and software for automating people's passage on a sports ground with the aim of increasing the speed of people's circulation on a sports ground.

The note contains 51 pages 28 images.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ЕЦП – електроний цифровий підпис;

СКБД – система керування базами даних;

API (англ. Application Programming Interface) – прикладний програмний інтерфейс;

RFID (англ. Radio frequency identification) — радіочастотна ідентифікація;

LTE (англ. Long Term Evolution») — назва мобільного протоколу передавання даних

БД – база даних;

SB (англ. ScrollBar) – віджет для переміщення по файлу;

MVC – (англ. model-view-controller) – схема поділу програмного продукту, призначеного для користувацького інтерфейсу і керуючої логіки на три окремих компоненти: модель, уявлення і контролер;

ВСТУП

Швидкий розвиток новітніх інформаційних технологій дає можливість створювати нові системи, сервіси, додатки для полегшення і прискорення різних дій людини в нашому світі.

За допомогою різноманітних сервісів, людина може з легкістю вирішити свої повсякденні проблеми з допомогою мобільного телефону чи планшетного комп'ютера. Це є зручний інструмент, який дозволяє за декілька хвилин забронювати чи купити квитки на потяг, замовити доставку їжі, забронювати квитки в кіно та інші дії.

На сьогодні, ми маємо значну кількість відкритих сервісів в мережі «Інтернет», які дозволяють за декілька хвилин налаштувати ваш особистий акаунт і почати використовувати їх.

Основним завданням всіх сервісів – це розробити зручний та швидкий клієнтський додаток, за допомогою якого можливо використовувати зазначені дії. Цей додаток повинен мати правильну архітектуру, за допомогою якої, всі данні користувачів будуть в безпеці і сервіс буде зручний в обслуговуванні.

Пропонується розглянути серверну частину сервісу, який дозволяє створювати події на вказаному спортивному-майданчику, тим самим бронювати час проведення цієї спортивної події.

Програма представляє собою серверний додаток, який обробляє, зберігає, відтворює інформацію від клієнтського додатку, а саме від кінцевого користувача.

Для надійності передачі даних було вирішено використовувати протокол передачі даних HTTPS, який дозволяє швидко та зручно шифрувати данні між клієнтом і сервером.

Записка містить 5 розділів.

У першому розділі описується огляд існуючих рішень систем бронювання

У другому розділі проводиться інструментів для розробки

У третьому розділі описані засоби реалізації програмної системи.

У четвертому розділі описана методика роботи користувача з програмною системою.

1. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ СЕРВІСІВ БРОНЮВАННЯ

Швидкість розвинення технології і збільшення кількості потреб відчиняють можливості майбутнього. За допомогою цих технологій було створено велику кількість сервісів бронювання. За допомогою сервісів бронювання у людини з’являєтеся можливість зекономити час, а для самого сервісу знайти нових клієнтів. Мобільний додаток являєтеся найзручнішим для використання подібних сервісів, так як потребує лише телефон, котрий являється невід’ємною частиною на сьогоднішній день. У цьому розділі пропонується розглянути основні підходи до бронювання в різних сферах послуг, та розглянути основні види бронювання, які необхідні в повсякденному житті.

1.1 Бронювання квитків в кінотеатрах

Системи онлайн бронювання квитків кінотеатрах мають такі види:

— Централізовані системи бронювання

— Локальні системи бронювання

Централізована система бронювання являє собою сервіс, який вміщує в собі велику кількість кінотеатрів. Це дає змогу користувачу користуватися цим сервісом в різних містах, які підключені до сервісу. Також це дає змогу переглянути всі сеанси в кінотеатрах и вибрати найзручніший час для перегляду кінострічки.

Локальні системи бронювання розраховані на одну мережу кінотеатрів. Цей вид системи розроблюється самим кінотеатром, в більшому числі випадків, як web-додаток. Також ця система дозволяє в майбутньому приєднатися до централізованої системи бронювання.

1.2 Бронювання номерів готелів

Системами онлайн бронювання готелів називаються системи, які відображають наявність реальних номерів в готелях, фотографії цих номері і

вільні інтервали для проживання. Ці системи заточені під швидкий запит на бронювання, який займає до 15 хвилин. Це дає змогу зробити централізований сервіс, який буде містити готелі по всьому світу.

В більшості випадків, зручніше за все використовувати web-додаток сервісів бронювання готелів.

На цих сайтах можна зустріти два види бронювань:

— Істинне онлайн-бронювання

— Псевдоонлайн-бронювання

Істинне онлайн-бронювання підрозуміває під собою коли користувач робить вибір, бронює і оплачує номер самостійно через платіжні системи або безпосередньо на сайті готелю. Для оплати клієнт використовує банківську картку, дані якої передаються в готель безпосередньо, і через хвилину отримує лист-підтвердження на електронну скриньку, що підтверджує бронь.

1.3 Бронювання місць в ресторані

Система онлайн-бронювання столів є додатком, що дозволяє будь-якому користувачу побачити поточний стан забронюваних і вільних столів, визначитися з вибором столу для себе, замовити страву чи напій з меню ресторану до часу приходу. Також стають все більш популярні сервіси доставки їжі.

1.4 Бронювання спортивних майданчиків

Основною ідеєю бронювання спортивних майданчиків – це надати гарантію користувачу, про те що в вибраний час і на вибраному майданчику будуть всі необхідні умови для заняття вибраним видом спорту. Також сервіс дає можливість зібрати команду для конкретного виду спорту.

1.4.1 Аналіз існуючих сервісів бронювання спортивних майданчиків

Яскравим прикладом бронювання спортивних майданчиків є інтернет-сервіс «Arena-Booking». Цей сервіс дає можливість переглянути доступні спортивні площадки в вашому місті та зробити онлайн-бронювання на вказаний час.

Перш за все сервіс має інтуїтивно-зрозумілий інтерфейс (рисунок 1.1). На першій сторінці сайту можна відразу переглянути популярні місця для спорту, дізнатися детальну інформацію про місце, ціну та вільний час.

Необхідно відмітити те, що даний сервіс дозволяє не тільки бронювати місце для спорту, а ще дає можливість записатися на секцію в зручний для вас час.

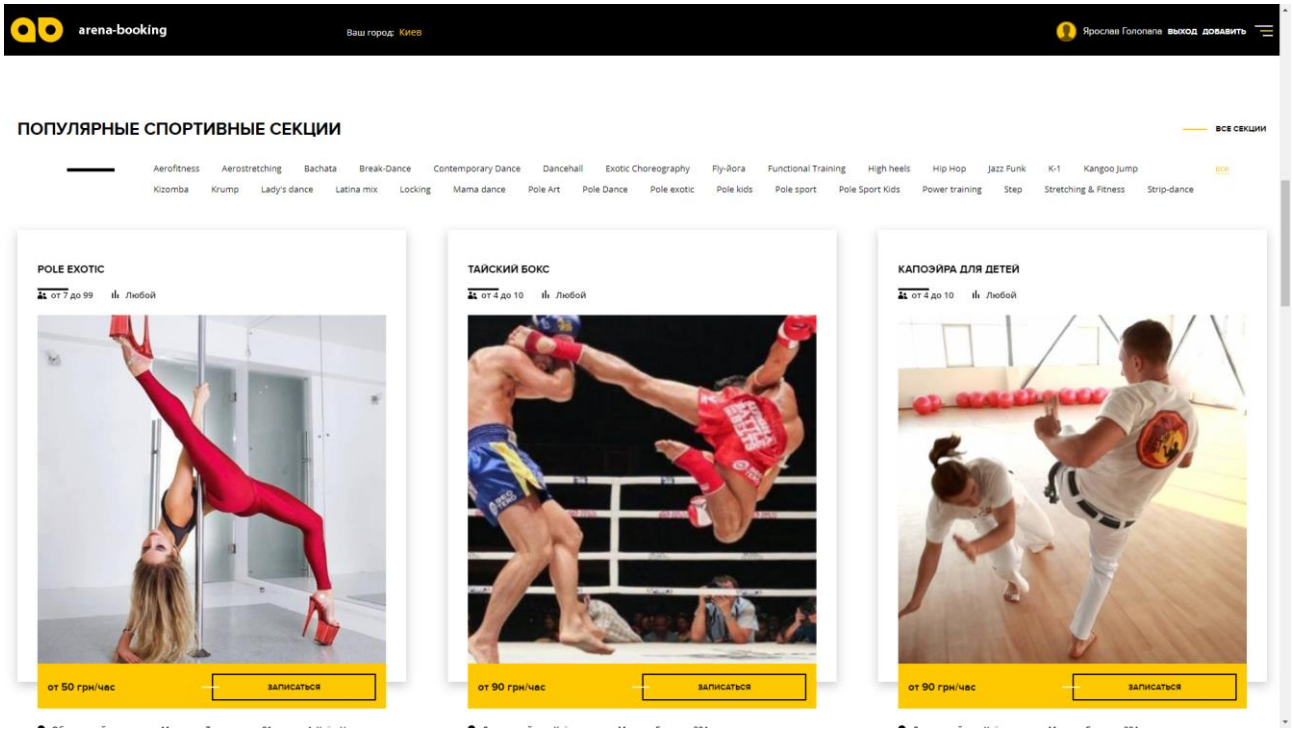

Рисунок 1.1 – Сайт сервісу «Arena-booking»

Для бронювання спортивного місця необхідно зареєструватися на сайті. Під час цієї процедури, сервіс запитає вашу електрону адресу і мобільний

номер. Необхідність вказувати мобільний номер є поширеною практикою систем захисту від спаму і зручності в контактуванні сервісу і клієнту. Також на вказану електронну адресу прийде лист з посиланням на підтвердження акаунтур[2].

Розділ вибору спортивної площадки має дуже зручний інтерфейс, за допомогою якого можна вибрати місто, вид спорту, зручний район міста, необхідний час і дату для проведення спортивної події. Це дає змогу зробити саме оптимальне рішення. Також на цій сторінці є зручна карта з маркерами, яка графічно відображає місцезнаходження кожного майданчику з вибраними фільтрами (рисунок 1.2).

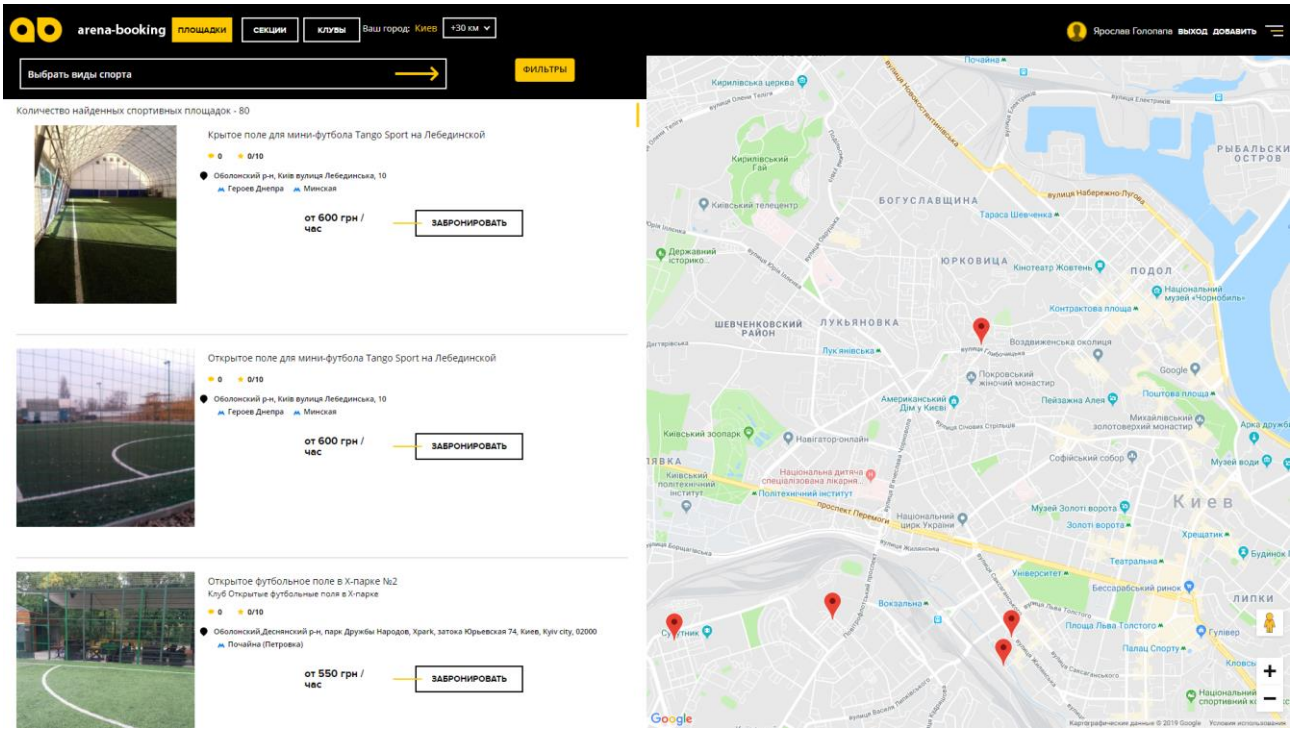

Рисунок 1.2 – Сторінка вибору спортивного майданчику

Після ознайомлення з підходящим місцем спорту, користувачу пропонується забронювати його на зручний час (рисунок 1.3). Під час цієї процедури буде представлено таблицю, з допомогою якої можна зробити бронювання. Мінімальний інтервал бронювання складає одна година. Після

успішного бронювання користувач отримує лист на електронну адресу з підтвердженням бронювання і повною інформацією про місце і час проведення.

