Міністерство освіти і науки України

Національний університет «Києво-Могилянська академія»

Факультет інформатики

Кафедра інформатики

Курсова робота

освітній ступінь – бакалавр

на тему: «**Розробка мобільного додатку на React Native**»

Виконав: студент 3-го року навчання,

Спеціальності

121 «Інженерія Програмного Забезпечення»

Студента Поети Іл’ї

Керівник Калітовський Б.В.

магістр комп’ютерних наук, асистент

«20» травня 2022 р.

Київ – 2021

Національний університет «Києво-Могилянська академія»

Факультет інформатики

Кафедра інформатики

Освітній ступінь бакалавр

Спеціальність 121 «Інженерія Програмного Забезпечення»

Освітня програма бакалавр

ЗАТВЕРДЖУЮ

Завідувач кафедри інформатики

Гороховський С. С.

“10” жовтня 2020 року

ЗАВДАННЯ
ДЛЯ КУРСОВОЇ РОБОТИ СТУДЕНТУ

Поеті Іл’ї

КАЛЕНДАРНИЙ ПЛАН

№ з/п	ПЕРЕЛІК РОБІТ	Термін виконання	Дата ознайомлення наукового керівника	Підпис наукового керівника	Примітки
1.	Вибір теми, затвердження її на засіданні кафедри та закріплення наукового керівника Узгодження календарного графіка підготовки кваліфікаційної роботи. Ознайомлення студента з критеріями оцінювання кваліфікаційної роботи (п. 8.5).	10 жовтня 2021			
2.	Вивчення джерел літератури, матеріалів архівів, періодичних видань, збір та узагальнення фактів, даних	10 жовтеня 2021 – 2 листопада 2021			
3.	Складання плану каліф. роботи та узгодження з науковим керівником	2 листопада 2021			
4.	Написання розділів роботи	2 листопада 2021 – 01 березня 2022			
5.	Проміжний контроль виконання роботи	01 лютого 2022			
6.	Написання кваліфікаційної роботи в цілому, ознайомлення з її першим варіантом наукового керівника	11 січня 2021 – 29 березня 2021			
	Розділ 1 (постановка проблеми, теоретичні основи, огляд літературних джерел)	25 січня 2022			
	Розділ 2 (аналітично-дослідницька частина)	01 березня 2022			
	Розділ 3 (проектно-рекомендаційна частина)	29 березня 2022			
7.	Повне завершення написання кваліфікаційної роботи, оформлення її згідно з вимогами й подання на відгук науковому керівнику	01 квітня 2022 – 06 травня 2022			
8.	Подання кваліфікаційної роботи для перевірки письмових робіт студентів НаУКМА на відповідність вимогам академічної доброчесності,	7 червня 2021			
9.	Публічний захист кваліфікаційної роботи перед екзаменаційною комісією	згідно з розкладом роботи ЕК			

Графік узгоджено 10 жовтня 2020 р.
Науковий керівник Калітовський Богдан
Виконавець курсової роботи Поета Іл’я

ЗМІСТ

Анотація

У даній роботі розглядаються методи для розробки мобільних додатків на мовах програмування JavaScript і TypeScript за допомогою використання технології React Native на прикладі застосунку, який спрямований на вирішення конкретної задачі. Допоміжними інструментами для розробки серверної частини також були NodeJS, PostgreSQL, Redis. Було створено додаток, який надає змогу користувачу відстежувати кількість людей на локаціях занять спортом.

Вступ

Розробка мобільних додатків – один із найбільш популярних та вибагливих напрямів IT індустрії, який невпинно розвивається. React Native займає ключову позицію в цій сфері, оскільки є одним з небагатьох представників мови програмування JavaScript, а отже, має велику потенційну спільноту і часто стає вибором програмістів, які мали досвід роботи з вище згаданою мовою, і надають їй перевагу серед усіх інших.

Метою роботи було дослідження методів розробки мобільних застосунків за допомогою React Native.

