МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
Факультет електроніки та інформаційних технологій

Кафедра комп'ютерних наук

Кваліфікаційна робота бакалавра

ІНФОРМАЦІЙНА СИСТЕМА ПОШУКУ ТА ОРГАНІЗАЦІЇ КОЛЕКТИВНИХ ФІЗИЧНИХ ЗАХОДІВ З ВИКОРИСТАННЯМ ФОРМАТУ СЕРІАЛІЗАЦІЇ ДАНИХ PROTOCOL BUFFERS

Здобувач освіти гр. ІН-81 Назар ВОЙТОВИЧ

Науковий керівник, Олег БЕРЕСТ
кандидат технічних наук, старший викладач

Завідувач кафедри Анатолій ДОВБИШ
доктор технічних наук, професор

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютерних наук

Затверджую_____
Зав. кафедри Довбиш А.С.
"_____" _____2022 р.

ЗАВДАННЯ
до кваліфікаційної роботи бакалавра

Студента 4-го курсу, групи ІН-81 спеціальності 122 -Комп'ютерні науки, денної форми навчання Войтовича Н.В.

Тема: Інформаційна система пошуку та організації колективних фізичних заходів з використанням формату серіалізації даних Protocol Buffers

Затверджена наказом по СумДУ
№_____ від _____2022 р.

Зміст пояснювальної записки: 1) постановка завдання; 2) літературний та аналітичний огляд; 3) вибір програмних засобів для реалізації; 4) розробка користувацького інтерфейсу; 5) огляд теоретичних засад; 6) розробка програмного забезпечення; 7) аналіз результатів виконаної роботи

Дата видачі завдання "_____"_____2022 р.
Керівник випускної роботи _____ Берест О.Б.
Завдання прийняв до виконання _____ Войтович Н.В.

РЕФЕРАТ

Записка: 63 стор., 25 рис., 6 табл., 2 додатки, 16 джерел.

Об’єкт дослідження — система пошуку та організації колективних фізичних заходів яка використовує формат серіалізації даних Protocol Buffers.

Мета роботи — розробка програмного забезпечення з використання бібліотеки Protocol Buffers, завданням якого є надання кінцевим користувачам змоги в межах певного міста знаходити або створювати об’яви з інформацією про колективні фізичні заходи.

Методи дослідження — мови програмування C++ та Objective-C, Git, середовище для розробки програмного забезпечення Xcode, компілятор коду protoc, тестування та життєвий цикл розробки програмного забезпечення.

Результати — розроблено клієнт-серверну систему для пошуку та організації колективних фізичних заходів, яка використовує формат серіалізації даних Protocol Buffers для передачі інформації між її частинами. Клієнт являє собою мобільний додаток розроблений за допомогою мов програмування C++ та Objective-C та розрахований для девайсів на операційній системі iOS. Написана за допомогою C++ серверна частина розрахована на обробку вхідних мережевих запитів від клієнта та, в залежності від їх наповнення, зворотну відправку необхідних відформатованих даних. Для довгострокового зберігання та зчитування інформації використовується система керування базою даних PostgresQL.

РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, КЛІЄНТ-СЕРВЕРНА СИСТЕМА, C++, OBJECTIVE-C, XCODE, PROTOCOL BUFFERS, PROTOC, UI/UX, FIGMA, БАЗИ ДАНИХ, POSTGRESQL.

ЗМІСТ

ВСТУП 6

1. ЛІТЕРАТУРНИЙ ОГЛЯД 7

1.1 Аналіз властивостей iOS додатків для пошуку колективних фізичних заходів 7

1.2 Огляд існуючих аналогів 8

1.2.1 Мобільний додаток «Sport.ly» 8

1.2.2 Мобільний додаток «Sidelyne: Sports Events» 11

1.3 Постановка задачі 13

2. ВИБІР ПРОГРАМНИХ ЗАСОБІВ ДЛЯ РЕАЛІЗАЦІЇ СИСТЕМИ 15

2.1 Вибір формату серіалізації даних 15

2.1.1 Формат серіалізації даних JSON 15

2.1.2 Формат серіалізації даних Protobuf 17

2.2 Вибір системи управління базою даних 18

2.2.1 Реляційна база даних MySQL 19

2.2.2 Реляційна база даних PostgresQL 20

2.3 Вибір мови програмування серверної частини 22

2.3.1 Мова програмування Python 23

2.3.2 Мова програмування Java 25

2.3.3 Мова програмування C++ 27

2.4 Вибір мови програмування iOS додатку 30

2.4.1 Мова програмування Swift 31

2.4.2 Мова програмування Objective-C 33

3. РОЗРОБКА КОРИСТУВАЦЬКОГО ІНТЕРФЕЙСУ IOS ДОДАТКУ 36

3.1 Графічний редактор Figma 36

3.2 Розробка користувацького інтерфейсу iOS додатку 37

4. ТЕОРЕТИЧНІ ЗАСАДИ ДЛЯ ВИРІШЕННЯ ПОСТАВЛЕНОЇ ЗАДАЧІ 40

4.1 Загальна характеристика системи 40

4.2 Роль користувача у системі 40

4.3 Проектування бази даних 42

4.3.1 Таблиці бази даних 43

5. ПРОГРАМНА РЕАЛІЗАЦІЯ 48

5.1 Принцип роботи системи 48

5.2 Навігація всередині додатку 49

ВИСНОВОК 52

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 53

ДОДАТКИ 55

ДОДАТОК А ГРАФІЧНЕ ЗОБРАЖЕННЯ БАЗИ ДАНИХ СИСТЕМИ 56

ДОДАТОК Б ОСНОВНІ КОДИ ПРОГРАМНИХ МОДУЛІВ СИСТЕМИ 57

ВСТУП

Заняття спортом є невід′ємною частиною тижневого розпорядку кожної людини яка піклується про своє здоров’я. Біг, їзда на велосипеді, плавання або футбол, – усе це є лише маленькою частиною переліку доступних шляхів для використання свого часу з користю та задоволенням.