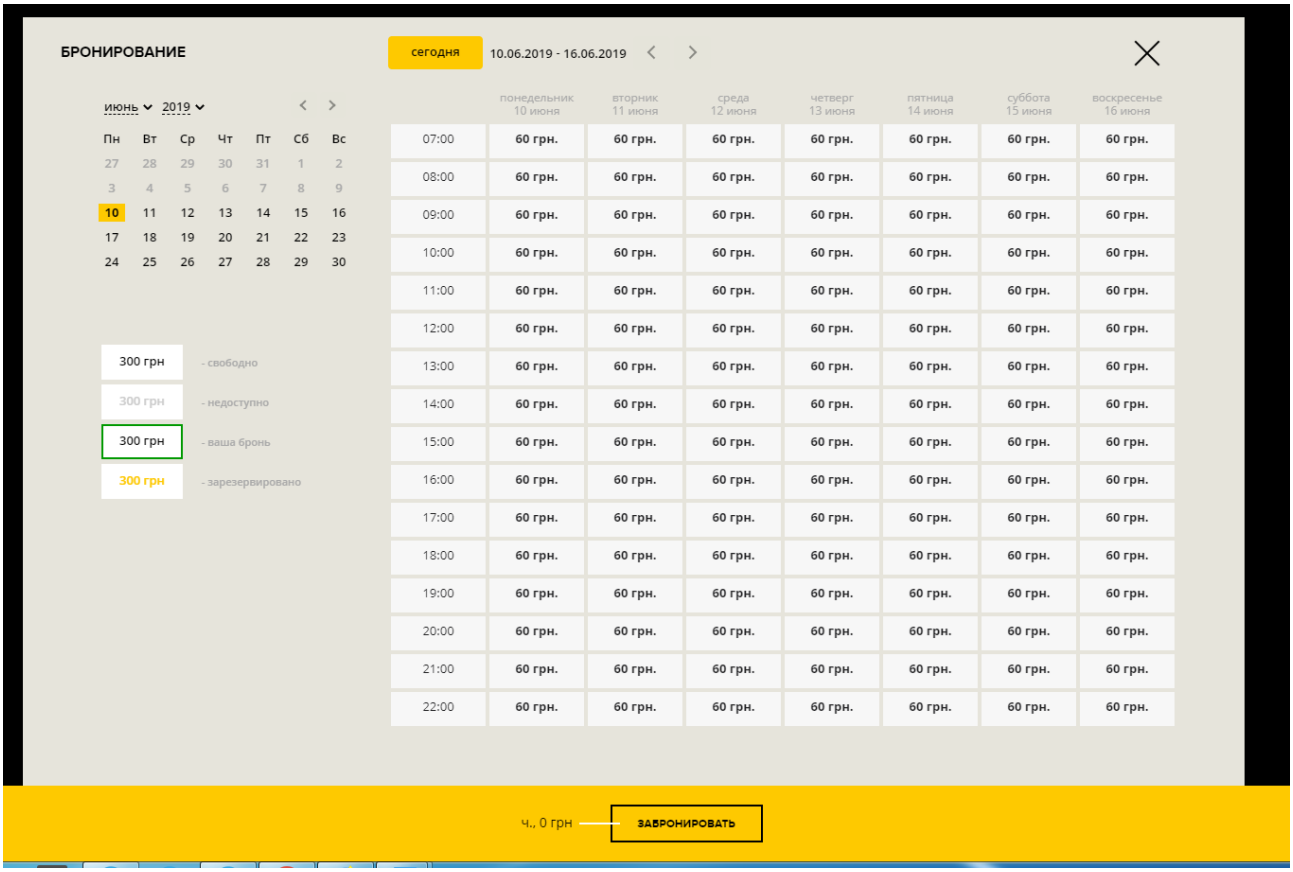

Рисунок 1.3 – Таблиця вибору бронювання майданчику

1.4.2 Аналіз проблеми бронювання спортивного майданчику

Невід'ємною частиною життя людини – є спорт. Тому значна кількість людей, постійно виділяє свій час на різноманітні спортивні заняття. Для цього людина вибирає необхідне місце та зручний час для тренувань. Після цього прямує до місця проведення і тренується.

Під час рішення проведення занять спортом, людина може зіткнутися з такими проблемами:

— Відсутні вільні місця на спортивному майданчику

— Відсутнє необхідне знаряддя чи обладнання для спорту

— Недостатня кількість людей для вибраного виду спорту

Ці проблеми є доволі поширені в наш час. Вони виникають через те, що нема врегульованого графіку проведення спортивних подій на майданчиках.

Тому було запропоновано створити систему, яка дозволяє контролювати прохід і кількість людей на спортивному майданчику, які займаються спортом в відведений час.

Головною сутністю цієї системи – є подія. Тобто людина створює подію і дає можливість приєднатися до неї іншим користувачам, тим самим отримує змогу потрапити на спортивний майданчик в відведений час. Користувачі в свою чергу можуть переглядати ці події і приєднуватися до них.

Для вирішення питання контролю спортивних майданчиків, було прийняте рішення встановити спеціальне обладнання, яке дозволяє контролювати пропуск людини на майданчик. Обладнання являє собою такі пристрої:

— електронний турнікет, який забезпечує санкціонований доступ до спортивного майданчику;

— RFID зчитувач, який отримує інформацію з пластикової картки (з RFID-міткою), брелка, телефону (який підтримує технологію NFC);

— Контролер, який оброблює інформацію з RFID зчитувача і вирішує чи має доступ ,певна людина в певний час, до спортивного майданчику;

Контролер містить в собі програмне забезпечення і пам'ять. Для надсилання дистанційних команд, використовується технологія передачі даних LTE. Для роботи з контролером необхідно встановити програмне забезпечення STOP.NET, яке постачається на диску.

Ілюстрація взаємодії компонентів сервісу, починаючи від користувача майданчика закінчуючи спорт-майданчиком, можна знайти на рисунку 1.3.

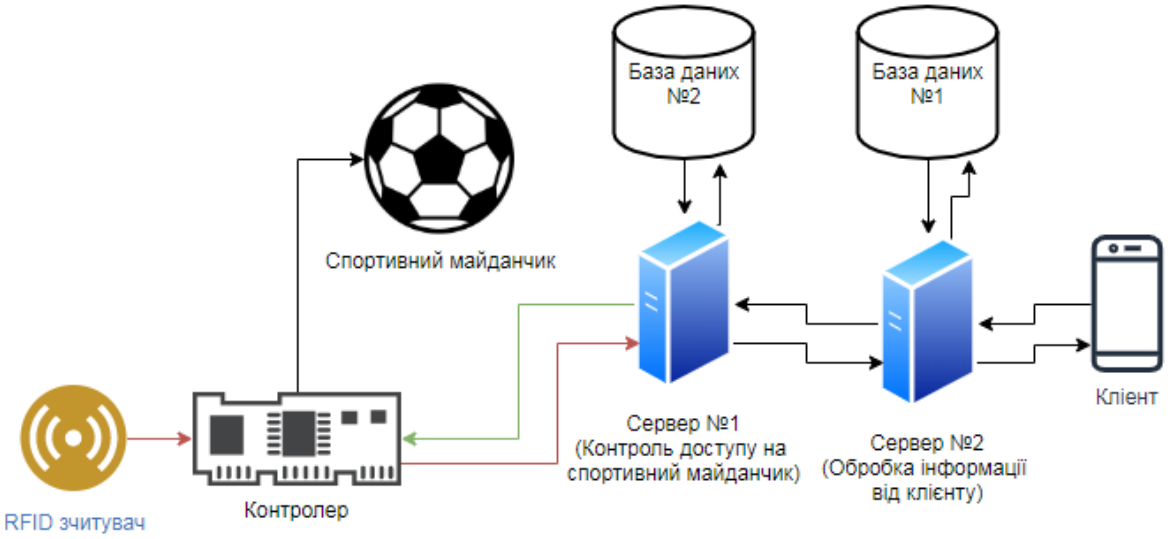

Рисунок 1.4 – Ілюстрація роботи системи

Розглянемо ланцюг дій, в момент створення події:

1. Користувач реєструє нову подію в сервісі.
2. Сервер №2 оброблює запит від клієнту на реєстрацію події і передає інформацію на сервер №1.
3. Сервер №1 приймає запит і зберігає необхідну інформацію. Тим самим створює правило, яке дозволить користувачу потрапити на цю подію в заданий час.
4.

Розглянемо ланцюг дій, коли користувач приєднується до події:

1. Користувач надсилає запит на сервер № 2 про приєднання до події
2. Сервер №2 оброблює і зберігає інформацію. Після чого передає цю інформацію далі на сервер №1.
3. Сервер №1 фіксує інформацію і створює правило, яке дасть можливість користувачу потрапити на подію.

Пропоную розглянути ланцюг дій, в момент рішення контролеру про доступ користувача до спортивного майданчику (рисунок 1.5) :

1. Користувач підносить пластикову картку до зчитувача

2. Контролер оброблює дані картки і відсилає на сервер №1
3. Сервер №1 оброблю дані і веде пошук на наявні правила пропуску користувача в базі даних.
4. В разі успішного пошуку, сервер відправляє команду відчинення турнікету на контролер.

Для того, щоб скористатися даним сервісом бронювання, необхідно зареєструватися в мобільному додатку та пройти процедуру аутентифікації в офісі, який обслуговує сітку спортивних майданчиків.

Ця процедура являє собою внесення в базу даних інформацію про користувача і видання картки-перепустки (RFID мітка).

Авторизація необхідна для:

— запобігання створення великої кількості спам-акаунтів, які би бронювали всі вільні місця;

— інформації про користувачів;

— повноцінного доступу користувачі до спорт-майданчику;

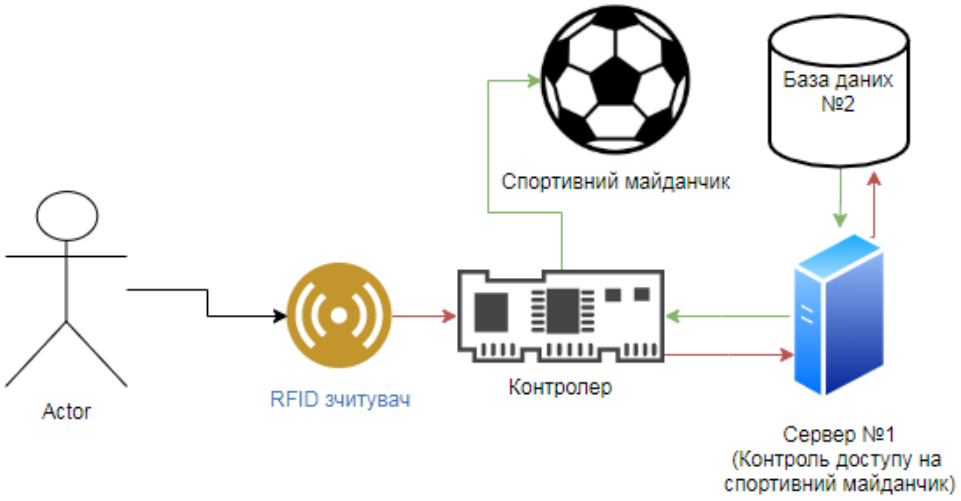

Рисунок 1.5 – Схема доступу користувача до спортивного майданчику

Серверна частина була спроектована таким чином, яка дає можливість легко створювати нові спортивні майданчики та встановлювати особливі правила проведення спортивних подій. Також серверна частина передбачає інтеграцію з різними видами клієнтських та серверних додатків.

Авторизований користувач має змогу:

— Створювати нові події

— Переглядати майбутні події

— Переглядати архів власних подій

— Приєднуватися до майбутньої події

— Покинути подію, до якої приєднався раніше

— Переглядати інформацію про доступні спортивні майданчики

Створені події зберігаються в базі даних, яка знаходиться на сервері. Доступ до редагування та вилучення події має лише адміністратор сервісу і користувач-творець[3].

Приєднуватися до подій можуть тільки авторизовані користувачі, в інакшому випадку необхідно отримати або відновити авторизацію.

Не авторизовані користувачі, можуть переглянути доступні події та отримати інформацію щодо подальшої авторизації.

До кожного користувача накладаються певні обмеження в створенні подій в день. Тобто, один користувач не має змогу забронювати весь день вільного часу на спортивному майданчику без узгодження адміністрації. Також для обслуговування спортивного майданчику, адміністрація може заборонити бронювання на деякий час.

Висновки до розділу

В даному розділі було розглянуто основні види бронювання та описані відмінності між ними. Окрім цього було описано вид бронювання спортивного майданчику, який використовується в розроблюваному сервісі. Описані в розділі підходи було застосовано у серверному додатку.

2. ЗАСОБИ РЕАЛІЗАЦІЇ ПРОГРАМНОЇ СИСТЕМИ

Проаналізувавши дану задачу та під час пошуку методів її вирішення, було прийнято рішення розроблювати серверну частину на основі веб-технологій. Головною перевагою цього методу перед іншими варіантами є його універсальність і можливість розробки клієнтської частини на багатьох мовах програмування і під різні пристрої.

Основним середовищем розробки був програмний пакет JetBrains WebStorm.

Для розробки серверної частини використовувалась мова програмування Node.js та фреймворк Express, який дозволяє оброблювати запити через протокол HTTP(S).

Для збереження даних про користувача, події, та спортмайданчики було прийнято рішення використовувати програмне забезпечення PosgreSQL.

2.1 Середовище JetBrains WebStorm

Середовище розробки WebStorm є надійним інструментом для розробки для JavaScript, HTML, CSS . Тому воно має всі необхідні інструменти для розробки і тестування додатків створених в цьому середовищі. Інструменти даного середовища вміло і швидко доповнюють строки коду, що пише програміст, та допомагає дотримуватися «гарного коду» . Крім того у WebStorm підтримує встановлення додатків, які допомагають налаштувати середовище під конкретну задачу, що дає можливість не використовувати стороннє програмне забезпечення під час розробки.

Використовувати середовище розробки WebStorm може кожен, адже на головному сайті JetBrains є безкоштовна пробна версія WebStorm, яка дасть можливість відкрити всі функції програмного забезпечення і відчути переваги над іншими середовищами розробки схожих на WebStorm.

Редактор пропонує різні функції:

— Редактор JavaScript
    — Розумне автозаповнення
    — Перевірка коду та швидке виправлення
    — Швидка кодова навігація та пошук використання коду
    — Рефакторинг коду
— Відладчик і тести
    — Відладчик JavaScript
    — Тест блоку JavaScript
— Підтримка Node.js
— Підтримка мов: TypeScript, CoffeeScript, Dart і ECMAScript
— Перевірка орфографії
— Повторюваний детектор коду
— HTML і CSS
    — Підтримка мов HTML5, CSS3, LESS та SASS
    — Валідація коду
    — Перегляд швидкого вмісту
    — Синхронізація файлів FTP
    — Інтеграція версій: Subversion, Git, Mercurial, Perforce, CVS

WebStorm[4], як і інші IDE на платформі IntelliJ IDEA, робить розробку простіше і зручніше. WebStorm забезпечуючи підсвічування і автодоповнення коду, перевіряє його на помилки, допомагає швидко переміщатися по проекту, безпечно вносити зміни за допомогою рефакторингів. У WebStorm є інструменти для налагодження коду і інтеграція з системами управління версіями.

WebStorm по-справжньому розуміє структуру вашого проекту і код, виявляє можливі проблеми ще до того, як ви відкрили проект в браузері, і пропонує їх рішення.

Вбудовані в середовище розробки інструменти для відладки коду зроблять для вас значну кількість роботи і зроблять її зручніше і швидше. На момент

написання серверної частини було використано версію WebStorm 2017.2.1, яка має такі особливості для розробки серверної частини сервісу:

— Додано можливість допомоги з налаштуванням додатку Express, який виступає в ролі серверу для оброки HTTP(S) запитів. Його налаштування є дуже важливим. Тому JetBrains звернули на це увагу і встановили новий модуль;

— З новою версією з’явилась можливість автоматичної заміни асинхронних функцій, яка перетворює Promise на функції з префіксом async, які перетворюються з асинхронних в синхронні . Для цього необхідно виділити тіло функції і використати комбінацію Alt-Enter. Ця операція можлива в усіх файлах, які підтримують мову програмування JavaScript;

— Тепер стала доступна функція моніторингу пакетів для мови NodeJs. Це дає зручний інтерфейс пошуку та встановлення нових необхідних пакетів для розробки серверної частини;

— З новим оновленням долучилась нова можливість рефакторингу мови стандарту EcmaScript 6. Цей стандарт є набагато зручнішим в порівнянні зі стандартом EcmaScript 5.1;

— За допомогою нових змінних в модулі індексації файлів, пошук став набагато швидше і це дає можливість зекономити час на пошуку необхідного коду в цілому проекті.