Основним із засобів досягнення мети була розробка додатку, який мав би вирішувати проблему, з якою стикається достатня кількість людей. Нею стала необхідність спортсменів відстежувати кількість людей на місцях для занять спортом з метою вибору локацій з найбільшою, найменшою або оптимальною для того чи іншого виду спорту кількістю гравців або учасників.

Крім цього, було проаналізовано наявні аналоги застосунку, які мають на меті вирішувати поставлені задачі, а також їх переваги та недоліки, які враховувалися при подальшій розробці додатку. Була визначена актуальність проблеми розробки, надійність прийнятих програмних та архітектурних рішень.

Розділ І. Аналіз проблеми і постановка задачі для мобільного застосунку

Аналіз проблеми

Проблема занять вуличними видами спорту або ж на локаціях публічного використання полягає в тому, що спортсмен залежить від інших, які також використовують ті чи інші публічні спортивні майданчики. Через це постає необхідність відстежувати відвідування загально доступних локацій іншими спортсменами.

Основною групою дослідження були футболісти, а також проблема «один в полі гравець», яка щільно пов’язана з вуличними командними видами спорту, – постійна потреба у гравцях.

З іншого ж боку було проведено дослідження серед любителів майданчиків, які спрямовані на проведення вправ на підтримання загальної фізичної підготовки. Якщо у випадку з командними видами спорту постає необхідність пошуку найбільш заповненої локації, то у цьому випадку – спортсмен часто шукає найменш заповнений майданчик.

Крім вище вказаних проблем, виникає необхідність надавати інформацію іншим про своє поточне місце перебування, а саме відвідуваний в даний момент спортивний майданчик, задля спричинення або уникнення спільного користування.

Необхідно також враховувати, що застосунок є аналогом або можливим способом вирішення проблем, які існують, і має легко замістити наявні рішення. Це означає, що додаток має бути простим з точки зору досвіду користувача. А саме надавати легкий та швидкий інтерфейс, який би не потребував значних зусиль зі сторони клієнта та необхідності навчання новому методу вирішення проблеми.

Огляд ринку та наявних рішень

Перед початком розробки було проаналізовано наявні аналоги, а також оцінено їх недоліки та переваги. Серед усіх інших варто виділити

«Площадка»

Даний застосунок є найбільш повноцінним з точки зору наявного функціоналу, однак має значні недоліки. По-перше, у ньому присутній складний процес реєстрації, для початку використання додатку необхідно заповнити дані, підтвердити особу за допомогою номера телефону, що не є зручним та підвищує складність використання, яка є критичною на початку досвіду роботи з додатком. По-друге, на даний момент не доступний для використання на території України.

«OrgMySport»

Другий застосунок дуже схожий на попередній, хоча відрізняється функціоналом та користувацьким інтерфейсом, основними перевагами є зосередження на локаціях майданчиків та наявність внутрішнього чату. Однак має аналогічні суттєві недоліки: ускладнений та не досконалий процес реєстрації і не можливість використання на території України станом на зараз.

«WorkOut»

Останній додаток, але не менш важливий. Його вирізняє з-поміж інших можливість одразу користуватися наявним функціоналом. Тим не менш, інтерфейс є перевантаженим та не досконалим з точки зору дизайну.

Опис рішення

З огляду на аналіз проблеми можна сформулювати задачі та необхідний функціонал: застосунок повинен мати змогу надавати можливість відстежувати кількість людей на локації, крім того, додаток має надавати зручний, швидкий інтерфейс, який би приваблював користувачів. Досвід

клієнта має бути спрощеним до однієї-двох дій з метою приваблення великого кола користувачів і, найголовніше, витіснення наявних рішень, тобто, створення постійного клієнтської бази.

Висновки до розділу 1

Проаналізувавши проблеми та аналоги, які існують на ринку та мають на меті вирішити поставлені питання, а також їх недоліки та переваги, було сформовано список задач відповідно до актуальних задач, які застосунок має вирішити, та аспектів, на які варто звернути увагу при проектуванні та розробці програмного продукту.