Для усіх спортсменів існують свої ідеальні умови для проведення фізичних заходів, при досягненні яких він відчуває себе найбільш готовим до поставлених перед ним завдань. Проте, їх що разове дотримання для декого може стати нелегкою справою.

Одна із проблем, яка може поставати перед особистістю, перешкоджаючи досяганню поставленої цілі - це пошук партнерів. Саме вона може стати рушієм мотивації, адже для декого ефект соціальної взаємодії під час тренувань є двигуном особистого прогресу.

Важливо зазначити, що процес занять у групі несе у собі також декілька якісних переваг, недоступних у разі індивідуально підходу:

- змагання між членами групи допомагає мотивації утримуватися на високому рівні;
- товариші можуть підказати на можливі помилки у виконанні вправ, або на нові їх версії;
- моральна підтримка від колег[1];

Розвиток мережі інтернет надав нові можливості для об’єднання та організації людей. Ідея роботи полягає в використанні цих можливостей для вирішення проблеми пошуку колективу для занять спортом.

Метою дипломного проекту є розробка системи орієнтованої на ринок мобільних додатків під платформу iOS, завданням якої є допомога людям у пошуку та організації групових тренувань у своїх містах. Користувач матиме змогу зареєструвати персональний аккаунт та за його допомогою переглядати або створювати оголошення спортивних заходів, а також долучатись до них.

1. ЛІТЕРАТУРНИЙ ОГЛЯД

1.1 Аналіз властивостей iOS додатків для пошуку колективних фізичних заходів

Інформація наведена у вступній частині вказує на те, що тема дипломної роботи є актуальною, оскільки мобільний додаток матиме змогу значно зменшити час затрачений на пошук та організацію спортивних заходів з точки зору збору колективу учасників. Години, збережені в такий спосіб, можна витрачати на більш продуктивні речі, у тому числі збільшивши протяжність тренувань.

Готова система являтиме собою інформаційну службу для розміщення спортивно орієнтованих оголошень. Основною цільовою аудиторією є люди зацікавлені в групових заняттях з певних видів спорту, представлених в цих оголошеннях.

З огляду на попередньо вказані аргументи, основними завданнями продукту є:

- надання інтерфейсу створення особистого облікового запису, за допомогою якого буде відбуватись подальша взаємодія з додатком;
- відображення користувачу доступних оголошення про проведення фізичних заходів, які проходять в межах обраного міста;
- надання інформації про поточний статус окремого оголошення (кількість учасників, дата і місце проведення, тощо.);
- надання змоги зареєструвати свою участь у обраних заходах;
- відображення історії усіх заходів, у яких користувач брав участь;

Починаючи з 30 червня 2022 Apple зобов’язує усі додатки які розміщаються на площадці їх продажу «App Store» надати користувачу змогу видалити свій аккаунт, тож цей функціонал також потрібно попередньо передбачити під час проектування системи[2].

1.2 Огляд існуючих аналогів

Для ефективної побудови плану проектування системи, вимог до неї, її властивостей, а також її функцій необхідно провести огляд вже існуючих варіантів реалізації. Це дозволить використати у власній розробці усі їх переваги та уникнути знайдених проблем.

1.2.1 Мобільний додаток «Sport.ly»

Мобільний додаток «Sport.ly» - це програма, яка підходить для людей люблячих займатися спортом або легкою атлетикою. Вони мають можливість приєднуватися або створювати майбутні місцеві зустрічі на основі розташування та рівня навичок. Тобто це путівник для тих, хто прагне знайти місцевих друзів, з якими можна насолоджуватися улюбленими спортивними подіями[3].

Додаток зустрічає користувача екраном реєстрації, на якому пропонується або створити новий аккаунт, або увійти в уже зареєстрований обліковий запис.

Також слід зазначити, що присутній альтернативний спосіб отримати доступ до контенту додатку за рахунок інтеграції з зовнішнім інтерфейсом програмування від Facebook. Це дає змогу поділитися зі «Sport.ly» інформацією про себе надавши доступ до власного облікового запису створеного на платформі Facebook.

Без успішного процесу реєстрації додаток не надає доступу до перегляду внутрішнього контенту який стосується інформації що до спортивних заходів, отже цей крок є необхідним.

Екран реєстрації наведений на рис. 1.1.