2.1.1 Якість коду

Важливою частиною розробки серверного додатку є не тільки написати певний функціонал, а і зробити це правильним чином та забезпечити те, щоб розроблений сервіс однаково якісно працював на всіх операційних системах, які підтримують мову програмування NodeJs, з цією задачею гарно допомагає таке доповнення, як ESLint. Дане доповнення завжди зверне увагу програміста на

недоліки, які би могли спровокувати несумісності при переведенні на іншу операційну систему.

2.1.2 Автоматичне доповнення синтаксису

WebStorm постійно обновлює і підтримує всі нові та старі технології, фраємворки для мови JavaScript і EcmaScript, а тому кожен розробник отримує інструменти доповнення коду для багатьох фраємворків[5]. Також WebStorm дає можливість переглянути необхідні аргументи для кожної наявної функції. Це значно скорочує час розробки, адже розробник не витрачає час на постійне переміщення між файлами проекту. Крім цього варто відмітити що середовище розробки дає можливість повного налаштування інтерфейсу для зручної роботи.

2.1.3 Командний рядок

Всі розробники, які використовують WebStorm отримують ще більш зручну та багатофункціональну команду строку. За допомогою її можна отримати доступ до системних команд, також запустити інтерпретатор NodeJs, який в свою чергу запустить головний скрипт.

Під час виконання скрипту, можуть виникнути помилки в коді, ці помилки відразу будуть надіслані до консолі, в якій будуть підсвічені посилання на рядки коду, де є помилка. Це є дуже зручним інструментом, який економить час на пошук необхідного рядка коду.

Поміж цього, команда строка пропонує і доповнює можливі закінчення команди. Так при додаванні скриптів до файлу package.json, який містить в собі налаштування проекту, WebStorm надає пропозиції щодо команд, які доступні в налаштуваннях проекту.

2.1.4 Налагодження та усунення помилок

Webstorm постійно слідкує за кодом, який пише розробник і підсвічує червоною лінією ділянки коду, які не пропустить інтерпретатор NodeJs. Також середовище розробки має дуже зручну систему навігації помилок в файлі коду.

Справа біля SB є стрічка, яка має червоні різнокольорові риски. Якщо в коді було допущено помилку, або вона була знайдена в момент виконання скрипту, на ній з'явиться червона риска, яка буде сигналізувати помилку в коді. Якщо натиснути на неї, це дасть можливість швидко перейти на необхідну ділянку коду, для її виправлення.

Також файли, в яких було знайдено помилки будуть також підсвічуватися, як на рис 3.1.

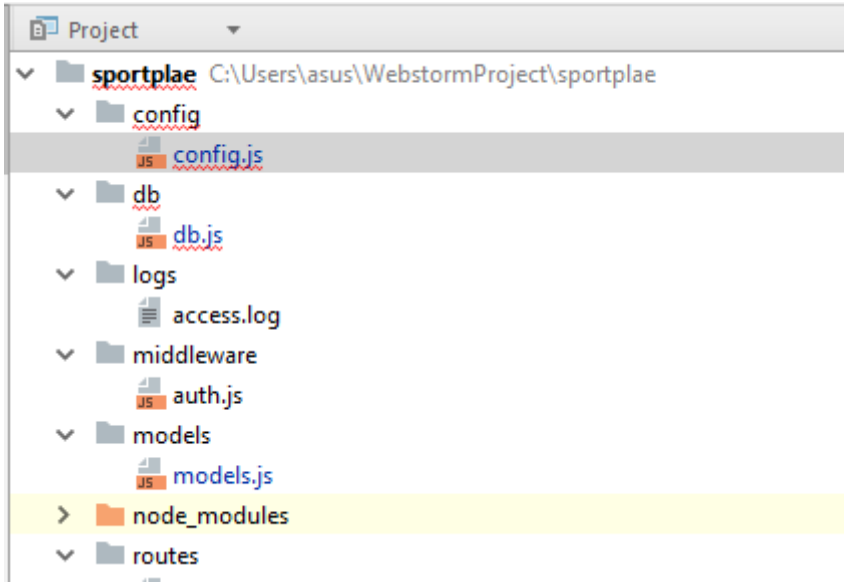

Рисунок 3.1 – Ппідсвітка файлів, які містять помилки

2.1.5 Багатовіконний режим редагування

Важливою функцією є багатовіконний режим роботи з файлами. Це дає можливість розмістити на одному экрані декілька файлів. Цей інструмент є дуже зручним, коли розробник постійно переключается між файлами і витрачає на це час.

2.2 Вибір архітектури програмного комплексу

Поставлена задача потребує правильної архітектури для подальшого розвинення сервісу. Тому було прийнято рішення використовувати архітектуру, яка складається з трьох ланок:

— Клієнтська частина

— Серверна частина

— База даних

Ця структура являє собою найпопулярніше рішення для подібних сервісі. Це дає можливість розширити систему в майбутньому, яка би складалась з декількох серверів, які обслуговували клієнтів[6]. Ця система зображена на рисунку 2.2 [9].

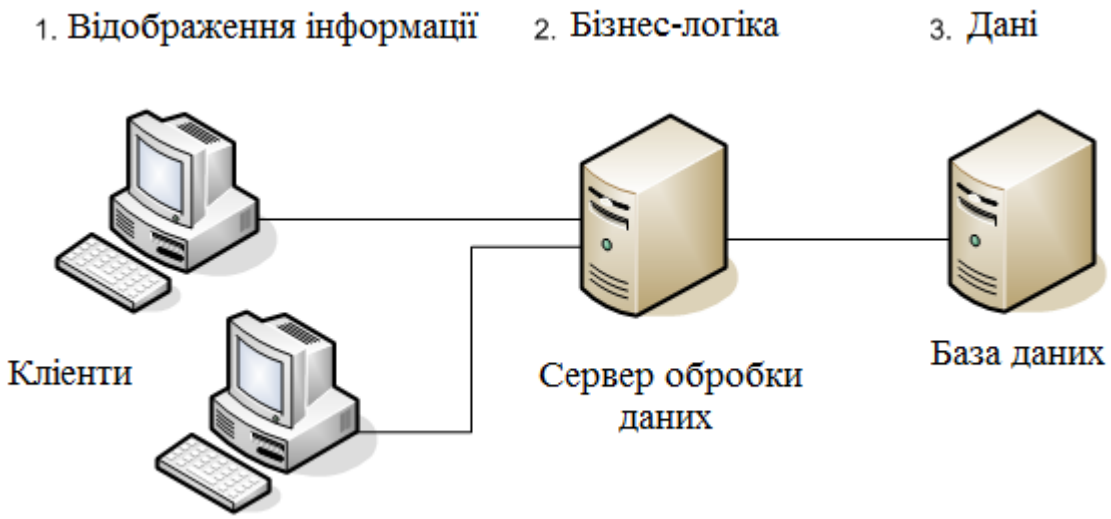

Рисунок 2.2 – Триланкова архітектура програмного комплексу

Багаторівневе програмування - це модель розробки програмного забезпечення, в якій основною метою є поділ (роз'єднання) частин, що складають програмну систему, або ж клієнт-серверну архітектуру: бізнес-логіку, клієнтський додаток і базу даних. Таким чином, для створення різних інтерфейсів в одній системі, не треба робити ніяких змін в даних.

Головною перевагою цього стилю є те, що розробка може здійснюватися на декількох рівнях, і в разі будь-яких змін, це вплине тільки на необхідний рівень без необхідності перевірки серед вихідних кодів інших модулів, оскільки він буде скорочений.

Рівень відображення інформації: той, що бачить користувач (також званий "рівень користувача"), представляє собою систему для користувачів, які передають інформацію. Цей рівень обробляє інформацію користувача (виконує попередню фільтрацію, щоб перевірити, що помилок при відправлені даних немає). Він також відомий як графічний інтерфейс користувача і повинен мати властивість бути "дружнім" (зрозумілим і простим у використанні) для користувача. Цей рівень спілкується тільки з бізнес-рівнем.

Бізнес-рівень: де знаходяться програми, отримуються запити користувача і відповіді надсилаються після процесу обробки цих даних. Це називається бізнес-рівень (і навіть бізнес-логіка), тому що тут встановлюються всі правила, які повинні бути виконані. Цей рівень зв'язується з рівнем відображення, приймає запити і представляє результати, а з рівнем даних – запитує необхідну інформацію з баз даних, або відправляє запит на збереження цих даних. Прикладні програми також розглядаються тут.

Рівень даних- це рівень який зберігає дані і відповідає за їх доступ. Він формується одним або декількома базами даних, які відповідають за зберігання даних, приймають запити на зберігання або отримання інформації з бізнес-рівня.

Всі ці шари можуть знаходитися в одному комп'ютері, хоча найпоширенішим є те, що існує безліч комп'ютерів, на яких знаходиться рівень відображення даних (вони є клієнтами архітектури клієнт / сервер). Рівень бізнес-логіки може перебувати на одному комп'ютері, проте при зростанні потреб, вони можуть бути розділені на два або більше комп'ютерів. Таким чином, якщо розмір або складність бази даних зростає, вона може бути розділена на кілька комп'ютерів, які будуть отримувати запити від комп'ютера, в якому знаходиться бізнес рівень.

Крім того, це дозволяє розподілити завдання створення програми за рівнями. Таким чином, кожна робоча група повністю абстрагується від інших рівнів, так що достатньо знати API, який існує між рівнями [7].

Якщо, навпаки, складність у бізнес-рівні вимагала поділу, то цей рівень міг би знаходитися в одному або декількох комп'ютерах, які надсилали б запити до однієї бази даних. У дуже складних системах можна мати серію комп'ютерів, на яких працює бізнес-рівень, і ще одну серію комп'ютерів, на яких працює база даних.

Термін "рівень" відноситься до способу сегментації рішення з логічної точки зору:

— презентація (Відомий як веб-рівень у веб-додатках або як рівень користувача у програмах);
— бізнес-логіка (Відомий як рівень програми);
— дані (Відомий як рівень бази даних);

Розглянемо основні переваги трирівневої архітектури:

— незалежність рівнів один від одного;
— швидке масштабування системи ;
— висока безпека і надійність;
— висока надійність;
— мінімальне навантаження на клієнтську частину

Також пропоную розглянути недоліки такої системи:

— багаторівнева система потребує більше ресурсів для розробників;
— потреба в більшій пропускної спроможності;
— потреба в більшій кількості серверного обладнання;

Перш за все, ця концепція використовується для зменшення навантаження користувача. Таким чином кожен рівень відповідає за свої обов’язки. Це дає

змогу зробити більш зручний і зрозумілий програмний код для подальшої підтримки і розширення функціоналу.

2.2 Загальна характеристика архітектури серверного додатку

При проектуванні серверної частини було прийняте рішення використовувати шаблон проектування MVC (Model View Controller).

Шаблон проектування MVC дає об'єктам одну з трьох ролей: модель (model), уявлення (view) або контролер (controller). Цей шаблон не тільки розподіляє ролі у об'єктів, але так само визначає те, як вони будуть взаємодіяти один з одним. Кожен з трьох типів об'єктів відділений від інших абстрактними законами (якщо можна це так назвати) і зв'язується з іншими об'єктами за допомогою цих законів.

MVC займає гарне місце в проектуванні додатків. У цього шаблону багато переваг. Наприклад, багато об'єктів в додатку можна використовувати багато разів і вони незалежні від інших типів. Додатки MVC краще і простіше розширювати. Багато технологій створені з використанням MVC.

Пропоную розглянути всі ланки даного шаблону проектування:

— модель (модель) - це подання інформації або даних, які потрібно відобразити користувачеві. Модель має класи, а в свою чергу класи визначають різні частини вашого сайту. Наприклад, якщо у вас є веб-сайт електронної комерції, то в ньому існують різні аспекти, такі як клієнти, продукти, категорії, замовлення. Класи будуть створені для кожного з цих розділів вашого веб-сайту. Клас клієнта буде мати всі властивості клієнта, такі як ім'я, прізвище, адреса, контактна інформація. Клас замовлень буде мати всі властивості замовлення, такі як дата його створення та клієнт, який розмістив замовлення. Клас продукції буде мати всі властивості продукту. Модель отримує всі ці дані з бази даних і дані можуть бути прочитані, оновлені і видалені;

— презентація (View) - об'єкти презентації це такі об'єкти, які користувач бачить на екрані. Ці об'єкти знають як себе показувати і можуть реагувати на дії користувача. Основна мета об'єктів предствлення - відображати дані з об'єктів моделі програми та дозволяти редагування цих даних. Незважаючи на це, об'єкти view зазвичай відділяються від об'єктів model в додатку MVC..

— Контролер (Controller) - об'єкт контролера виконує роль посередника між одним або декількома view об'єктами і об'єктами model. Об'єкти контролера, таким чином, є каналом, через який view об'єкти дізнаються про зміни в об'єктах моделі і навпаки. Об'єкти контролера також можуть виконувати настройку і координування завдань для програми та керувати життєвими циклами інших об'єктів.

Зв'язок об'єктів відбувається таким чином: Об'єкт контролера інтерпретує дії користувача, створені у view об'єктах, і передає нові або змінені дані на модельний рівень. Коли об'єкти моделі змінюються, об'єкт контролера пов'язує ці нові дані моделі з об'єктами уявлення, щоб вони могли його відображати. Ця суть проілюстрована на рисунку 2.3 [10].

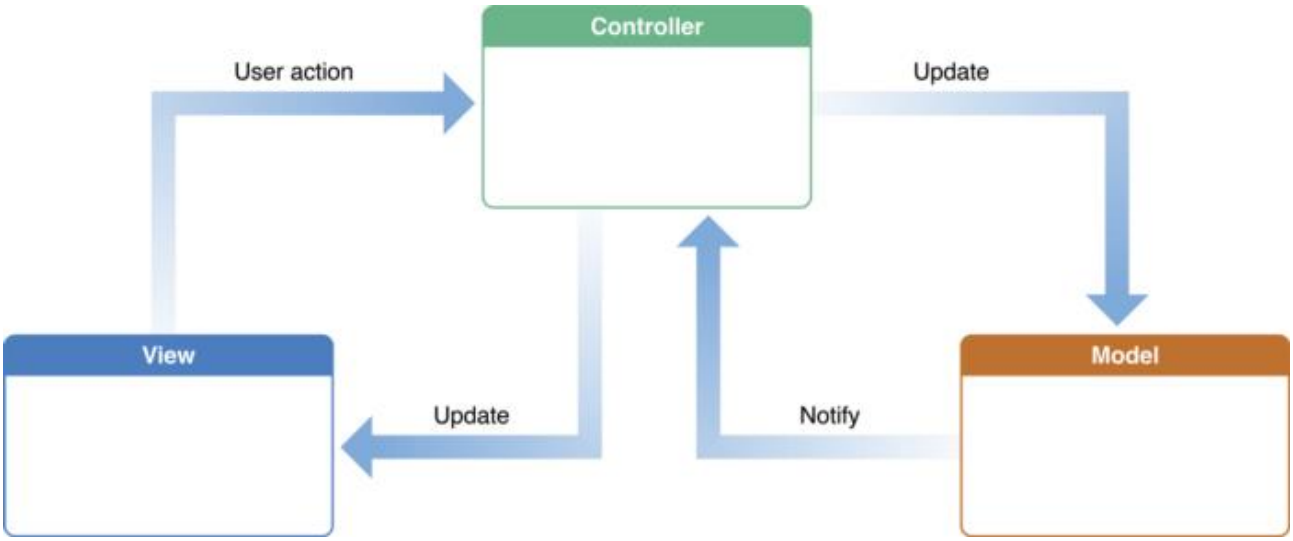

Рисунок 2.3 Шаблн проектування MVC - Model View Controller

Пропоную розглянути кожен компонент MVC, як його було використано у дипломному проекті.