Розділ ІІ. Проектування та розробка

Вибір стеку технологій та теоретичні відомості

Клієнтська частина

React Native

React Native – бібліотека, яка призначена для розробки мобільних додатків з відкритим вихідним кодом. Вона є кросплатформенною, отже, дає змогу писати додатки як для мобільної операційної системи Android, так і для IOS, що вирізняє її серед аналогів, які спрямовані на вирішення тієї ж задачі.

Технологія React Native створена на основі бібліотеки для розробки користувацьких інтерфейсів веб-застосунків – React, з якою має багато спільного: філософія розробки, методи проектування застосунку, інструменти для досягнення цілей при програмуванні. Але і відрізняється методами досягнення відображення елементів інтерфейсу: бібліотека React Native не підтримує мову гіпертекстової розмітки – HTML, та мову каскадних стилів – CSS, але надає інструменти та засоби заміщення вище згаданих технологій, особливостями розробки мобільних додатків та, оскільки існує різниця між потребами користувачів веб-застосунків і мобільних, – і функціями, якими ці потреби задовольняються.

Бібліотека React Native – створена і застосовується разом з мовою JavaScript, що надає великий вибір наявних рішень тих чи інших проблем, що виникають при розробці. Крім цього, вона є частиною середовища клієнтського JavaScript, що має одну з найбільших спільнот, що значно спрощує розробку.

Допоміжні бібліотеки React Native

react-native-maps

Бібліотека, яка надає зручний програмний інтерфейс для імплементації вбудованої мапи в мобільний застосунок. Вона була створена на базі MIT.

expo-location

Бібліотека, яка керує дозволом застосунку на використання та відстеження поточної локації користувача

react-native-async-store

Бібліотека, яка дає змогу зберігати довготривалі дані на стороні клієнта.

Socket.IO

Бібліотека для встановлення WebSocket взаємодії із сервером.

Expo

Середовище розробки і управліннями проектами створених за допомогою React Native.

Серед великої кількості можливостей, які надає даний інструмент, необхідно виділити:

- Автоматична збірка проекту

  Expo cli надає легкий інтерфейс для розробки, а саме збирає проект до етапу написання коду та запуску на мобільному прострої
- Зручне управління розробкою

  Expo об’єднує багато етапів створення програмного продукту в один менеджер командної стрічки expo-cli, як-от: встановлення та налаштування, розробка, тестування, інтеграція, публікування проекту.
- Можливість зручного real-time тестування проекту

  Expo Client – це застосунок, який входить в середовище expo та який програміст може використати для швидкого встановлення, оновлення власного застосунку на етапі розробки. Крім іншого він пропонує real-time застосування змін в коді, що значно полегшує роботу.
- Надання великого доповненого функціоналу

  Крім середовища розробки expo містить велику кількість функціоналу, який значно пришвидшує етапи становлення програмного продукту.

Одним із таких є файл конфігурації проекту “app.json”. У ньому наявна велика кількість опцій для налаштовування додатку, крім іншого, велика кількість функцій, які потребують значної кількості коду при використанні інших інструментів розробки, як от Java, реалізовані саме через редагування налаштувань, що містяться в даному файлі, наприклад реалізація вікна “splash” у мобільному додатку.

- Можливість публікувати власний проект

  Expo дає змогу публікувати проект розробнику на репозиторій dev.expo з метою тестування та зручного поширення серед користувачів за допомогою використання Expo Client.

Серверна частина

NodeJS

NodeJS – рушій для запуску JavaScript коду поза межами браузеру, який був створений з ціллю уникнення залежності мови програмування JavaScript від середовища веб-браузера. На даний момент NodeJS – є одним із лідерів у сфері розробки серверних рішень, але також широко використовується при створенні утиліт на основі командної стрічки, допоміжних рішень для розробки, програм для досліджень.