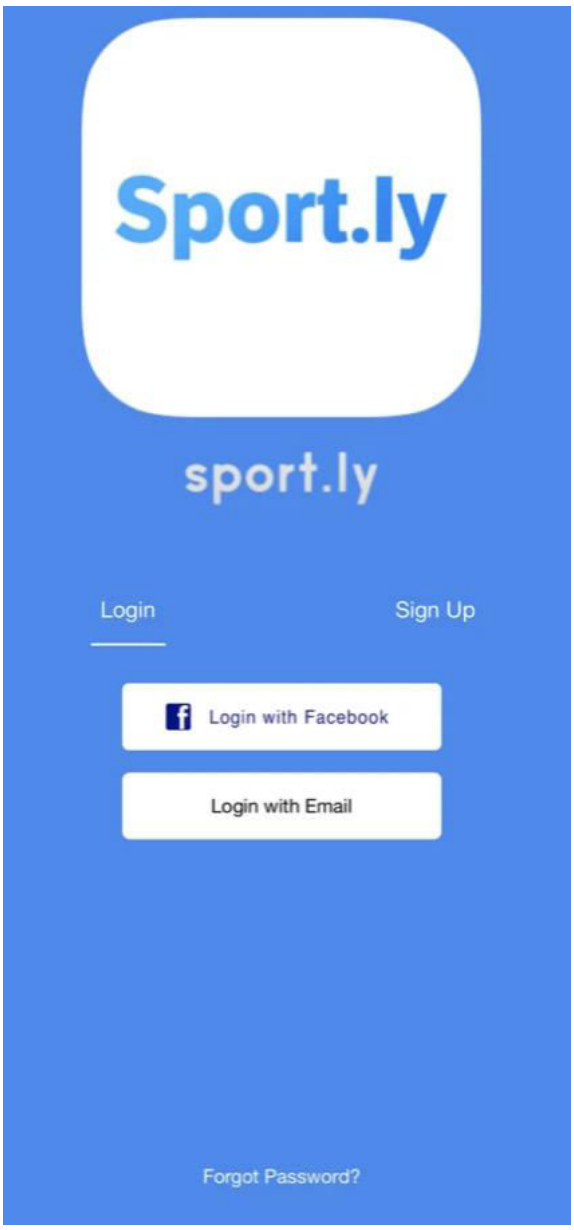

Рисунок 1.1 – Екран реєстрації додатку «Sport.ly»[3]

Пройшовши реєстрацію користувач отримує доступ до усіх можливостей закладених у застосунку, а саме:

- інтерактивний перегляд на мапі усіх доступних спортивних заходів, з подальшою можливістю їх фільтрації;
- створення та редагування власних оголошень спортивних заходів;
- листування з іншими користувачами застосунку;
- перегляд історії спортивних заходів, у яких користувач брав участь;
- пошук спортивних заходів за чіткими критеріями (вид спорту, рівень навичок, час та місце проведення, тощо.)

Екран переліку видів спорту наведений на рис. 1.2

Рисунок 1.2 – Екран переліку видів спорту додатку «Sport.ly»[3]

Додаток має приємний дизайн, функціонально здатний повністю виконувати усі поставлені перед ним задачі, навіть додаючи нові елементи користувацького інтрерфейсу у вигляді приватних повідомлень або розширеної імплементації пошуку за чіткими критеріями.

Проте, дана система не позбавлена недоліків:

- скачування додатку доступне лише для окремого кола країн, до яких не входить Україна, а отже на її території його використання неможливе;
- підтримка застосунку припинилася у 2017 році, і з того часу нових версій застосунку не надходило;
- відсутня локалізація, доступна лише англійська мова;

1.2.2 Мобільний додаток «Sidelyne: Sports Events»

Мобільний додаток «Sidelyne: Sports Events» - це платформа призначена для спортивних клубів і федерацій, метою яких є отримання більшого охоплення, а, отже, більшого доходу[4].

Кожний користувач цього мобільного додатку отримує доступ до інформації про перелік спортивних подій які проходять на певній місцевості. Також присутня можливість чіткого сортування за критеріями, такими як: вид спорту, дата та час проведення заходу, тощо.

Екран сортування наведений на рис.1.3.

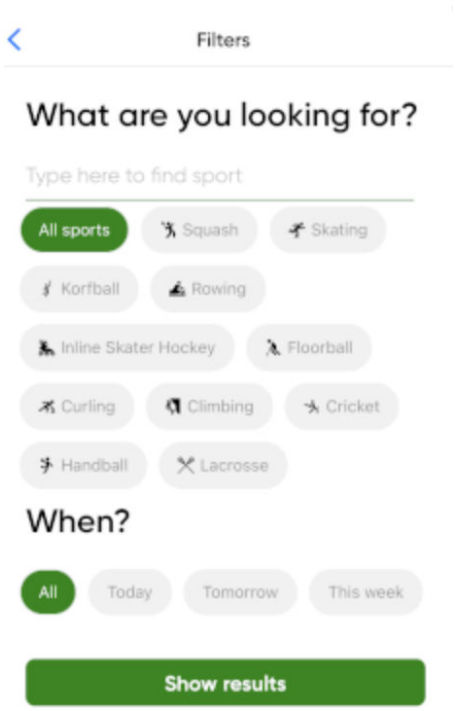

Рисунок 1.3 –Екран сортування додатку «Sidelyne: Sports Events»[4]

Слід також зазначити, що для відображення наявних спортивних заходів у додатку використовується інтерактивна мапа місцевості на якій вказані точки інтересу з позначкою у вигляді символу певного виду спорту, заняття з якого там проходять.

Точка інтересу «Бадмінтон» наведена на рис.1.4.

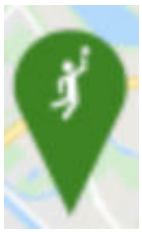

Рисунок 1.4 –Зображення точки інтересу додатку «Sidelyne: Sports Events»[4]

Мапа є інтерактивною, тому будь яку точку можна обрати натиснувши на неї пальцем. В наслідок такої дії з нижньої частини екрану мобільного телефону випливає нове модальне вікно в середині якого відображається детальна інформація про спортивний захід прив’язаний до цього місця та його статус.

Перевагою такої мапи над звичайним списком з можливістю прокручування є те, що вона в короткий термін здатна надати цілісну картину місцезнаходження спортивних заходів. Це, в свою чергу, дозволяє спортсмену обрати найкращий для нього за розташуванням варіант з усіх присутніх на даний час.

Наявність функції фільтрування унеможливлює ситуації коли екран стає загромадженим великою кількістю інформації, тобто точками інтересу, оскільки користувач має змогу відокремити від решти лише ті події, які його цікавлять.

Екран з пошуком, інтерактивною мапою та обраною на ній точкою інтересу «Хокей» наведено на рис.1.5.