Контролер – це сукупність коду, який відповідає за доступ до даних, тобто до бази даних. Цей код контролює вхідні та вихідні запити через протокол HTTP(S). Також він обслуговує запити до бази даних та маршрутірізацію. Ще одною задачею контролеру – є перевірка цілісності даних та відповідності до бізнес-моделі.

Модель – перевіряє чи відповідають передані від контролеру дані необхідними характеристиками для їх подальшого збереження. Також варто відмітити, що саме підключення та взаємодія з базою даних та іншими сховищами виконується також на рівні моделі додатку. Останньою задачею є підготовка даних до візуалізації даних та передачі їх на рівень представлення.

Представлення – представляє собою мобільний додаток на базі Android. Який напряму спілкується з контроллером через HTTP(S) запити. Це дає можливість змінювати клієнт і не змінювати контролер.

2.3 Опис архітектури серверного додатку

Node або Node.js - програмна платформа, яка працює на рушію V8 (тобто транслює JavaScript в машинний код процесору), що перетворює JavaScript з високорівневої мови в мову загального призначення (машинний код). Node.js додає можливість з допомогою JavaScript взаємодіяти з пристроями введення-виведення через свій API (написаний на мові C ++), підключати інші зовнішні бібліотеки, написані на різних мовах, забезпечуючи виклики до них з JavaScript-коду. Маючи більше мільярда завантажень, Node.js процвітає у створенні програм у реальному часі, інтернет-мереж та мікросервісах[7].

Розглянемо найголовніші переваги рушію V8:

— Компіляція вихідного коду JavaScript безпосередньо в власний машинний код, минаючи стадію проміжного байт-коду. Це дає можливість пришвидшити запуск і обробку коду.

— Ефективна система управління пам'яттю, яка веде до швидкого об'єктному виділення і маленьким паузам збірки «сміття».
— V8 зупиняє виконання коду під час виконання збірки «сміття».
— Зменшує вплив на додаток при застосування складанні «сміття».
— V8 може точно визначати, де знаходяться в пам'яті об'єкти і покажчики, що дозволяє уникнути витоку пам'яті при помилкової ідентифікації об'єктів в якості покажчиків.

Іншими словами: Node.js пропонує вам можливість писати неймовірно продуктивний серверний код з використанням JavaScript. Як говориться в офіційному описі: Node.js - це середовище виконання, що використовує той же JavaScript-движок V8, який ви можете знайти в браузері Google Chrome. Але цього недостатньо для успіху Node.js. У Node.js використовується libuv - крос-платформна бібліотека підтримки з акцентом на асинхронний ввід-висновок.

З точки зору розробника, Node.js однопоточний, але під капотом libuv використовує потоки, події файлової системи, реалізує цикл подій, включає в себе тред-пулінг і так далі. У більшості випадків ви не будете взаємодіяти з libuv безпосередньо.

Це означає, що Node.js дасть вам можливість не задумуватися про операції з оперативною пам’яттю, про видалення «сміття» і про контроль потоків.

Завдяки тому, що програмна платформа Node.js підтримує операції вводу і виводу вона може оброблювати запити, які приходять на системні порти в операційній системі[8]

Для створення повноцінного серверу використовувався бібліотека Express Js. Ця бібліотека написана на JavaScript. Express бере на себе обробку запитів на мережевому рівні.

2.4 Опис інструментів розробки

Для розробки серверної частини було використані такі інструменти:

— Мова програмування JavaScript

— Інтерпретатор мови, NodeJs
— Бібліотека Expres
— Бібліотека Sequelize
— Реляційна база даних PostgreSQL
— Програмний засіб PgAdmin 3 LTS
— Программа Postman

Пропоную детальніше розглянути основні інструменти, за допомогою яких було реалізовано серверну частину.

Бібліотека Express - це зручний, швидкий і надійний інструмент для створення веб-сервісів. Він дає можливість оброблювати HTTP(S) запити. Також він має зручний інтерфейс для маршрутизації запитів, перехоплення запитів і для відправлення відповіді клієнту.

Що стосується додатку Sequelize, то ця бібліотека дає можливість спілкування з базою даних на мові ORM, яка є біль зрозумілішою перед класичною мовою SQL.

Деякі переваги включають в себе те, що ви можете писати запити БД через простий JS, і він абстрагує деякі складності для перевірки даних і підключень БД. Крім того, велика перевага використання ORM, такого як sequelize, полягає в тому, що ви зможете переходити до іншої системи баз даних без необхідності переписувати всю логіку запитів БД.

ORM є процесом зіставлення між об'єктами і системами бази даних відносин. ORM діє як інтерфейс між двома системами. ORM надає розробникам переваги від базових, як, наприклад, економія часу і зусиль, а зосередження уваги на бізнес-логіці. Код є надійним, а не надлишковим. ORM допомагає в ефективному управлінні запитами для декількох таблиць. Нарешті, ORM (наприклад, sequelize) здатний з'єднуватися з різними базами даних (що дуже зручно при переході з однієї бази на іншу).

Sequelize - це ORM для Node.js. Sequelize легко вчитися і має десятки цікавих функцій, таких як синхронізація, асоціація, перевірка і т.д. Він також має підтримку PostgreSQL, MySQL, MariaDB, SQLite і MSSQL.

Для розробки і тестування серверної частини я використовував програмне забезпечення Postman. Воно дає можливість створювати і надсилати HTTP запити. Проте головною особливістю є те, що в данному продукті є можливість створювати проект, котрий буде містити в собі повну документацію для API. Це дає змогу запрограмувати всі запити на сервер і проводити тести на відповідність даних, які оброблює сервіс. На рисунку 2.4 зображений інтерфейс програми.

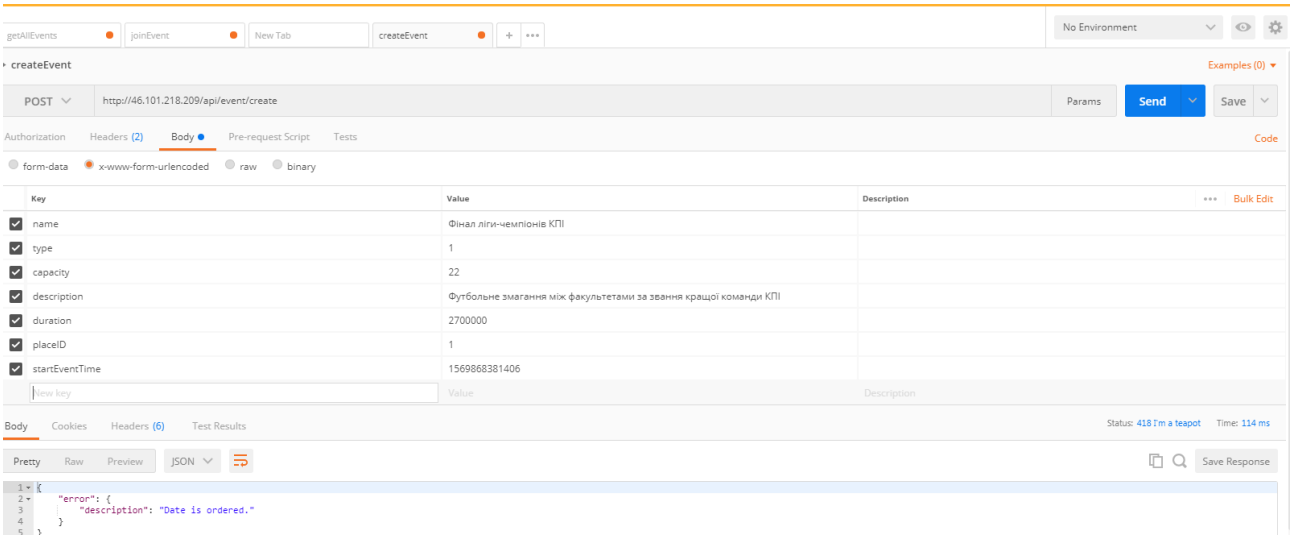

Рисунок 2.4 – Інтерфейс програми Postman

Стосовно відладки роботи з базою даних, я використовував програмний засіб PgAdmin. Ця програма є найпопулярнішою і багатофункціональною платформою з відкритим кодом і розробкою для PostgreSQL, найдосконалішою базою даних Open Source у світі.

З допомогою PgAdmin можно швидко переміщатися між базами даних, робити нові записи в базі даних, проектувати запити, виконувати пошук даних та багато іншого. Це значно прискорює роботу розробника і дає можливість візуалізації даних в базі. Інтерфейс програми зображений на рисунку 3.4.

Тому використання різних компонентів в сукупності дає значно кращу продуктивність і пришвидшує розробку.

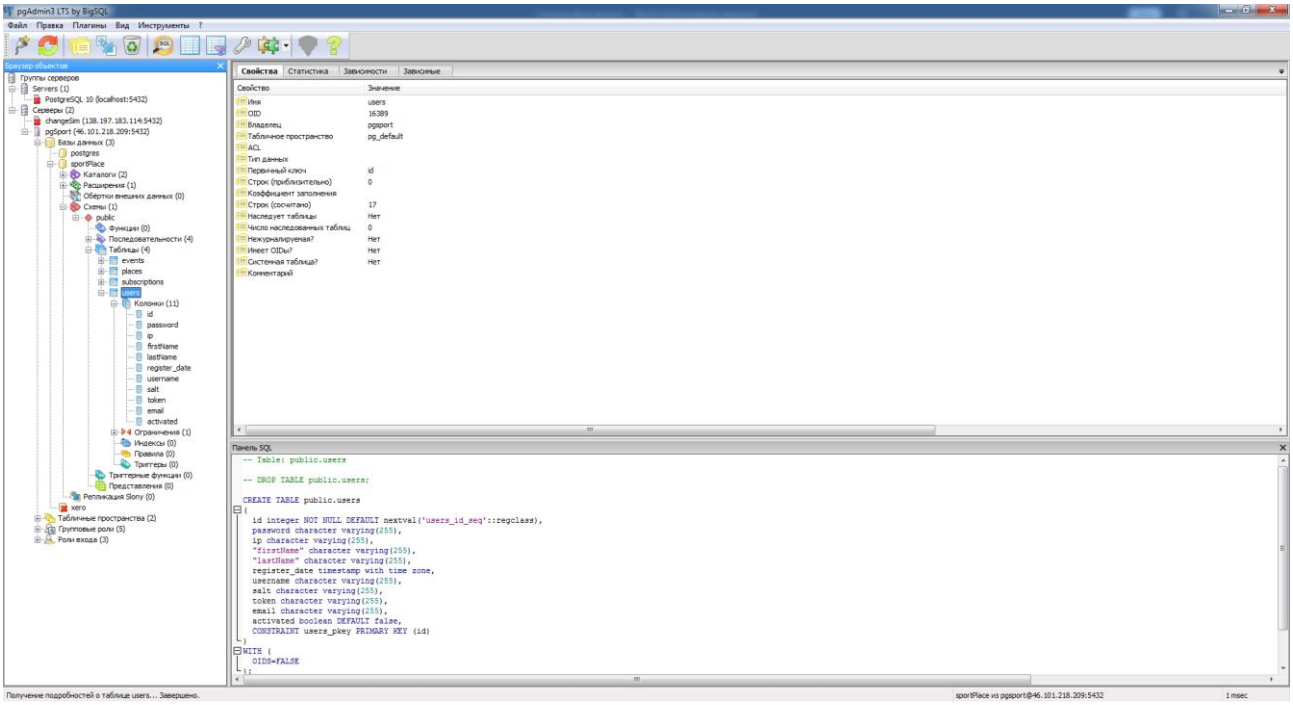

Рисунок 2.4 – Інтерфейс програми для роботи з базою даних PgAdmin

2.4.1 Проектування бази даних

Під час аналізу поставленої задачі для розробки серверної частини, було вирішено використовувати систему бази даних, яка би дозволяла швидко оброблювати запити, мала високу відмовостійкість, дозволяла би автоматично підтримувати саму себе і правильно індексувати поля в таблицях. Тому було вибрано систему баз даних PostgreSQL.

PostgreSQL, також відомий як Postgres, є вільною і відкритою системою управління реляційними базами даних. Він призначений для роботи з різними навантаженнями, від окремих машин до сховищ даних або веб-служб з багатьма одночасними користувачами[9]

PostgreSQL має транзакції з властивостями атомності, консистенції, ізоляції, довговічності, автоматично оновлюваних переглядів, матеріалізованих видів, тригерів, зовнішніх ключів і збережених процедур.

Під час проектування бази даних було запропоновано наступну архітектуру (рисунок 2.4) бази даних, яка би дозволяла швидко масштабуватись і мати надійну структуру і легкість виконання запитів (рисунок 2.5).

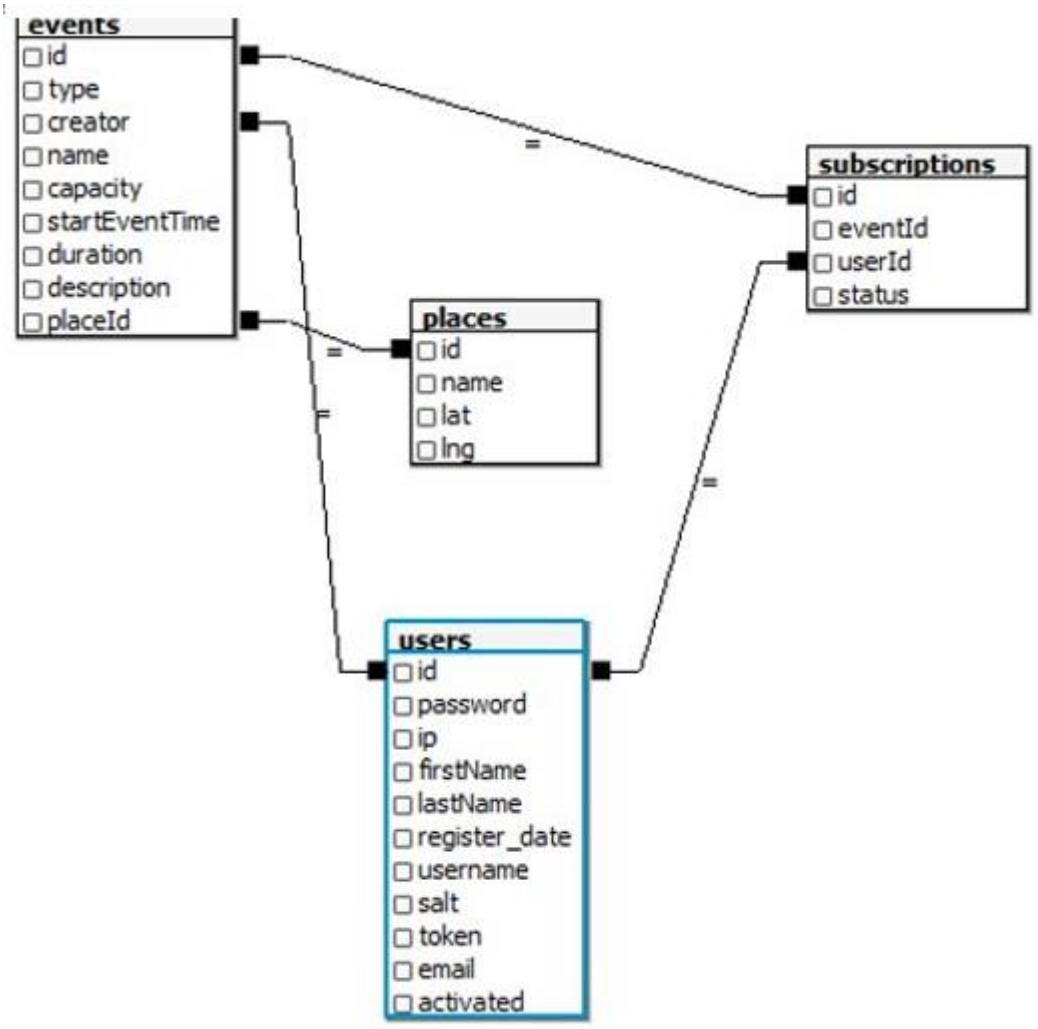

Рисунок 2.4 – Візуальне відображення архітектури бази даних.