Невід’ємною частиною NodeJS є менеджер пакетів npm, який дозволяє швидко, зручно та гнучко управляти бібліотеками та іншими допоміжними рішеннями під час розробки. Він надає змогу легкої інтеграції серверної частини з компонентами мікросервісною архітектури, як от: СКБД, реляційної, Бази даних для кешу, Брокерів повідомлень та ін.

PostgreSQL та sequelize

PostgreSQL – реляційна СКБД, яка на даний момент є найпоширенішим вибором серед аналогів для програмістів.

Sequelize – ORM (object-relational mapping), інструмент основною ціллю якого є полегшення роботи з базою даних розробнику. Цей інструмент перетворює реляції БД у об’єкти JavaScript, що значно пришвидшує роботу.

Redis

Redis – це нереляційна СКБД, основною задачею якої є управління кешем. Вона стає оптимальним рішенням, при використанні великої кількості записів та однотипного читання даних з БД.

WebSocket та Socket.IO

Оскільки застосунок працює з великою кількістю тимчасових даних та значною частиною роботи з сервером є операції оновлення, а саме активних відвідувачів спорт локації, було прийнято рішення використовувати протокол WebSocket. Він є надбудовою над протоколом HTTP, але найголовнішими відмінностями є те, що він не є stateless, а також є швидшим.

HTTP протокол є stateful, оскільки на кожен запит є незалежним від попереднього та наступного. Це забезпечується відправкою заголовків під час кожного запиту, тобто інформації достатньої, щоб окремо ідентифікувати та обробити кожен запит. Це є перевагою при проектування програмного інтерфейсу, оскільки частини застосунку стають незалежними одна від одної. Що призводить до збільшення можливостей для масштабування та подальшої будови додатку. На відміну від WebSocket, який надсилає усі необхідні заголовки підчас встановлення з’єднання та ініціалізації взаємодії, а на кожному наступному запиті надсилає мінімум інформації. Це дає значний перевагу при розробці великої серверної системи, та проектуванні архітектури, спрямованої на велике навантаження та велику кількість запитів.

WebSocket є швидшим, оскільки на кожному запиті містить лише необхідну інформацію, що зменшує час на обробку запиту, та дає можливість налаштування більш гнучкої взаємодії клієнта та сервера.

Опис реалізації

Розробка додатку складалася з 6 етапів:

1. Створення User Flow
2. Створення дизайну застосунку
3. Проектування моделі даних та мікросервісна архітектура
4. Проектування API
5. Реалізація серверної частини
6. Реалізація клієнтської частини

Модель даних та Мікросервісна Архітектура

На етапі проектування було розроблено модель роботи з даними (див. додаток А). Збереження даних було розділено між реляційною СКБД – PostgreSQL, та базою даних для кешу – Redis.

Перша використовується для збереження довготривалих даних, а саме інформації про користувача та наявні майданчики.

Друга ж у свою чергу виконує завдання збереження тимчасової інформації, оскільки надає значно швидшу в порівнянні з попередньою СКБД роботою з даними.

Поміж іншого реляційна СКБД PostgreSQL використовувалася для забезпечення цілісності даних, а отже використовувалася на етапі валідації даних при записі їх до бази даних для кешу Redis (див. додаток Б).

Було спроектовано архітектуру серверного застосунку на основі патерну проектування систем MVC, model-view-controller. Його принцип полягає у розподіленні застосунку на шари роботи та представлення даних, де model – призначений для взаємодії із СКБД, view – для надання інтерфейсу кінцевому користувачу, controller – для пов’язання model та view.

Розробка серверної частини

Оновлення наявних на майданчику користувачів.

Основною задачею перед серверним застосунком була надання надійного швидкого інтерфейсу для оновлення, тобто додавання та видалення, поточних користувачів на локації. Це досягалося за рахунок спроектованої моделі даних, а саме реалізації розділення даних на «постійні» та «тимчасові»

і використання відповідного сховища, та використання протоколу WebSocket, який не потребує накладних витрат для досягнення поставлених програмних задач.