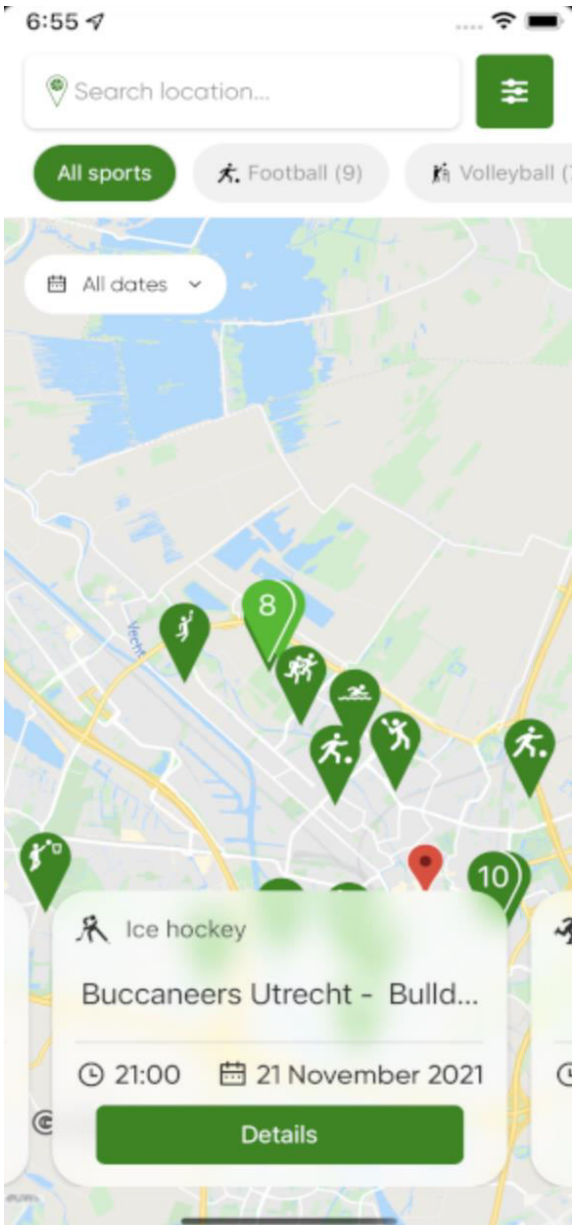

Рисунок 1.5 –Інтерактивна мапа додатку «Sidelyne: Sports Events»[4]

До недоліків цього додатку можна віднести те, що він недоступний до скачування на території України, а, отже, його використання в її межах є неможливе. Також відсутня локалізація Української мови.

1.3 Постановка задачі

В кінцевому результаті очікується отримати завершену та протестовану клієнт-серверну систему. Додаток для мобільної платформи iOS виконуватиме роль клієнту який зможе надати користувачу увесь набір функціоналу котрий

окреслений у вступній частині. Серверна частина являтиме собою набір сервісів до яких буде звертатися додаток та отримувати за запитом необхідну інформацію.

Для отримання працездатної версії програми в найкоротший термін її побудова буде проходити у чіткій послідовності за розбитим на пункти планом:

1. необхідно погодити набір технологій які будуть застосовані у розробці кожного з компонентів програми: клієнта, серверної частини та бази даних. Оскільки для кожної задачі існує певний їх набір, то робити вибір з нього потрібно виходячи з усіх наявних факторів: час затрачений на розробку, ціна використання, простота опановування, тощо;

2. необхідно обрати інструменти для розробки дизайну проекту. За їх допомогою потрібно погодити загальний стиль додатку і розробити максимально інтуїтивний та простий у використанні користувацький інтерфейс;

3. необхідно розробити архітектуру майбутнього клієнтського мобільного додатка та усіх сервісів серверної частини. Також на цьому етапі повинен з’явитися фінальний варіант вигляду структури з обраною в пункті 1 базою даних, з якою в майбутньому буде контактувати клієнт за посередництвом сервісів;

4. приступити до розробки системи починаючи з налаштування бази даних та розробкою усіх необхідних сервісів серверної частини, а закінчуючи мобільним додатком під платформу iOS;

5. провести загальне тестування отриманого продукту з подальшим виправлення усіх знайдених в процесі помилок. Зробити необхідні висновки щодо результатів виконаної роботи;

У ході успішного виконання усіх пунктів плану будуть розроблені, протестовані та готові до реалізації на відповідних площадках для розповсюдження: серверна частина і мобільний додаток під платформу iOS для пошуку та організації колективних фізичних заходів;

2. ВИБІР ПРОГРАМНИХ ЗАСОБІВ ДЛЯ РЕАЛІЗАЦІЇ СИСТЕМИ

Слідуючи за раніше погодженим планом розробки системи, першим пунктом необхідно погодити набір технологій та програмних засобів, які будуть використовуватися в процесі написання програми. Їх вибір повинен бути заснований з урахуванням усіх переваг та недоліків, які вони з собою несуть, наприклад: забезпечення максимальної продуктивності, рівень навантаження на обладнання, підтримка зі сторони розробника, можливість подальшого розширення функціоналу, тощо.

2.1 Вибір формату серіалізації даних

Для того, щоб клієнтська частина змогла зрозуміти відповідь на надісланий запит до сервера, вона повинна підтримувати той самий формат серіалізації даних, в якому його до неї було відіслано. Якщо ж цього не відбувається, то для мобільного додатку ця відповідь буде виглядати як незрозумілий набір байтів, і їх подальше доцільне використання буде неможливим.

Кожний формат має як свої недоліки, так і плюси, тому їх потрібно обирати відповідно до по потреб та специфікацій проекту.

2.1.1 Формат серіалізації даних JSON

JSON – JavaScript Object Notation – це найпопулярніший на даний момент формат передачі даних, який дозволяє поєднувати великі обсяги інформації у одному, зрозумілому для прочитання людиною, текстовому блоці.

Усі дані представлені у вигляді «ключ-значення» та розділені за допомогою символу «:». Зазвичай, ключем є власна назва, яка коротко описує контент, який розміщується за ним. Декілька пар полів «ключ-значення»,

розміщені між символами «{» та «}», утворюють певний об’єкт, який теж повинен розміщуватися за власним ключем. Якщо символи «{» та «}» замінити на «[» та «]» відповідно, то це означитиме, що дані всередині утворюють масив[5].