Розглянемо основні сутності даної архітектури бази даних.

Events (з англ. Події) - таблиця, яка містить в собі інформацію про події, які створювали користувачі. Кожний запис в цій таблиці містить інформацію про тип події, творця події, назву, інформацію про місце и час події.

Users (з англ. Користувачі) - таблиця, яка містить в собі інформацію про користувачів. Кожний запис в цій таблиці містить інформацію про ім’я, фамілію, секретний пароль та прапорця авторизації користувача.

Places (з англ. Місця) - таблиця, яка містить в собі інформацію про спортивні майданчики. Кожний запис в цій таблиці містить інформацію про координати спортивного майданчику і його ім’я.

Subscriptions (з англ. Підписки) - таблиця, яка містить в собі інформацію про підписки. Кожний запис в цій таблиці містить інформацію про підписку на подію, тобто коли користувач хоче приєднатися до події, створюється запис в

Підводячи підсумки, варто відмітити, що PostgreSQL - це база даних загального призначення, яка використовується для створення сервісів з великим навантаженням. Найбільш поширеними випадками використання PostgreSQL є малі та глобальні проекти.

```
CREATE TABLE public.events
(
    id integer NOT NULL DEFAULT nextval('events_id_seq'::regclass),
    type integer,
    creator integer,
    name character varying(255),
    capacity integer,
    "startEventTime" timestamp with time zone,
    duration bigint,
    description character varying(255),
    "placeId" integer,
    CONSTRAINT events_pkey PRIMARY KEY (id),
    CONSTRAINT "placeId_foreign_idx" FOREIGN KEY ("placeId")
        REFERENCES public.places (id) MATCH SIMPLE
        ON UPDATE CASCADE ON DELETE SET NULL
)
WITH (
    OIDS=FALSE
);
ALTER TABLE public.events
    OWNER TO pgsport;
```

Рисунок 2.5 – Приклад створення таблиці.

Висновки до розділу

Для розробки сервісу бронювання спортивний майданчиків було прийнято рішення використовувати набір додатків мови програмування JavaScript, таких

Node.js (для серверного додатку) та базу даних PostgreSQL. Такий набір технологій, разом з шаблоном проектування MVC, дозволяє в майбутньому легко масштабувати систему і робити її безпечною та захищеною від зловмисників.

3. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Розроблений мною дипломний проект має таку структуру:

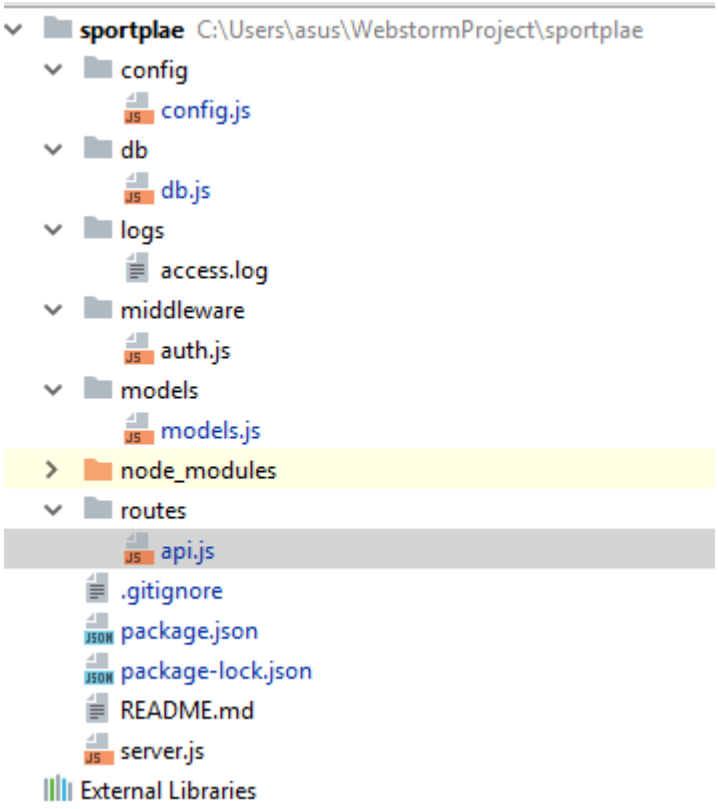

Рисунок 3.1 – Ієрархія папок проєкту.

Складовими частинами серверного додатку є піддиректорії config, db, models, routes,middleware,logs.

Директорія config – містить в собі конфігураційний файл. Він містить в собі об’єкт конфігурації для з’єднання з базою даних. Його необхідно редагувати лише в тому випадку, якщо дані для з’єднання з базою даних змінилися або створилися при інсталяції бази даних. Файл містить в собі розділ «dev» і розділ «prod». Перший розділ використовується з тестовою базою даних, а другий розділ, використовується з справжньою базою даних (рисунок 4.2).

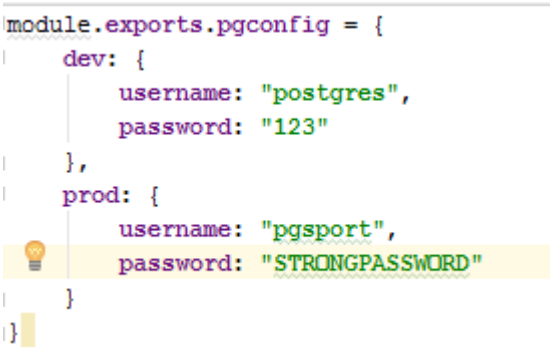

Рисунок 3.2 – Конфігураційний фал для з’єднання з базою даних.

З компонентів, які знаходяться саме у папці pages виконуються запити на сервер, і саме ці компоненти передають дані на компоненти «без логіки», які знаходяться у папці components, та служать у більшості контентом для компонентів вищого порядку, описаних у pages.

— Папка routes – відповідає за маршрутизацію запитів і містить бізнес-логіку всього сервісу;
— Папка models – відповідає за налаштвання моделей в базі даних;
— Папка middleware – містить в собі перехоплювач запитів, який необхідних для правильної аутентифікації користувача;

3.1 Опис функціональності системи

Серверний додаток який призначений для отримання інформації від клієнту, оброку, зберігання і видачу інформації був розроблений і спроектований під час виконання дипломної роботи. За допомогою цього додатку, користувачі зможуть створювати події та приєднуватися до них. Також сервер буде реєструвати користувача на пропускному пункті спортивного майданчику, для його пропуску на майданчик.

3.2 Опис таблиць бази даних

База даних сервісу складається з чотирьох таблиць, які створюють надійну систему для зберігання, редагування та отримання інформації. Для побудови запитів та створення міграцій, було використано додаток для NodeJS – Sequelize. Він автоматично, при запуску, вносить зміни в таблицю, якщо це необхідно – після чого робить її валідацію.

«Events» - таблиця, яка містить в собі:

— унікальні ідентифікатор для кожної події, який генеруються за допомого автоінкрементування;

— тип події, який буде проводитися на спортивному майданчику;

— ідентифікатор користувача, який створив подію;

— назва події;

— максимальна кількість людей для події;

— дата початку події;

— тривалість;

— опис;

Таблиця має таку структуру:

```
25  const EventModel = sequelize.define('event', {
26      id: {type: Sequelize.INTEGER, primaryKey: true, autoIncrement: true},
27      type: Sequelize.INTEGER,
28      creator: Sequelize.INTEGER,
29      name: Sequelize.STRING,
30      capacity: Sequelize.INTEGER,
31      startEventTime: {
32          type: Sequelize.DATE,
33          get () {
34              return new Date(this.getDataValue('startEventTime')).getTime();
35          }
36      },
37      duration: Sequelize.BIGINT,
38      description: Sequelize.STRING
39  }, {
40      tableName: 'events',
41      freezeTableName: false,
42      timestamp: false
43  });
```

Рисунок 4.3 – Створення таблиці «events» за допомогою Sequelize

Наступна таблиця – таблиця користувачів – «users», яка зберігає інформацію про кожного користувача і має наступну структуру (рисунок 4.4)

```
const UserModel = sequelize.define('user', {
    id: {type: Sequelize.INTEGER, primaryKey: true, autoIncrement: true},
    password: Sequelize.STRING,
    ip: Sequelize.STRING,
    firstName: Sequelize.STRING,
    lastName: Sequelize.STRING,
    register_date: {"type": Sequelize.DATE...},
    username: Sequelize.STRING,
    salt: Sequelize.STRING,
    token: Sequelize.STRING,
    email: Sequelize.STRING,
    activated: {type: Sequelize.BOOLEAN, defaultValue: false}

}, {
    tableName: 'users',
    freezeTableName: false,
    timestamp: false
});
```

Рисунок 3.4 – структура таблиці users

Розглянемо важливі впровадження в даній таблиці:

— поле «activated» говорить нам про інформацію щодо авторизації користувача і можливість створювати чи приєднуватися до подій

— поле «token» містить в собі ключ, за допомогою якого можлива ідентифікація користувача при обробці запитів в серверному додатку

— поле password містить в собі hash-суму MD5 пароля і згенерованої солі. Це значить, що в базі даних не зберігаються особисті паролі. Також використання солі дає більше впевненості в стійкості перебору пароя

Третя таблиця – таблиця про підписки – «subscriptions», яка зберігає інформацію про кожну підписку на подію (рисунок 3.5)

```
const EventSubscription = sequelize.define('subscription', {
    id: {type: Sequelize.INTEGER, primaryKey: true, autoIncrement: true},
    eventId: Sequelize.INTEGER,
    userId: Sequelize.INTEGER,
    status: {type: Sequelize.BOOLEAN, defaultValue: true}
}, {
    tableName: 'subscriptions',
    freezeTableName: false,
    timestamp: false
});
```

Рисунок 3.5 – структура таблиці subscriptions

Четверта таблиця – «palces» - таблиця, яка зберігає інформацію про місце проведення спортивних подій (рисунок 3.6) .

```
const PlaceModel = sequelize.define('place', {
    id: {type: Sequelize.INTEGER, primaryKey: true, autoIncrement: true},
    name: Sequelize.STRING,
    lat: Sequelize.FLOAT,
    lng: Sequelize.FLOAT
}, {
    tableName: 'places',
    freezeTableName: false,
    timestamp: false
});
```

Рисунок 3.6 – структура таблиці subscriptions

3.3 Встановлення зав’язків між таблицями

За допомогою додатка Sequelize було восстановлено[10] наступні залежності (Рисунок 3.7). Це дає можливість за один запит отримати інформацію зразу з декількох таблиць.

Відношення «hasMany» говорить по те, що подія має багато підписок і навпаки, підписка залежить від поді.

Відношення «hasOne» говорить по те, що подія має одну сутність в наступній таблиці.

```
83  EventModel.hasMany(EventSubscription, {foreignKey: 'eventId', sourceKey: 'id'});
84
85  EventSubscription.belongsTo(EventModel, {foreignKey: 'eventId', targetKey: 'id'});
86
87
88  PlaceModel.hasOne(EventModel,{foreignKey: 'placeId', sourceKey: 'id'});
89
90  EventModel.belongsTo(PlaceModel,{foreignKey: 'placeId', sourceKey: 'id'});
91
92  UserModel.hasOne(EventSubscription, {foreignKey: 'userId', sourceKey: 'id'});
93
94  EventSubscription.belongsTo(UserModel, {foreignKey: 'userId', sourceKey: 'id'});
```

Рисунок 3.7 – Встановлення зав’язків між таблицями

4. ОПИС ПРОГРАМНОГО ІНТЕРФЕЙСУ

Під час написання дипломної роботи був розроблений програмний комплекс, який дозволяє оброблювати інформацію від клієнта.

4.1 Інсталяція та системні вимоги

Через те, що серверний додаток оброблює інформацію від клієнтської частини – він не потребує інсталяції на пристрій користувача. Для встановлення серверної перш за все необхідно встановити програмний комплекс NodeJs версії 10.16.01. Це можна зробити на офіційному сайті. NodeJs підтримує три операційні системи: Winodws, MacOs, Linux.

Після цього необхідно інсталювати базу даних PostgreSQL. Після установки бази даних необхідно створити нову базу даних «sportPlace».