Розглянемо функціонування бекенду на прикладі додавання користувача до локації поетапно (див. Додаток Б):

1. Встановлення WebSocket зв'язку та операція “add-user”.
   На першому кроці
2. Відповідь сервера клієнту та продовження обробки запиту.

2.1. Оскільки клієнту не обов’язково чекати на результат запиту, сервер відправляє відповідь зі статусом «Успіх»

2.2. У цей час сервер передає обробку запиту далі, використовуючи асинхронний виклик функції, тобто не чекаючи на результат її виконання. На даному кроці внутрішня реалізація JavaScript, а саме Event loop, виступає у ролі брокера повідомлень і є окремою ланкою в архітектурі серверного застусунку.

3. На цьому кроці відбувається перевірка наявності майданчику з таким ідентифікатором у реляційній СКБД PostgreSQL, яка виступає гарантом цілісності даних додатку.
4. І, на кінець, за умови успішної валідації даних відбувається запис нового користувача до бази даних кешу Redis, яка в майбутньому буде використовуватися для читання та отримання усіх користувачів на локацій.

Розробка клієнтської частини

Налаштування і робота з локацією та бібліотекою react-native-maps

Основною частиною користувацького інтерфейсу є мапа. Для вирішення цієї задачі використовувалася бібліотека з відкритим вихідним кодом, яка була створена MIT, react-native-maps. Вона надає широкий набір функцій для взаємодії користувача з представленням локацій.

Спочатку її було додано до відображення (див. Додаток В) для подальшої взаємодії. Разом з нею була додана кнопка для відображення поточного місцезнаходження користувача (див. Додаток Г).

Для взаємодії з поточною локацією мапи використовувався засоби програмування React – хуки, основною задачею яких є надання можливість відстежування оновлення значення певної змінної (див. Додаток Д). Крім цього використовувалися методи onRegionChange, функція, яка виклакається під час кожного оновлення поточної локації мапи та приймає значення нових координат.

Відображення та відстеження поточної геолокації користувача досягалося за використання бібліотеки середовища розробки expo – expo-location. Вона керує поточною локацією користувача, а також дозволом застосунку на використання даних геолокації шляхом запитів до операційної системи. У застосунку бібліотека використовувалася для встановлення початкового місцезнаходження мапи та точки, на яку вона вказує. У хуці React useEffect було створено перевірку на наявність дозволів використання геолокації, а також запит на його надання за негативної відповіді (див. Додаток Д).

Також використовувалося локальне збереження локації користувача у випадку повторного використання застосунку без дозволу використовувати геолокацію або за відсутності такої можливості (див. Додаток Д).

Збереження даних на стороні клієнта

Для досягнення надійності використання застосунку за відсутності доступу до серверної частини або ж мережної взаємодії використовувалося локальне збереження даних в сховищі. Основним інструментом для досягнення цієї мети була бібліотека react-native-async-storage. Це сховище, яке зберігає дані у вигляді ключ-значення, обидва – у стрічковому вигляді.

Оскільки бібліотека надає обмеження щодо типу збереженого значення ключа, а саме лише стрічкового, використовувався тип даних JSON, і його вбудована в мову JavaScript реалізація роботи (функції JSON.stringify для конвертування даних у формат JSON і їх подальшого збереження у сховище та JSON.parse для зведення стрічки JSON в об’єкт мови JavaScript і читання даних зі сховища).

Основною задачею було збереження останнього місця перебування користувача, щоб при наступного можливого використання застосунку без мережної взаємодії або ж доступу до геолокації відображати попередню збережену локацію користувача. Був створений окремий модуль для роботи зі сховищем (див. Додаток Е), в якому відбувався запис та читання стрічок у форматі JSON зі сховища за допомогою AsyncStorage. Модуль надалі використовувався для відновлення останнього місця знаходження користувача на запуску застосунку (див. Додаток Е), а також збереження цієї інформації для подальшого використання.