Приклад даних представлених у форматі JSON наведено на рис.2.1.

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [],
  "spouse": null
}
```

Рисунок 2.1 –Дані представлені у форматі JSON[6]

При порівнянні з конкурентами у форматі серіалізації даних JSON можна виділити декілька переваг:

1. підтримується майже всіма наявними на даний момент мовами програмування, отже майже кожен програміст зможе його використовувати у роботі;

2. формат може бути легко прочитаний людиною;

3. популярний формат, а отже має велику кількість програмістів, від яких можна отримати технічну підтримку. Також має велику кількість інформаційних пов’язаних ресурсів у відкритому доступі.

Проте, він не позбавлений і недоліків:

1. підтримує обмежений набір типів даних які не можуть бути розширені (рядок, ціле число, дробове число, тощо.);

2. не підтримує схеми, а отже розробник не зможе декодувати отримані дані, якщо не буде мати прикладу відповіді;

3. не найшвидший процес серіалізації та десеріалізації даних;

Швидкодія системи є вагомим важелем при виборі інструментів розробки, тому фактор тривалості процесу серіалізації даних є вагомим при виборі формату.

2.1.2 Формат серіалізації даних Protobuf

Protobuf – Protocol Buffers – це нейтральний до мови програмування або платформи механізм для серіалізації структурованих даних прямим і зворотнім шляхами. Він є подібним до JSON, за винятком того, що фінальний вигляд даних займає менше місця, а на їх кодування та декодування витрачається не так багато часу.

Protocol Buffers являють собою комбінацію мови програмування Proto та компілятору який бере за основу схему у вигляді окремих повідомлень описаних цією мовою у файлах з розширенням «.proto» і генерує специфічний для кожної платформи код.[7]

Приклад схеми повідомлення наведено на рис.2.2.