Для запуску серверу на мові NodeJs необхідно встановити пакети, які зазначені в файлі package.json. Це можна зробити, якщо перейти в корінь директорії проекту і використати команду «npm install» (рисунок 4.1)

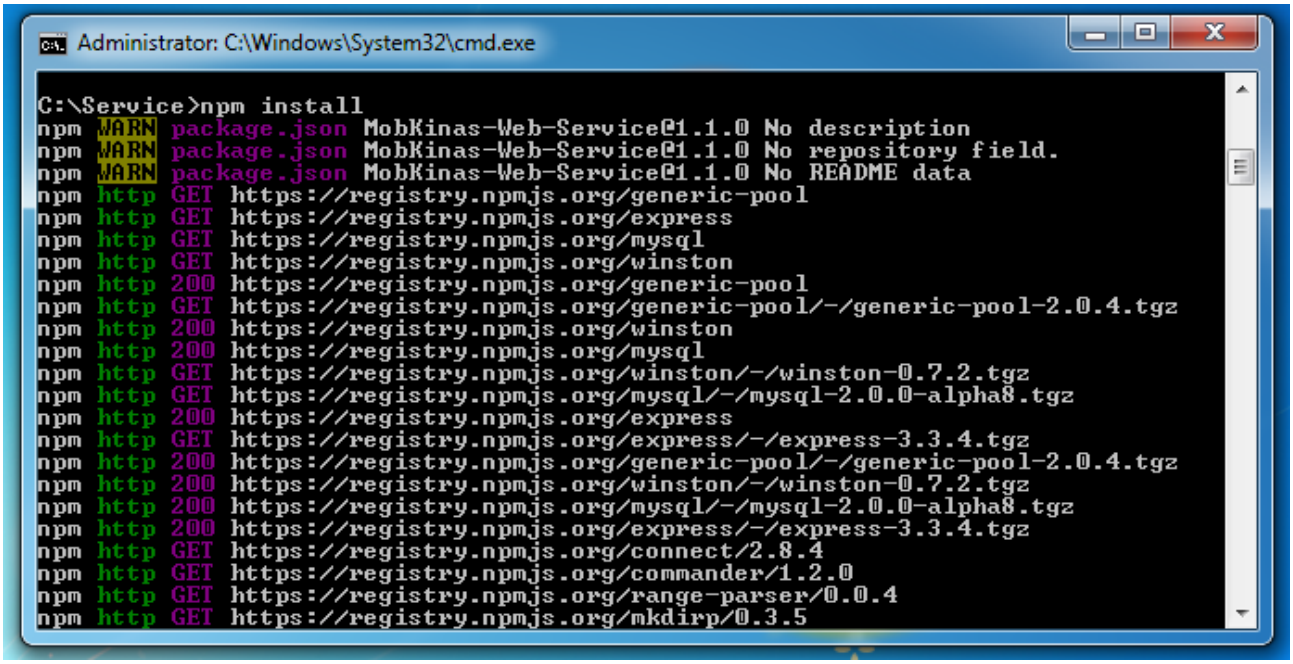

Рисунок 4.1 – Завантаження програмних пакетів для серверної частини

Після завантаження пакетів необхідно відредагувати в теці config, файл config.js і вказати в розділі «PROD» логін і пароль від бази даних PostgreSQL, який було впроваджено в базу даних при інсталяції.

В консолі повинно з’явитися повідомлення «Server is running», що означає про успішний запуск серверного додатку.

4.2 Інструкція з використання програмного інтерфейсу

Щоб використати програмний інтерфейс серверного додатку, необхідно виявити ip-адресу серверу, на якому було запущено програмний додаток. Це можна зробити за допомогою команди ifconfig (рисунок 4.2).

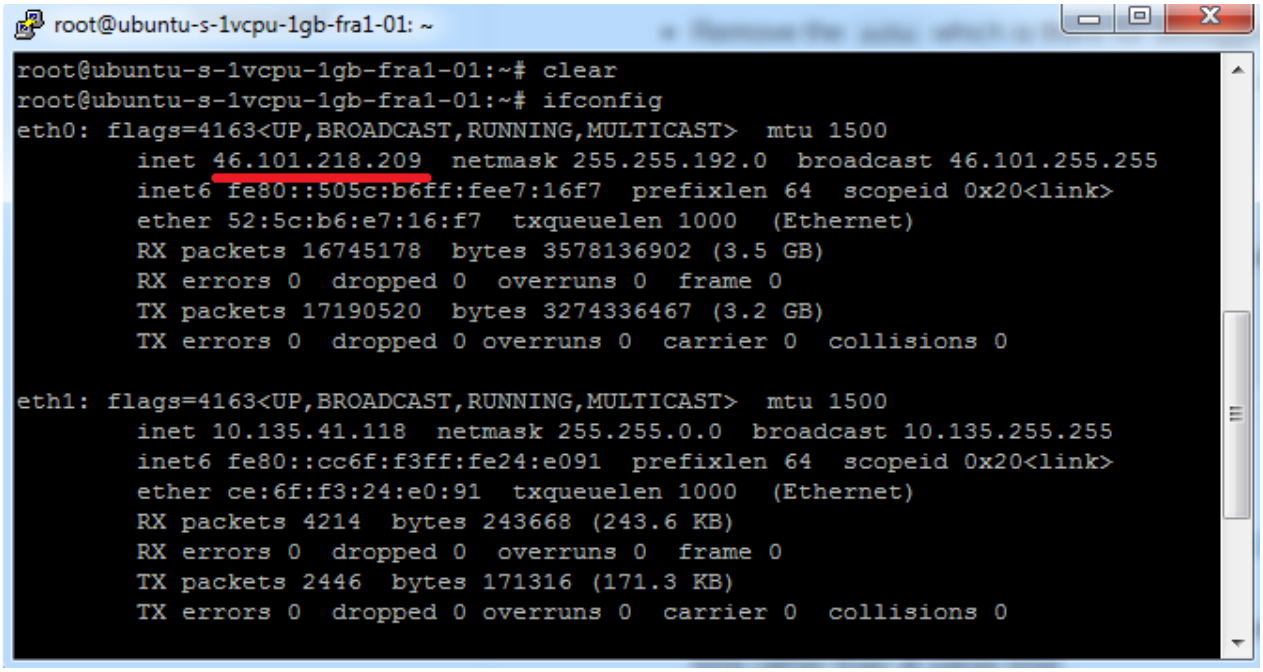

Рисунок 4.2 — Виявлення ip-адреси серверу

Для початку роботи з серверним додатком, необхідно авторизуватись. Це можливо зробити двома шляхами:

— Ввійти в систему з існуючою поштовою скринькою і паролем

— Зареєструватись в системі

Розглянемо запит створення нового користувача. Для цього необхідно сформувати запит POST через протокол HTTP і передати в тілі такі параметри:

— email – електронна скринька користувача;

— password – пароль користувача;

— username – коротке ім’я;

— firstName – Ім’я користувача;

— lastName – Фамілія користувача;

Також для запитів типу POST необхідно в з заголовці запиту передати поле «Content-Type» і встановити йому значення «application/x-www-form-urlencoded» і відправити по шляху «/api/register», вказавши перед цім протокол HTTP і ip-адресу серверу. Після успішної реєстрації, сервер поверне відповідь у форматі JSON.

Успішна відповідь буде містити в собі поле «token», яке необхідне для подальшої роботи з сервером від ім’я користувача. Надалі всі запити повинні мати заголовок «Authorization» в якому буде знаходитися даний токен.

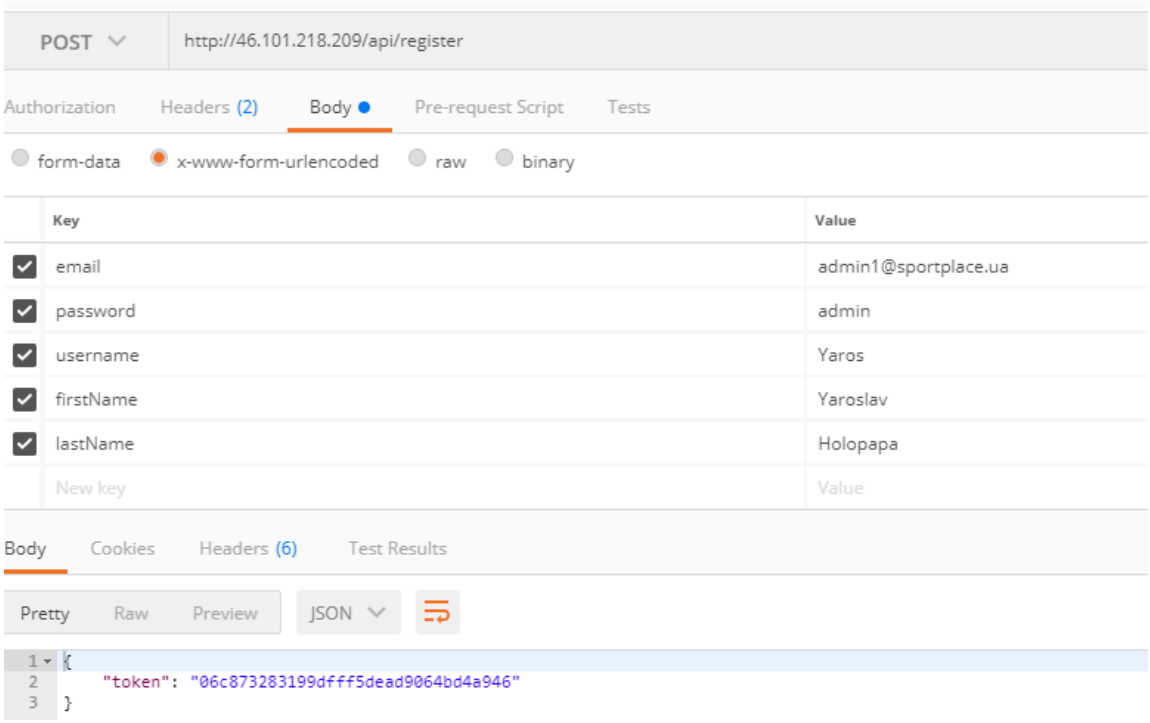

Рисунок 4.2 — Запит реєстрації

Розглянемо запит на отримання інформації про доступні спортивні майданчики, які обслуговує сервіс. Для цього нам необхідно відправити GET-запит на шлях «api/places». Після успішного виконання, сервер поверне нам масив з інформцією про доступні спортивні площадки (рисунок 4.3).

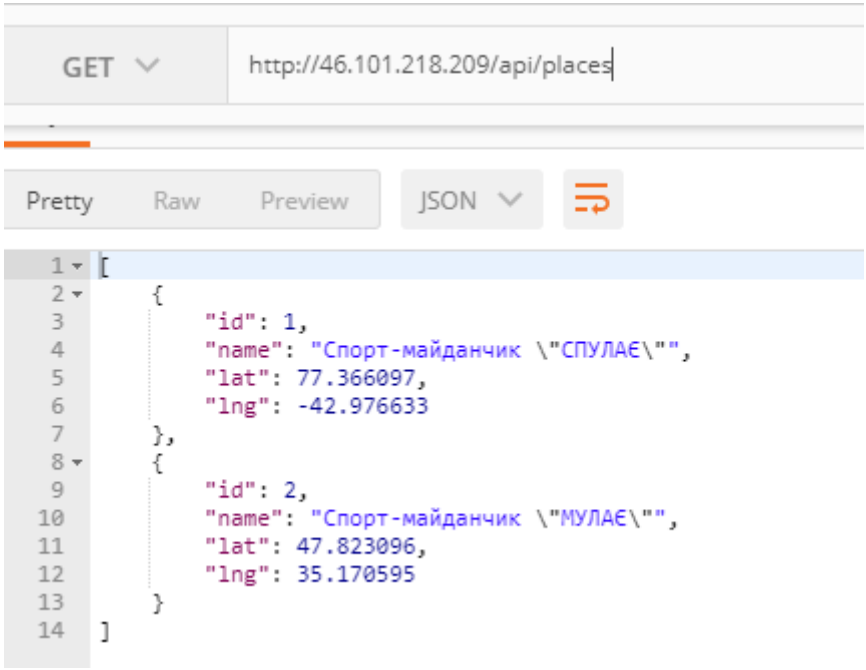

Рисунок 5.4 — Отримання списку спортивних-майданчиків

Розглянемо запит на створення нової спортивної події на вказаному спортивному майданчику. Для даного запиту, необхідно мати секретний ключ «token», який ми отримали при авторизації користувача. Цей ключ необхідно вказати в заголовку «Authorization» (рисунок 4.4). Після чого, треба створити POST-запит и передати в його тілі такі параметри:

— type - тип події;

— capacity - кількість гравців;

— description - опис події;

— duration - тривалість в міллісекундах;

— placeID - ідентифікатор спортивного майданчику;

— startEventTime - початок події в форматі timestamp;

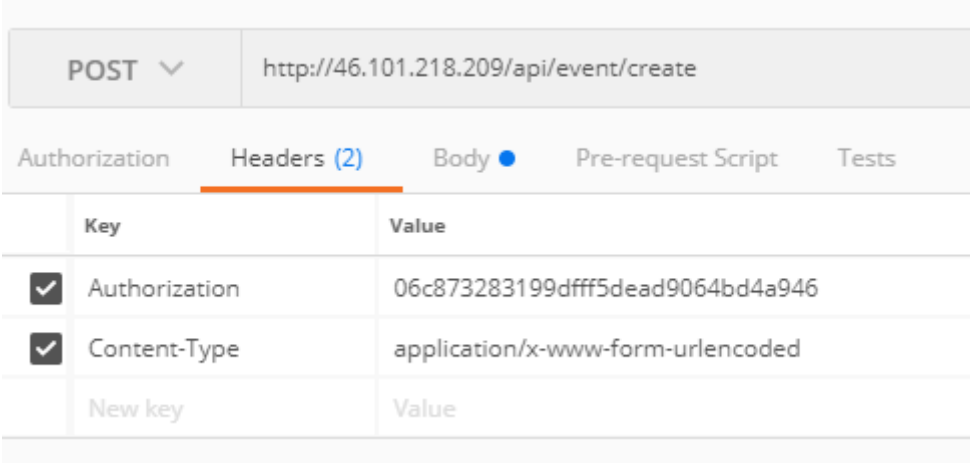

Рисунок 4.4 — Вказання заголовку «Authorization»

В разі успіху, сервер відповість статусом 200 (успішне виконаня запиту). В разі помилки – відповість статусом 418(I’m teapod) і поверне опис помилки.

Аналогічною є процедура редагування події. Для цього необхідно передати потрібні поля, які потрібно змінити на шлях «/api/event/edit».

Для отримання всіх майбутніх подій, необхідно виконати GET-запит на шлях «/event/all». Аналогічною є процедура запиту створених користувачем подій (шлях «event/created») і подій до яких приєднався користувач (шлях «event/joined»). В разі успіху сервер поверне JSON масив з даними про події (рисунок 4.5).


```
{
    "startEventTime": 1569868381406,
    "id": 14,
    "type": 1,
    "creator": 18,
    "name": "Фінал ліги-чемпіонів КПІ",
    "capacity": 22,
    "duration": 2700000,
    "description": "Футбольне змагання між факультетами за звання кращої команди КПІ",
    "placeId": 1,
    "joinedCount": 0,
    "joined": false,
    "place": {
        "id": 1,
        "name": "Спорт-майданчик \"СПУЛАЄ\"",
        "lat": 77.366097,
        "lng": -42.976633
    }
}
```


Рисунок 4.5 — Відповідь на запит майбутніх подій

Розглянемо запит для приєднання до створеної події. Для даного запиту необхідно сформувати POST-запит на шлях «api/event/join». Аналогічною є процедура відписки від події зі шляхом «api/event/leave». В тіло запиту необхідно передати параметр «eventId» і надати йому необхідний ідентифікатор події.

Після успішної реєстрації на подію, сервер поверне вам об’єкт електронного-квитка (рисунок 4.6).