Налаштування і робота з серверною частиною

Одним з поставлених цілей було досягнення ефективної передачі та отримання даних з сервера. Для цього використовувався протокол мережевої взаємодії WebSocket та інші засоби оптимізації взаємодії.

Налаштування взаємодії з сервером відбувалося за допомогою бібліотеки socket.io. Було створено модуль usePlUsersMap, який містив одноіменний хук. Його задачею було надання актуальної інформації щодо користувачів на локації, та додавання поточного користувача до одної із них та відправки цих даних на сервер (див. Додаток Є). Було проініціалізовано WebSocket з’єднання із сервером та передача поточної локації користувача, щоб отримати лише ті майданчики, які знаходяться поруч з ним (див. Додаток Є). Хук повертає оновлювальне значення структури даних Map, яке містить усі локації, які доступні користувачі, та поточні люди на них, а також функцію

прив’язки користувача до певної локації, яка надсилає оновленні дані усім користувачам, що переглядають цю локацію (див. Додаток Є).

Висновки до розділу 2

Результатом роботи, що описана в даному розділі, є застосунок, який задовольняє поставленим вимогам та вирішує проблема та задачі, які були визначені на етапі аналізу ринку та предметної області. Крім цього, були враховані недоліки та переваги аналогів та створена відповідна до них реалізація. Підсумовуючи, можна виділити поставлені та досягнені цілі: перш за все, застосунок вийшов простим у використанні, він не потребує жодної зайвої взаємодії зі сторони користувача; додаток надає змогу користувачам відстежувати кількість відвідувачів спортивних локацій; застосунок відповідає нормам надійності та його архітектура стійка до значного навантаження.

Висновки

Було дослідження методи та засоби розробки мобільних додатків з використання технології React Native. Крім цього було проаналізовано, спроектовано та розроблено додаток з використанням вище згаданої технології, який мав на меті вирішувати актуальну для багатьох спортсменів проблему – відстежування поточної кількості людей на локаціях для подальшого вибору серед них.