```
message Person {
  optional string name = 1;
  optional int32 id = 2;
  optional string email = 3;
}
```

Рисунок 2.2 –Схема повідомлення написана на мові Proto [7]

Маючи таку чітку схему та заздалегідь згенерований компілятором код для його обробки можливість помилки при роботі з парсингом даних зводиться до мінімуму.

Маючи бінарний формат представлення даних ми втрачаємо можливість читати отриману інформацію без попередньої її обробки та схеми Proto. Проте, натомість ми отримуємо набагато більшу швидкість обробки отриманих об’єктів програмою, що при великих об’ємах значно зменшує час очікування користувача, тому покращує його загальні враження від користування додатком.

Маючи компілятор та схеми «.proto» з’являється можливість підтримувати взаємозв’язок формату даних між різними сегментами системи незалежно від того, на якій мові вони написані, що робить подальший їх вибір більш незалежним.

Підсумовуючи усе написане вище, форматом серіалізації даних можна обрати Protocol Buffers, оскільки його переваги у швидкодії та одночасній підтримці декількох мов програмування за допомогою схем позитивно вплинуть на можливість подальшої підтримки кодової бази системи та її продуктивності в цілому.

2.2 Вибір системи управління базою даних

У процесі експлуатації системі необхідно місце де буде зберігатись інформація яка буде доступна усім її частинам. Дані про користувача, наявні

спортивні оголошення та історія їх відвідування, - усе це повинно бути записано до бази даних та, за потреби, зчитаними і переданими користувачу iOS додатку.

Зазвичай клієнт не звертається напряму до бази даних, у цьому йому допомагають сервіси серверної частини, які виступають посередниками між ними і в процесі можуть обробляти дані у необхідний кінцевій стороні формат.

Бази даних розробляються відповідно до потреб та властивостей певної архітектури системи, а тому в даному випадку вона планується як «клієнт-серверна», то і вибір потрібно робити враховуючи необхідність подальшої підтримки усіх сервісів.

Обирати фінальний варіант необхідно з зазначенням таких ключових факторів, як:

- за типом та характером інформації, для збереження якої вони розроблялись;
- за способом в який вони зберігають інформацію на її носіях;
- за структурою організації збереженої інформації;
- за рівнем швидкодії та навантаження на обладнання при великих об’ємах даних;

Оскільки в майбутньому система планує розширятися, то найкращим варіантом будуть реляційні моделі, оскільки для збільшення їх пропускної здатності не потрібно встановлювати додаткові сервери, достатньо просто покращити апаратні здатності вже існуючого.

Основним стандартом мов виконування запитів в реляційних моделях є SQL, можливості якого здатні повністю покрити усі вимого проекту, тому його використання буде доречним.

2.2.1 Реляційна база даних MySQL

MySQL – це найбільш популярна база даних з відкритою кодовою базою, розробка якої займається компанія Oracle.

MySQL не має повної підтримки стандарту мови запитів SQL. Це було зроблено для того, щоб максимально підвищити швидкодію та зручність використання розробником. Проте, при потребі, є можливість розширення з додавання необхідного функціоналу який не шкодить стандарту.

При виконанні запиту MySQL загружає усю відповідь у оперативну пам'ять клієнта, що у випадку великих об’ємів не завжди позитивно впливає на продуктивність системи[8].

Приклад виконання запиту у клієнті MySQL Workbench до бази даних MySQL наведено на рис.2.3.

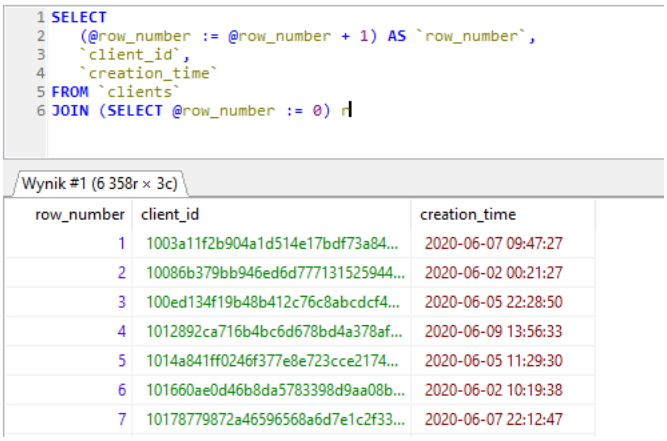

Рисунок 2.3 –Виконання запиту у клієнті MySQL Workbench [9]

Вагомим плюсом MySQL є те, що він підтримує роботу з декількома механізмами зберігання даних на вибір. Ці механізми ніяк не впливають на синтаксис запитів, лише змінюють поведінку запису даних на диск та їх зчитування з нього. Мається підтримка таких механізмів як: MEMORY, Berkley DB, InnoDB та MyISAM.

2.2.2 Реляційна база даних PostgresQL

PostgresQL – це, за переконанням авторів, найбільш розвинута база даних з відкритою кодовою базою.

PostgresQL повністю підтримує стандарт мови запитів SQL, включаючи найсучасніші версії, що робить його набагато складнішим у розробці ніж згаданий вище MySQL.

Вибору між механізмами зберігання даних у випадку з PostgresQL немає, оскільки для роботи з таблицями він може використовувати лише стандартний «storage engine». Самі таблиці, в свою чергу, організовані у вигляді об’єктів, дії з якими виконуються за допомогою об’єктно орієнтованих функцій[8].

Приклад виконання запиту до бази даних PostgresQL наведено на рис.2.4.

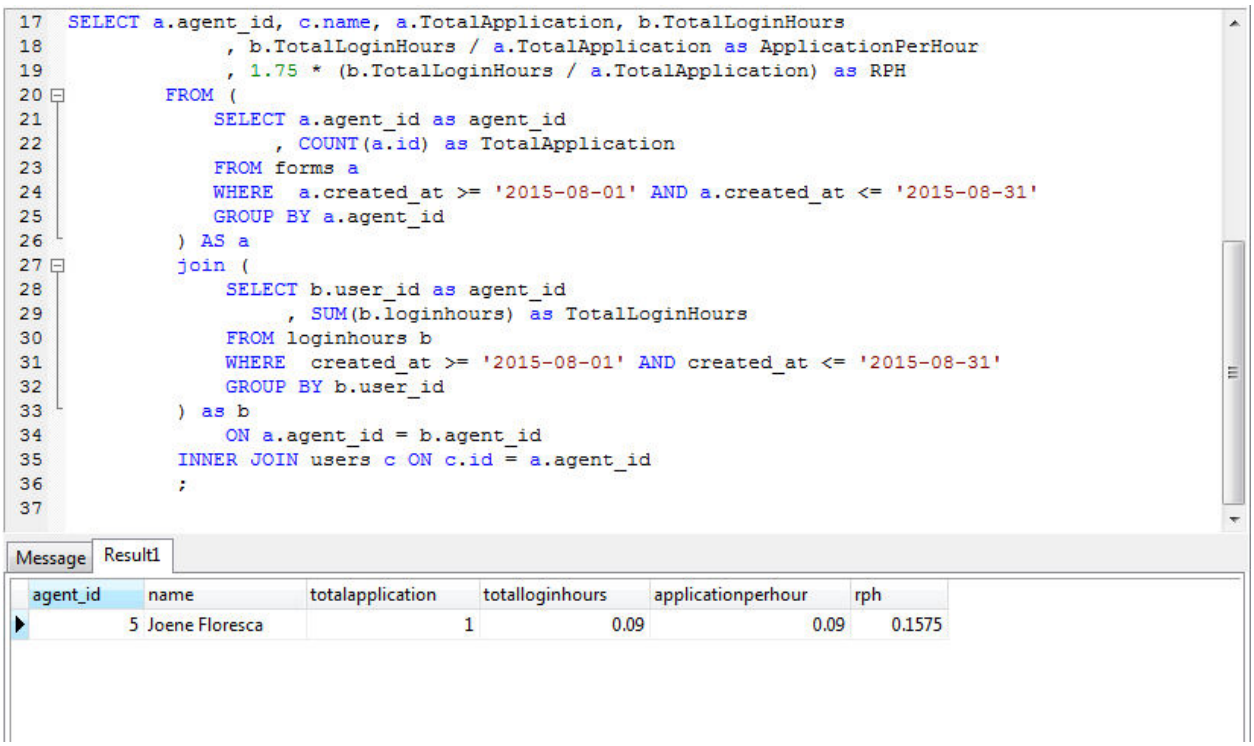

Рисунок 2.4 –Виконання запиту до бази даних PostgresQL, та відповідь на нього

Вагомим плюсом в роботі з PostgresQL є його робота з пам’яттю. На відміну від MySQL він не переміщує дані до оперативної пам’яті, а лише повертає покажчик на те місце де вони зберігаються у самій базі даних.

Також мається підтримка регулярних виразів, рекурсивних запитів та наслідування.

Початково, на відміну від MySQL, PostgresQL не був орієнтований на максимально можливу швидкодію, оскільки в його основні задачі входила підтримка стандартів. Проте, з годом, ця ситуація змінилася і він почав навіть випереджати свого конкурента по цьому параметру.

Цей факт, в купі з іншими перевагами, такими як можливість маніпуляції даними, не переміщуючи їх з диска, дозволяє обрати PostgresQL як базу даних для розроблюємої системи.

2.3 Вибір мови програмування серверної частини

Вибір мови програмування серверної частини є основоположним етапом початку її розробки. Від результату будуть залежати усі аспекти фінального продукту, а саме:

- швидкість розробки та тестування;
- доступність модулів, які є можливість підключити під час розробки програми та позбавити розробника від необхідності їх написання власноруч;
- швидкість роботи створеного програмного забезпечення;
- складність подальшої підтримки та розширення функціоналу продукту;

Важливо, щоб дана мова програмування мала у вільному доступі інструменти для взаємодії з раніше обраною системою управління базою даних PostgresQL, оскільки їх самостійне написання є достатньо складною задачею і займе великий проміжок часу.

Також, важливо щоб розробник мав досвід роботи з мовою програмування, оскільки кожна з них має свої особливості та характеристики починаючи з синтаксису, і закінчуючи рівнем підтримки різноманітних

інструментів розробки, таких як формати серіалізації даних. Для опанування всіх аспектів також витрачається час, тому, за можливістю, потрібно обрати саме той варіант, на налаштування та початок роботи з яким піде найменший проміжок часу. Ще одним доводом до цього аргументу може бути той факт, що чим більше навичок роботи з певною мовою програмування має фахівець, тим більшим є шанс, що рівень знаходження помилок у фінальному коді продукту буде якомога меншим.

У мовах програмування розрахованих на веб розробку існує розподілення на клієнтські, тобто ті, код яких виконується на стороні клієнта, та серверні, код яких виконується відповідно на стороні сервера. Оскільки дана дипломна робота не включає в себе розробку веб версії клієнту, її роль візьме на себе мобільний iOS додаток, то потрібно робити вибір лише з серверних варіантів.

2.3.1 Мова програмування Python

Python – це мова програмування високого рівня яка використовує динамічний вид типізації. Це означає, що будь яка змінна не буде мати чіткого типу до того моменту, поки їй не присвоять якесь конкретне значення.

Такий підхід має в собі як позитивні, так і негативні якості. Багато розробників вважають, що розробка на динамічних мовах програмування в середньому виявляється швидшою та для декого може бути навіть приємнішою. До негативних сторін відносять більшу вірогідність вияву помилок вже на стадії роботи програми, тому для їх уникнення доводиться писати більше тестів, що несе втрати в часі.

Ця мова програмування вважається дуже легкою в освоєнні. На даний момент існує велика кількість інформативних ресурсів та курсів які допомагають ним оволодіти, проте на це все одно потрібно використати час для досконалого оволодіння базовими аспектами.

На даний момент за допомогою спільноти були розроблені велика кількість фреймворків які можуть бути використані для вирішення проблем від розроблення веб програм до їх тестування.

Приклад коду написаного на мові програмування Python наведено на рис.2.5.