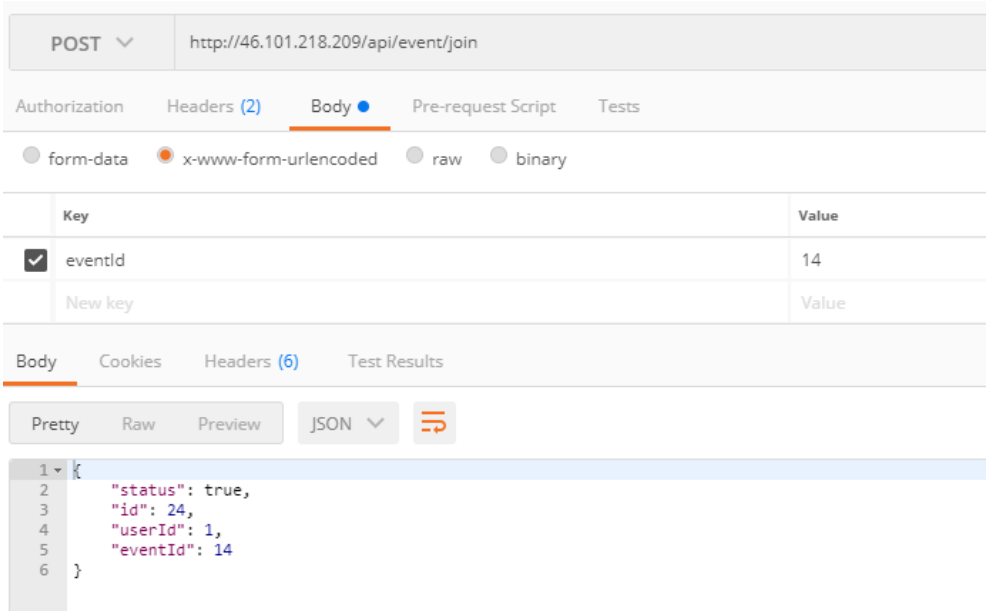

Рисунок 5.6 — Одержання електронного-квитка

Для отримання інформації про користувача, і дізнатися, чи отримав користувач доступ для користуванням сервісу, необхідно сформувати GET запит по шляху «api/me». В разі успіху – сервер відповість обєктом (рисунок 4.7). Аналогічно, для редагування даних про користувача, необхідно зробити POST запит по шляху «api/me/edit» і передати необхідні поля для редагування (рис 4.8).

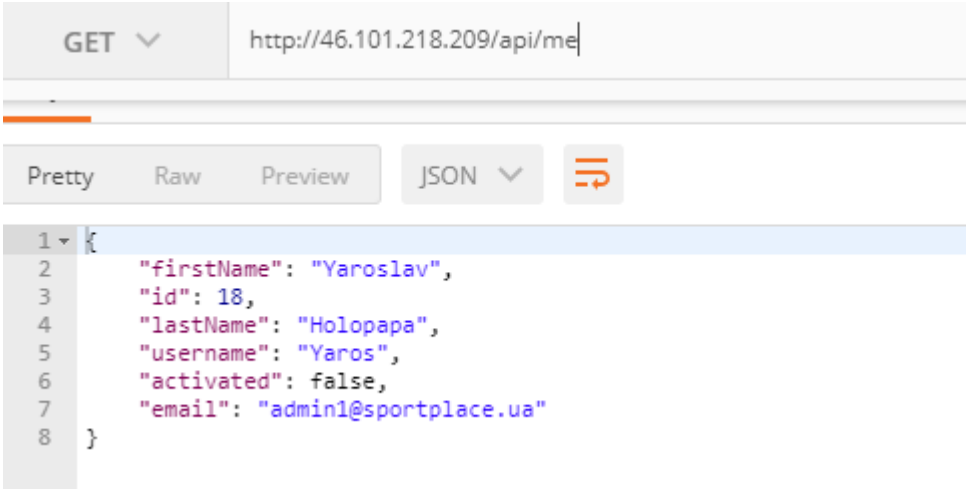

Рисунок 4.7 — Отримання інформації про користувача

Для інформування клієнту, необхідно звернути увагу на поле «activated», яке має бульове значення і в разі значення «true» користувач пройшов авторизацію і має повноцінний доступ до сервісу. Це дає можливість створювати і приєднуватися до подій.

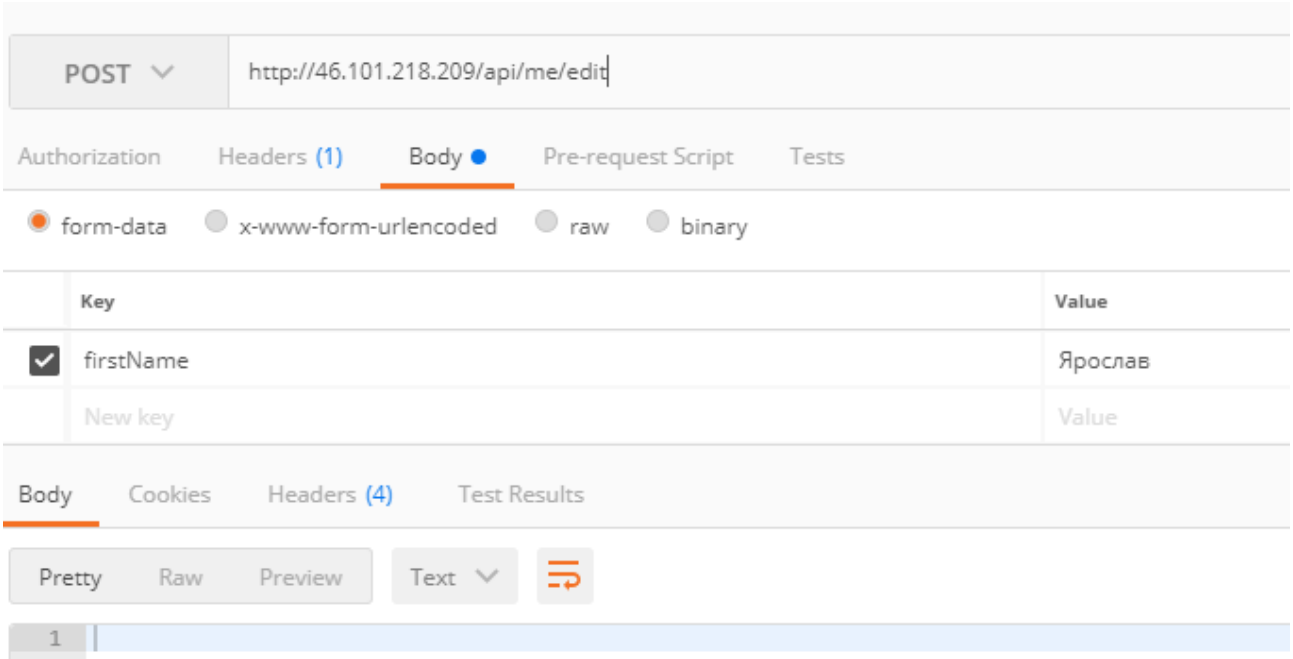

Рисунок 4.8 – Редагування інформації про користувача

ВИСНОВКИ

Під час написання диплому і проходження переддипломної практики були покращені навички проектування архітектури програмного забезпечення та його розробки за допомогою середовища розробки WebStorm та мови програмування JavaScript, що дозволило у повній мірі використати MVC, тобто розділяти обов’язки в програмному коді.

В процесі написання диплому було вивчено і проаналізовано декілька сервісів бронювання, що покращило бачення основних проблем, недоліків і переваг. Це дало гарний фундамент для побудови бізнес-логіки на початкових стадія.

В процесі розробки було використано велику кількість технологій, за допомогою яких можна реалізувати серверний додаток. Він був спроектований належим чином, що дозволяє масштабувати систему не змінюючи програмного коду. Також за допомогою бібліотеки Express стало можливо робити кластери серверів для вирівняння навантаження.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Саломаа А. Криптография с открытым ключом. Пер. с англ. – М.: Мир, 1995. – 318 с.
2. Смарт Н. Криптография. Москва: Техносфера, 2005. – 528 с.
3. JavaSript [Електронний ресурс]. – Режим доступу: https://learn.javascript.ru/. Дата доступу: травень 2019. Назва з екрану.
4. NodeJs expressjs [Електронний ресурс]. – Режим доступу: https://metanit.com/web/nodejs/4.1.php. Дата доступу: травень 2019. Назва з екрану.
5. Anthony Gore — Full‑Stack Vue.js 2 and Laravel 5 [Електронний ресурс]. — 2015. – Режим доступу: https://bit.ly/2OEODzR.
6. React – A JavaScript library for building user interfaces [Електронний ресурс]. – Режим доступу: https://reactjs.org/
7. Изучаем Node.js. - М.: Питер, 2013. - 400 с.
8. Кантелон, М. Node.js в действии / М. Кантелон. - М.: Питер, 2015. - 810 с.
9. Сухов, Кирилл Node.js. Путеводитель по технологии / Кирилл Сухов. - М.: ДМК Пресс, 2015. - 416 с.
10. Хэррон, Дэвид Node.js Разработка серверных веб-приложений на JavaScript / Дэвид Хэррон. - М.: ДМК Пресс, 2014. - 144 с.
11. Хэррон, Дэвид Node.js Разработка серверных веб-приложений на JavaScript / Дэвид Хэррон. - Москва: СПб. [и др.] : Питер, 2014. - 144 с.

ДОДАТОК А

Розробка спеціалізованого Інтернет-сервісу “Відкритий спортмайданчик з е-сервісами”(back-end)

Специфікація

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТВ51145_19Б

Аркушів 2

Київ 2019

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТВ51145_19 Б	**Записка.docx**	**Текстова частина дипломної роботи**
Компоненти		
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТВ51145_19 Б 12-1	**maydanchik.zip**	**Основний архів з кодом на мові JavaScript**

ДОДАТОК Б

Розробка спеціалізованого Інтернет-сервісу “Відкритий спортмайданчик з е-сервісами”

Текст програми

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТВ51143_19Б

Аркушів 3

Київ 2019

Текст серверного додатку

```
/* LIBS */
const express = require('express');
const fs = require('fs');
const http = require('http');
const https = require('https');
const bodyParser = require("body-parser");
const path = require('path');
const morgan = require('morgan')

const DEV = process.env.NODE_ENV === 'dev';
// const DEV = true;

const server = express();

server.use(bodyParser.json());
server.use(bodyParser.urlencoded({extended: true}));



   server.listen(80, () => console.log('run'));


const accessLogStream = fs.createWriteStream(path.join(__dirname, '/logs/access.log'), {flags: 'a'});

server.use(morgan('combined', {stream: accessLogStream}));


server.use(require('./routes/api'));

server.get('/ping', (req, res) => {
   res.status(200).send("pong");
});
```