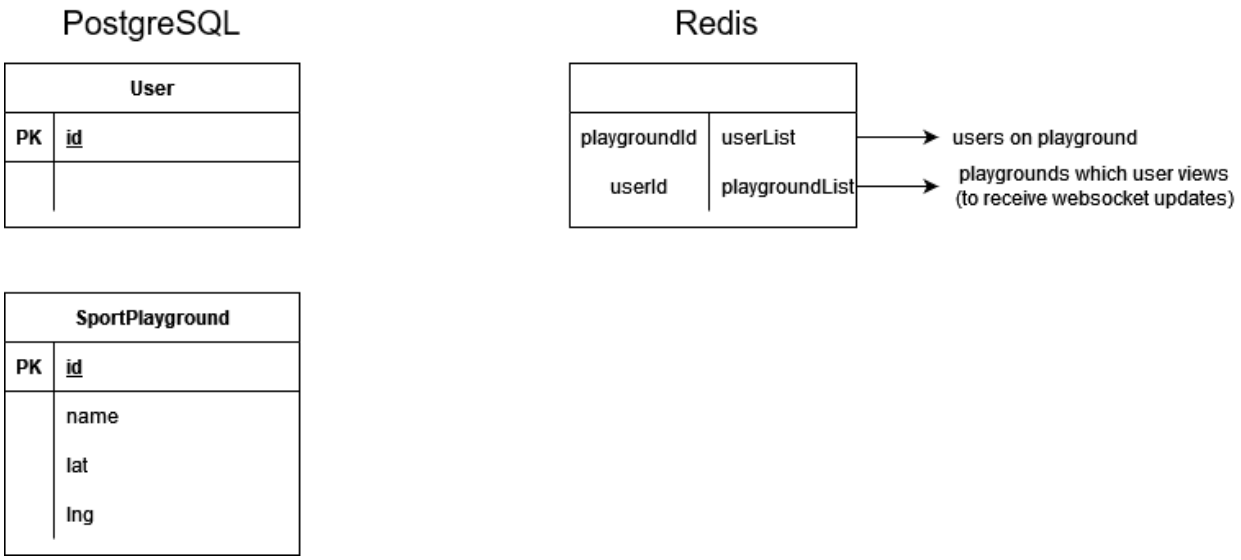
Використані джерела

1. Офіційна документація бібліотеки react-native [Електронний ресурс] / Copyright © 2022 Meta Platforms, Inc.. – Режим доступу до ресурсу: https://reactnative.dev/docs/getting-started
2. Офіційний github-репозиторій та документація бібліотеки react-native-maps [Електронний ресурс] / MIT license 2022. – Режим доступу до ресурсу: https://github.com/react-native-maps/react-native-maps
3. Офіційна документація бібліотеки Socket.IO [Електронний ресурс] / Copyright © 2022 Socket.IO. – Режим доступу до ресурсу: https://socket.io/docs/v4/
4. Офіційна документація Redis [Електронний ресурс] / Redis Ltd. © 2022. – Режим доступу до ресурсу: https://redis.io/docs/
5. Офіційна документація Sequelize [Електронний ресурс] / Copyright © 2022 Sequelize Contributors. – Режим доступу до ресурсу: https://sequelize.org/docs/v6/

Додатки

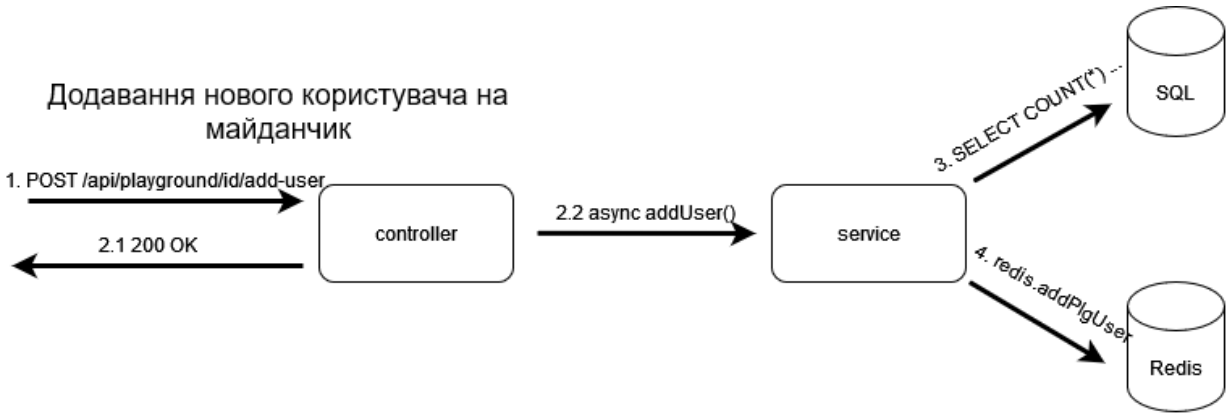
Додаток А

Модель даних



Додаток Б

Архітектура серверної частини



Додаток В

Додавання мапи до інтерфейсу програми.

```
      // додавання хуків
      const [initialRegion, setInitialRegion] = useState(null);
      const [region, setRegion] = useState(null);

// (...)

      // додавання мапи на відображення
      <MapView
            initialRegion={initialRegion}
            region={region}
            style={styles.map}
            showsUserLocation={locationPermission}
      >
      </MapView>

// (...)

      // додавання стилів для відображення мапи
      const styles = StyleSheet.create({
        map: {
          ...StyleSheet.absoluteFillObject,
        },
      });
```

Додаток Г

Додавання кнопки поточної локації користувача

```
      <TouchableOpacity onPress={onLocationPress}
style={styles.myLocationButton}>
            <Image source={locationPermission ? locationActive :
locationInactive}/>
      </TouchableOpacity>

// (...)
      myLocationButton: {
            fontSize: 20,
            position: 'absolute',
            bottom: 20,
            right: 20,
      }
```