```
from bs4 import BeautifulSoup
import requests
import os, os.path, csv

listingurl = "http://www.espn.com/college-sports/football/recruiting/databaseresults/_/sportid/24/class/2006/sc

response = requests.get(listingurl)
soup = BeautifulSoup(response.text, "html.parser")

listings = []
for rows in soup.find_all("tr"):
    if ("oddrow" in rows["class"]) or ("evenrow" in rows["class"]):

        name = rows.find("div", class_="name").a.get_text()
        hometown = rows.find_all("td")[1].get_text()
        school = hometown[hometown.find(",")+4:]
        city = hometown[:hometown.find(",")+4]
        position = rows.find_all("td")[2].get_text()
        grade = rows.find_all("td")[4].get_text()

        listings.append([name, school, city, position, grade])

with open("footballers.csv", 'a', encoding='utf-8') as toWrite:
    writer = csv.writer(toWrite)
    writer.writerows(listings)

print("ESPN College Football listings fetched.")
```

Рисунок 2.5 –Приклад коду написаного на мові програмування Python

Найбільш популярними фреймворками для написання серверної частини систем є:

- Django;
- Flask;
- Bottle;

Для покриття написано коду тестами та їх проведення також існує декілька готових рішень:

- Pytest;
- Mixer;
- Faker;

Наведені вище списки фреймворків для вирішення задач не є повними, вони складають лише частину найбільш популярних з них, і якщо з якоїсь причини в процесі виясниться що вони не підходять для задач конкретної системи, їм можна буде спробувати знайти заміну з решти не вказаних у цій роботі.

Головним з мінусів мови програмування Python вважається низька швидкість роботи в порівнянні з конкурентами яка закладена в основі проектування самої мови і не може бути виправлена ніякими фреймворками, патчами, тощо.

Роблячи фінальний висновок щодо вибору Python в якості мови програмування серверної частини системи необхідно зауважити, що вона являється дійсно гарним продуктом у своїй ніші. Проте, кілька ключових недоліків, як, наприклад, швидкість роботи, не дають їй встати на першу сходинку у рейтингу найкращих мов для вирішення поставленої перед даним проектом проблеми, оскільки продуктивність на даному проміжку проектування являється необхідною складовою, від якої неможливо відмовитись.

2.3.2 Мова програмування Java

Java – це об’єктно орієнтована мова програмування з статичним видом типізації, тобто кожній змінній потрібно вручну задати її тип.

Основною задачею при розробці цієї мови було виконання принципу «Write once, run anywhere» або «напиши один раз, виконуй будь-де». Тобто один і той самий код можна використовувати ну будь якій платформі якщо на ній заздалегідь налаштоване середовище для його виконання – Java Runtime Environment (JRE)[10].

Цей принцип виконується за допомогою переводу усього написаного програмістом контенту програми в байт код за допомогою компілятора. Далі

до роботи приступає віртуальна машина Java Virtual Machine (JVM) яка не залежить від середовища системи і виконує згенерований байт код[10].

Приклад коду написаного на мові програмування Java наведено на рис.2.6.