Текст Серверного додатку

```
const express = require("express");
const Sequelize = require("sequelize")
const check = require('express-validator/check').check;
const validationResult = require('express-validator/check').validationResult;
const router = express.Router();
const md5 = require("md5");
const crypto = require("crypto");
const jwt = require("jsonwebtoken");
const authMiddleWare = require("../middleware/auth");
const User = require("../models/models").User;
const Event = require("../models/models").Event;
const Place = require("../models/models").Place;
const Subscription = require("../models/models").Subscripton;



router.post("/api/register", [
   check('email').isEmail(),
   check('firstName').isLength({min: 2, max: 15}),
   check('lastName').isLength({min: 2, max: 15}),
   check('username').isLength({min: 3, max: 15}),
   check('password').isLength({min: 5})
], userRegister);
router.post("/api/login", userLogin);

router.get("/api/me", authMiddleWare, userMe);
router.post("/api/me/edit", authMiddleWare, userEdit);
```

```
//Events methods
router.post("/api/event/create", authMiddleWare, createEvent); //
router.post("/api/event/edit", authMiddleWare, editEvent);
router.post("/api/event/join", authMiddleWare, joinEvent); //
router.post("/api/event/leave", authMiddleWare, leaveEvent); //
router.get("/api/event/all", authMiddleWare, getAllEvent); //
router.get("/api/event/created", authMiddleWare, getEventCreated);//
router.get("/api/event/joined", authMiddleWare, getEventJoined);//
router.get("/api/places", authMiddleWare, getAllPlaces); //


async function getAllPlaces(req, res) {
   const places = await Place.findAll();
   res.send(places);
}

function userRegister(req, res) {

   const errors = validationResult(req);
   if (!errors.isEmpty()) {
      return res.status(422).json({errors: errors.array()});
   }

   const email = req.body.email;
   const password = req.body.password;
   const username = req.body.username;
   const firsName = req.body.firstName;
   const lastName = req.body.lastName;
   const generatedSalt = crypto.randomBytes(16).toString();
   const md5Password = md5(generatedSalt + password + generatedSalt);
   const ip = req.headers["X-Forwarded-For"] || req.connection.remoteAddress;
   User.findOrCreate({
      where: {email: email},
      defaults: {
         password: md5Password,
         username: username,
         salt: generatedSalt,
         firstName: firsName,
         lastName: lastName,
         token: md5Password,
         ip: ip,
         register_date: Date.now()
      }
   }).spread((user, created) => {
      if (created)
         res.status(200).send({token: user.token});
      else
         res.status(422).send({error: "Email занят!"});
   });


}
```

```
async function userLogin(req, res) {
    try {
        const email = req.body.email;
        const userByEmail = await User.findOne({where: {email: email}});
        if (!userByEmail) {
            res.status(401).send();
            return;
        }
        const userSalt = userByEmail.dataValues.salt;
        const userSaltedPassword = userByEmail.dataValues.password;
        const md5ClientPassword = md5(userSalt + req.body.password + userSalt);
        if (userSaltedPassword === md5ClientPassword) {
            res.status(200).send({token: userByEmail.dataValues.token});
        }
        else
            res.status(401).send();

    } catch (err) {
        res.status(500).send();
        console.log(err);
    }
}

async function userMe(req, res) {
    res.status(200).send(req.user);
}

async function userEdit(req, res) {
    const userId = req.user.id;
    const firstName = req.body.firstName;
    const lastName = req.body.lastName;
    const username = req.body.username;

    const fieldsToUpdate = {
        ...firstName && {firstName: firstName},
        ...lastName && {lastName: lastName},
        ...username && {username: username}
    };

    User.findByPk(userId).then(user => {
        if (user) {
            user.update(fieldsToUpdate).catch(err => console.error);
            res.status(200).send();
        }
        else
            res.status(401).send();
    }).catch(err => {
        console.log(err);
        res.status(500).send();
    })
}
```

```
async function joinEvent(req, res) {
    const user = req.user;

    if (isNaN(+req.body.eventId)) {
        res.status(418).send();
        return;
    }
    const eventId = +req.body.eventId;

    const event = await Event.findByPk(eventId, {
        where: {
            startEventTime: {
                [Sequelize.Op.gt]: Date.now()
            }
        }
    });
    if (!event) {
        res.status(418).send({error: {description: "Event not found"}});
        return;
    }
    let count = await Subscription.count({
        where: {
            eventId: eventId,
            status: true
        }
    });

    if (count >= event.dataValues.capacity) {
        res.status(418).send({error: {description: "Event is full"}});
        return;
    }
    const subscription = await Subscription.findOrCreate({
        where: {
            userId: user.id,
            eventId: eventId,
        }
    }).spread((sub) => {
        sub.update({status: true});
        return sub;
    });
    res.send(subscription);
}

async function leaveEvent(req, res) {
    const user = req.user;
    const eventId = +req.body.eventId;
    const event = await Event.findByPk(eventId);
    if (!event) {
        res.status(418).send();
        return;
    }
```

```
    const subscription = await Subscription.findOne({
        where: {
            userId: user.id,
            eventId: eventId,
        }
    }).then((sub) => {
        if (sub) {
            sub.update({status: false});
            return sub;
        }
    });
    if (!subscription) {
        res.status(418).send({error: {description: "Bad eventId."}});
        return;
    }
    res.send(subscription);
}

async function getAllEvent(req, res) {

    let result = await Event.findAll({
        where: {
            startEventTime: {
                [Sequelize.Op.gt]: Date.now()
            }
        },
        attributes: {
            include: [[Sequelize.fn('COUNT', Sequelize.col('subscriptions.id')), 'joinedCount'], [Sequelize.where(Sequelize.fn('COUNT',
Sequelize.col('subscriptions.id')), '>', 0), 'joined']]
        },
        group: ['event.id', 'place.id', 'subscriptions.id'],
        include: [{
            model: Subscription,
            attributes: []
        }, {
            model: Place
        }]
    });

    res.json(result);
}

async function createEvent(req, res) {
    const name = req.body.name;
    const type = req.body.type;
    const creator = req.user.id;
    const capacity = req.body.capacity;
    const description = req.body.description;
    const duration = +req.body.duration;
    const placeId = +req.body.placeID;
    let startTime = +req.body.startEventTime;
```

```
if (!name || !type || !creator || !capacity || !duration || !startTime) {
   res.status(418).send({error: {description: "Invalid data"}});
   return;
}

if (isNaN(duration)|| isNaN(placeId) || isNaN(type) || isNaN(capacity) || isNaN(startTime)) {
   res.status(418).send();
   return;
}

startTime = new Date(startTime);

const options = {timezone: 'Etc/GMT'};
const startTimeSql = Sequelize.DATE.prototype._stringify(startTime, options);
const endTimeSql = Sequelize.DATE.prototype._stringify(new Date(startTime.getTime() + duration), options);
console.log(startTime, new Date(startTime.getTime() + duration));
const eventsAtSameTime = await
   Event.findAll({
         where: {
            [Sequelize.Op.or]: [
               {
                  startEventTime: {
                     [Sequelize.Op.lte]: startTime,
                     [Sequelize.Op.gte]: Sequelize.literal(`'${startTimeSql}'::timestamp - event.duration * interval '1 millisecond'`)
                  }
               },
               {
                  startEventTime: {
                     [Sequelize.Op.lte]: startTime,
                     [Sequelize.Op.gte]: Sequelize.literal(`'${endTimeSql}'::timestamp - event.duration * interval '1 millisecond'`)
                  }
               },
               {
                  startEventTime: {
                     [Sequelize.Op.gte]: startTime,
                     [Sequelize.Op.lte]: Sequelize.literal(`'${endTimeSql}'::timestamp - event.duration * interval '1 millisecond'`)
                  }
               }
            ]
         }
      }
   );


if (eventsAtSameTime.length > 0) {
   res.status(418).send({error: {description: "Date is ordered."}});
   return;
}

Event.create({
   type: type,
```

```
        creator: creator,
        name: name,
        placeId: placeId,
        capacity: capacity,
        startEventTime: startTime,
        duration: duration,
        description: description
    });
    res.json();
}

async function editEvent(req, res) {
    const name = req.body.name;
    const type = req.body.type;
    const eventId = req.body.eventId;
    const creator = req.user.id;
    const capacity = req.body.capacity;
    const description = req.body.description;
    const duration = +req.body.duration;
    let startTime = +req.body.startEventTime;

    console.log(startTime);
    if (!name || !type || !creator || !capacity || !duration || !startTime) {
        res.status(418).send({error: {description: "Неправельный ввод!"}});
        return;
    }

    if (isNaN(duration) || isNaN(type) || isNaN(capacity) || isNaN(startTime)) {
        res.status(418).send();
        return;
    }

    startTime = new Date(startTime);
    if (startTime < new Date()) {
        res.status(418).send({error: {description: "Start time is lower than now."}});
        return;
    }

    const options = {timezone: 'Etc/GMT'};
    const startTimeSql = Sequelize.DATE.prototype._stringify(startTime, options);
    const endTimeSql = Sequelize.DATE.prototype._stringify(new Date(startTime.getTime() + duration), options);
    const eventsAtSameTime = await
        Event.findAll({
            where: {
                [Sequelize.Op.or]: [
                    {
                        startEventTime: {
                            [Sequelize.Op.lte]: startTime,
                            [Sequelize.Op.gte]: Sequelize.literal(`'${startTimeSql}'::timestamp - event.duration * interval '1 millisecond'`)
                        }
                    },
                    {
```

```
                startEventTime: {
                    [Sequelize.Op.lte]: startTime,
                    [Sequelize.Op.gte]: Sequelize.literal(`'${endTimeSql}'::timestamp - event.duration * interval '1 millisecond'`)
                }
            },
            {
                startEventTime: {
                    [Sequelize.Op.gte]: startTime,
                    [Sequelize.Op.lte]: Sequelize.literal(`'${endTimeSql}'::timestamp - event.duration * interval '1 millisecond'`)
                }
            }
        ]
    }
  }
);


if (eventsAtSameTime.length > 0) {
    res.status(418).send({error: {description: "Date is ordered."}});
    return;
}

let event = await Event.findOne({where: {id: eventId}});
if (!event) {
    res.status(418).send();
    return;
}
event.destroy();

Event.create({
    type: type,
    creator: creator,
    name: name,
    capacity: capacity,
    startEventTime: startTime,
    duration: duration,
    description: description
});
res.json();
}

async function getEventCreated(req, res) {
    const user = req.user;
    let result = await Event.findAll({
        where: {
            creator: user.id,
            startEventTime: {
                [Sequelize.Op.gt]: Date.now()
            }
        },
        attributes: {
```

```
        include: [[Sequelize.fn('COUNT', Sequelize.col('subscriptions.id')), 'joinedCount'], [Sequelize.where(Sequelize.fn('COUNT', Sequelize.col('subscriptions.id')), '>', 0), 'joined']]
      },
      group: ['event.id', 'subscriptions.id','place.id'],
      include: [{
        model: Subscription,
        attributes: []
      }, {
        model: Place
      }]
  });
  res.json(result);
}

async function getEventJoined(req, res) {
  const user = req.user;
  let result = await Event.findAll({
      where: {
        startEventTime: {
          [Sequelize.Op.gt]: Date.now()
        }
      },
      attributes: {
        include: [[Sequelize.fn('COUNT', Sequelize.col('subscriptions.id')), 'joinedCount'], [Sequelize.where(Sequelize.fn('COUNT', Sequelize.col('subscriptions.id')), '>', 0), 'joined']]
      },
      group: ['event.id', 'subscriptions.id','place.id'],
      include: [{
        model: Subscription,
        where: {userId: user.id},
        attributes: []
      }, {
        model: Place
      }]
  });
  res.json(result);
}


module.exports = router;
```

Текст серверного додатку

```
const sequelize = require('../db/db').sequelize;
const Sequelize = require('sequelize');

const UserModel = sequelize.define('user', {
  id: {type: Sequelize.INTEGER, primaryKey: true, autoIncrement: true},
  password: Sequelize.STRING,
  ip: Sequelize.STRING,
  firstName: Sequelize.STRING,
  lastName: Sequelize.STRING,
  register_date: {
```

```
        type: Sequelize.DATE
    },
    username: Sequelize.STRING,
    salt: Sequelize.STRING,
    token: Sequelize.STRING,
    email: Sequelize.STRING,
    activated: {type: Sequelize.BOOLEAN, defaultValue: false}

}, {
    tableName: 'users',
    freezeTableName: false,
    timestamp: false
});

const EventModel = sequelize.define('event', {
    id: {type: Sequelize.INTEGER, primaryKey: true, autoIncrement: true},
    type: Sequelize.INTEGER,
    creator: Sequelize.INTEGER,
    name: Sequelize.STRING,
    capacity: Sequelize.INTEGER,
    startEventTime: {
        type: Sequelize.DATE,
        get () {
            return new Date(this.getDataValue('startEventTime')).getTime();
        }
    },
    duration: Sequelize.BIGINT,
    description: Sequelize.STRING
}, {
    tableName: 'events',
    freezeTableName: false,
    timestamp: false
});

const EventSubscription = sequelize.define('subscription', {
    id: {type: Sequelize.INTEGER, primaryKey: true, autoIncrement: true},
    eventId: Sequelize.INTEGER,
    userId: Sequelize.INTEGER,
    status: {type: Sequelize.BOOLEAN, defaultValue: true}
}, {
    tableName: 'subscriptions',
    freezeTableName: false,
    timestamp: false
});

const GroupModel = sequelize.define('group', {
    id: {type: Sequelize.INTEGER, primaryKey: true, autoIncrement: true},
    name: Sequelize.STRING,
    creator: Sequelize.INTEGER,
    hash: Sequelize.STRING,
    participants: Sequelize.ARRAY(Sequelize.INTEGER)
}, {
```

```
    tableName: 'groups',
    freezeTableName: false,
    timestamp: false
});

const PlaceModel = sequelize.define('place', {
    id: {type: Sequelize.INTEGER, primaryKey: true, autoIncrement: true},
    name: Sequelize.STRING,
    lat: Sequelize.FLOAT,
    lng: Sequelize.FLOAT
}, {
    tableName: 'places',
    freezeTableName: false,
    timestamp: false
});

//TODO: PLACES MODEL
//TODO: RULES
// const PlaceModel = sequelize.defin

EventModel.hasMany(EventSubscription, {foreignKey: 'eventId', sourceKey: 'id'});

EventSubscription.belongsTo(EventModel, {foreignKey: 'eventId', targetKey: 'id'});


PlaceModel.hasOne(EventModel,{foreignKey: 'placeId', sourceKey: 'id'});

EventModel.belongsTo(PlaceModel,{foreignKey: 'placeId', sourceKey: 'id'});

UserModel.hasOne(EventSubscription, {foreignKey: 'userId', sourceKey: 'id'});

EventSubscription.belongsTo(UserModel, {foreignKey: 'userId', sourceKey: 'id'});

sequelize.sync({alter: true}).then((data) => console.log).catch(err => {
    console.log(err);
});
module.exports.User = UserModel;
module.exports.Event = EventModel;
module.exports.Subscripton = EventSubscription;
module.exports.Place = PlaceModel;
// ServiceModel.hasMany(ActivationModel, {foreignKey: 'service', sourceKey: 'id'});
// ActivationModel.belongsTo(ServiceModel, {foreignKey: 'service', targetKey: 'id'});
```

Текст сервенрного додатку

```
const User = require("../models/models").User;

module.exports = function (req, res, next) {
    const token = req.get('Authorization');
    if (!token) {
        res.status(401).send();
        return;
    }
    User.findOne({
        where: {token: token},
```

```
        attributes: ['firstName', 'id', 'lastName', 'username', 'activated', 'email']
    }).then((result) => {
        if (!result)
            res.status(401).send();
        else {
            req.user = result.dataValues;
            next();
        }
    }).catch(err => {
        console.log(err);
        res.status(500).send();
    });

};
```

Текст сервернрго додатку

```
const Sequelize = require('sequelize');
const config = require("../config/config").pgconfig;
require('pg').defaults.parseInt8 = true
const DEV = process.env.NODE_ENV === 'dev';
const PgConfig = DEV ? config.dev : config.prod;
const db = new Sequelize('sportPlace', PgConfig.username, PgConfig.password, {
    host: 'localhost',
    dialect: 'postgres',
    logging: DEV,
    pool: {
        max: 5,
        min: 0,
        acquire: 30000,
        idle: 10000
    },
    define: {
        timestamps: false
    },
    dialectOptions: {
        decimalNumbers:true,
        useUTC: true //for reading from database
    },
    timezone: '+00:00',
    operatorsAliases: false
});


module.exports.sequelize = db;
```

Текст серверного додатку

```
 module.exports.pgconfig = {
    dev: {
        username: "postgres",
        password: "123"
    },
    prod: {
        username: "pgsport",
        password: "sport123"
    }
}
```

ДОДАТОК В

Розробка спеціалізованого Інтернет-сервісу “Відкритий спортмайданчик з е-сервісами”

Опис програми

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТВ51143_19Б

Аркушів 9

Київ 2019

АНОТАЦІЯ

Даний додаток містить опис програми для серверної частини. Створений серверний додаток виконує такі задачі:

— обробка запитів по протоколу HTTP;

— збереження даних в базу даних;

— виконувати особливі правила бізнес-логіки;

— відправлення даних на сервер видачі пропуску .

Програма взаємодіє з клієнтом завдяки програмного інтерфейсу

При розробці серверного додатку було використано мову JavaScript та SQL у середовищі програмування WebStorm з використанням системи автоматичної збірки проекту npm.

ЗМІСТ

ЗАГАЛЬНІ ВІДОМОСТІ

У цьому додатку міститься опис сервісу бронювання який був спроектований і розроблений. У додатку Б міститься програмний код головних модулів розроблюваної системи.

Серверний додаток працює на операційній системі Windows 7, для правильої роботи необхідно мати встановлений інтерпритатор NodeJs і пакетний менеджер NPM.

При розробці мобільного налаштунку було використано:

- **Середовище розробки WebStorm;**
- **Система автоматичної збірки проекту NPM.**

ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Розроблені компоненти виконують завдання бронювання спортивних подій, а саме бронюють певний інтервал часу на певному спортивному майданчику.

Розроблений додаток можна інтегрувати та використовувати у реальному житті з реальними спортивними майданчиками з е-сервісами.

ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Для реалізації мобільного додатку знадобилось використання багатьох бібліотек та фреймворків. В проект було впроваджено архітектурний шаблон MVC.

В проекті було використано бібліотеку Express для впровадження обробки даних і спілкування клієнт-сервер за допомогою протоколу HTTP

Для роботи з базами даних було використано бібліотеку Sequelize

Для асинхронності було використано бібліотеку RxJava.

Для віправлення запитів для створення подій і перегляд необхідно авторизуватися через програмний інтерфейс, після чого буде виданий ключ, який буде ідинтифікувати користувача при кожному запиті.. Після того як користувач авторизується, йому буде надана можливість бронювання місця у події та можливість створення власної події.

ВИКОРИСТОВУВАНІ ТЕХНІЧНІ ЗАСОБИ

Для забезпечення повноцінної роботи та досягнення високої ефективності роботи створеного додатка було використано середовище розробки WebStrom, набір інструментів розробника бази даних PgAdmin та мову JavaScript.

Розроблений додаток працює в операційній системі Windows 7 та новіше.

ВИКЛИК І ЗАВАНТАЖЕННЯ

Розроблена система потребує інсталяції інтерпритатору NodeJs, PostgreSQL і необхідних пакетів під операційну систему Windows. Необхідно запустити команду «npm run serve» . Після запуску користувач отримає доступ до всього функціоналу додатка.

ВХІДНІ І ВИХІДНІ ДАНІ

Вхідними даними для розроблених додатків є інформація яка зчитується з конфігураційних файлів.

Вхідні дані мають вигляд текстового файлу у форматі JSON.

Вихідними даними є дані сервера, які відправляются у вигляді JSON-об’єктів, у вигляді цілочисельних значень та строк.