Додаток Д

Встановлення початкової локації за допомогою хуку useEffect та бібліотеки expo-location

```
useEffect(() => {
    // отримання попередньо локально збереженої локації
    const getStorageLocation = async () => {
      const locationFromStorage = await customStorage.getLocation();

      if (!locationFromStorage) {
        return;
      }

      setInitialRegion(locationFromStorage);
      setRegion(locationFromStorage);
    };

    (async () => {
      // перевірка на наявність необхідних дозволів
      const { status } = await Location.getForegroundPermissionsAsync();

      // у випадку відсутності дозволу
      if (status === 'granted') {
        let location;
        try {
          // запит на отримання дозволу
          location = await Location.getCurrentPositionAsync();
        } catch (e) {
          // у випадку відмови користувача у наданні дозволу використання
геолокаці
          // отримання попередньо локально збереженої локації
          await getStorageLocation();
          return;
        }
        setLocationPermission(true);
        // отримання поточних координат
        const { coords } = location;
        setLocation(location);
        const region = {
          latitude: coords.latitude,
          longitude: coords.longitude,
          latitudeDelta: 0.0922,
          longitudeDelta: 0.0421,
        };
        // встановлення
        setInitialRegion(region);
        setRegion(region);
        return;
      }
```

```
    // отримання попередньо локально збереженої локації
    await getStorageLocation();
  })();
}, []);
```

Додаток Е

Робота зі сховищем для збереження останньої локації користувача.

Модуль customStorage.js

```
import AsyncStorage from '@react-native-async-storage/async-storage';

export const customStorage = {
  getLocation: async () => await AsyncStorage.getItem('location').then(location
=> JSON.parse(location)),
  setLocation: async (location) => await AsyncStorage.setItem('location',
JSON.stringify(location)),
};
```

Його використання:

```
    // отримання попередньо збереженої у сховищі локації
    const getStorageLocation = async () => {

      // читання локації зі сховища
      const locationFromStorage = await customStorage.getLocation();

      if (!locationFromStorage) {
        return;
      }

      setInitialRegion(locationFromStorage);
      setRegion(locationFromStorage);
    };

  // функція, що викликається при натисненні на кнопку "моя поточна локація"
  const onLocationPress = async () => {
    const { status } = await Location.requestForegroundPermissionsAsync();
    if (status !== 'granted') {
      return;
    }

    // (...)

    // збереження локації у сховищі
    await customStorage.setLocation(region);
    setRegion((prev) => region);
  }
```

Додаток Є

Використання хуку usePlUsersMap

```
import { useEffect, useRef, useState } from "react";
import io from "socket.io";


export const useChat = (location) => {
    const [setPlUsersMap, plUsersMap] = useState(new Map());

    const socketRef = useRef(null);

    useEffect(() => {
        // ініціалізація з'єднання
        socketRef.current = io("/api/socket_connection", {
            query: {}
        });

        // встановлення локації та відстежування майданчиків, які знаходяться в
її межах
        socketRef.current.emit('playgrounds:init', {
            JSON.stringify(location)
        });

        // видалення з'єднання при руйнуванні компонента, в якому
використовується хук
        return () => {
            socketRef.current.disconnect();
        };
    });

    // оновлення користувачів окремого майданчику
    socketRef.current.on('users', (plIdUsersJSON) => {
        const [plId, users] = JSON.parse(plIdUsersJSON);
        setMessages((prevPlUsersMap) => {
          prevPlUsersMap.set(plId, users);
          return prevPlUsersMap;
        });
    });

    // додавання поточного користувача до локації
    const addUserToPlayground = (plId, users) => {
        socketRef.current.emit('users:add', JSON.stringify([plId, users]));
    };

    return { plUsersMap, addUserToPlayground };
};
```