```
import java.io.IOException;

public class TomcatEmbedded {

    private static final String EMPTY = "";

    public static void main(String... args)
        throws Exception {
        File baseFolder = new File(System.getProperty("user.dir"));
        File appsFolder = new File(baseFolder,  child: "apps");

        Tomcat tomcat = new Tomcat();
        tomcat.setBaseDir(baseFolder.getAbsolutePath());
        tomcat.setPort(8080);
        tomcat.getHost().setAppBase(appsFolder.getAbsolutePath());

        // Call the connector to create the default connector.
        tomcat.getConnector();

        tomcat.addWebapp(EMPTY,  docBase: ".");
        Wrapper wrapper = tomcat.addServlet(EMPTY,  servletName: "hello", new HelloServlet());
        wrapper.setLoadOnStartup(1);
        wrapper.addMapping( s: "/*");

        tomcat.start();
        tomcat.getServer().await();
    }
```

Рисунок 2.6 –Приклад коду написаного на мові програмування Java

Необхідно також згадати про механізм вбудований для управління пам’яттю під назвою «garbage collector» або «збирач сміття». Кожний створений об’єкт в середині Java Runtime Environment має вбудований рахівник створених на нього посилань. Якщо якомусь об’єкту «А» необхідно для роботи існування об’єкту «Б» він створює на нього посилання, що у свою чергу збільшує лічильник посилань об’єкту «Б». Якщо ж необхідність у цьому об’єкті перестає існувати, або ж сам об’єкт «А» видаляється з пам’яті то це, в свою чергу, зменшує лічильник посилань об’єкт «Б». Досягання лічильником нульової відмітки говорить середовищу що цей об’єкт більше не використовується і його можна безпечно видаляти задля звільнення місця на майбутні операції.

Також при роботі з Java велику роль має досвід програміста, оскільки без детального знання усіх аспектів мови програмування кінцевий результат може значно поступатись у характеристиках конкурентам.

До основних недоліків мови програмування Java відносять більше споживання ресурсів обладнання в порівнянні з деякими конкурентами. Виконання команд виконується поверх Java Virtual Machine, на що використовуються додаткові ресурси. Більше навантаження означає необхідність мати більш потужне обладнання на яке потрібно витрати більшу суму коштів.

Ще одним недоліком є неможливість підтримки низького рівня програмування, тому програма не матиме можливості доступу до системного рівня ресурсів.

Роблячи фінальний висновок щодо вибору Java варто зазначити, що вона є розвинутою, стабільною, зрілою та зручною у використанні мовою програмування яка розвивається вже не перше десятиліття. Проте, вона не позбавлена й ключових недоліків. Необхідність використання програми з допомогою віртуальної машини, в порівнянні з тими, які були скомпільовані одразу, призводить до значних втрат у характеристиках які стосуються швидкодії. Через неможливість отримання доступу до низького рівня системи виникає необхідність у використанні додаткових модулів написаних на більш пристосованих для цих задач мовах програмування, наприклад C++. Для стандартних задач існують стандартні бібліотеки, проте вони можуть бути залежними від платформи і їх використання позбавляє програму основної переваги [11].

Підсумовуючи все написане вище можна зробити висновок що мова програмування Java в повній мірі не відповідає усім потребам та вимогам проекту, тому її використання не буде доречним.

2.3.3 Мова програмування C++

C++ - це універсальна мова програмування яка використовує статичну типізацію даних по такому ж принципу, як це було описано на прикладі Java. Універсальною вона називається по тій причині, що опанувавши її розробник отримує змогу писати програми для вирішення різноманітних задач під різні платформи: Декілька варіантів використання C++:

- написання десктопних програм для операційних систем таких як Windows або Linux;
- розробка комп'ютерних ігор для тих же операційних систем;
- розробка серверної частини сайтів (back-end);

Дана мова програмування вважається кросплатформенною, оскільки існує велика кількість компіляторів його коду,наприклад: clang, g++, gcc, тощо. Більшість з них має підтримку різноманітних операційних систем, а це означає, що один і той самий написаний код можна скомпілювати для роботи в різних середовищах.

Вважається, що C++ має один з найбільших так званих «порогів входження», тобто часу на його опанування для отримання достатнього рівня навичок роботи з ним, в середньому, витрачається більш ніж для його конкурентів або аналогів. Навіть проста операція в інших мовах програмування, така як, наприклад, приведення типу рядка до типу цілого числа і виведення його в консоль для новачка може виявитись досить нетривіальною задачею з необхідністю написання великої кількості коду, що у випадку інших мов може виявитись виявлятись набагато легше.

Компілятори C++ в більшості випадків займають перші сходинку чартів в яких мова йдеться про швидкість компіляції та роботи програм. За допомогою статичного формату типізації даних, компіляторам не треба витрачати час на визначення їх типу самостійно. Другий пункт досягається за рахунок різних типів оптимізації, в тому числі, при роботі з пам'яттю машини.

Мова проектується та розробляється таким чином, щоб підтримувати можливість комфортної роботи з багатьма стилями програмування, до яких

відносяться: процедурне програмування, об’єктно-орієнтоване програмування, загальне програмування, тощо[12].

Приклад коду написаного на мові програмування C++ наведено на рис.2.7.

```
class APIRequestHandler : public HTTPRequestHandler {
public:
    void handleRequest(HTTPServerRequest& request,
        HTTPServerResponse& response) {
        Application& app = Application::instance();

        HTMLForm form(request, request.stream());
```

Рисунок 2.7 –Приклад коду написаного на мові програмування C++

По аналогії з багатьма іншими мовами програмування, C++ має велику спільноту програмістів які створюють та викладають у відкритий доступ різноманітні фреймворки та бібліотеки з допомого яких можна вирішувати певні задачі.

Приклад фреймворків для написання htpp серверів для мови програмування C++:

- Crow;
- Httpp;
- Oat++;
- Drogon;

Всі вище наведені приклади постійно оновлюються та мають певно спільноту користувачів до яких можна звернутись у разі виникнення питань щодо їх використання.

Враховуючи раніше обрану систему контролю бази даних PostgresQL, C++ необхідно також мати інструменти для роботи з нею, які також є у відкритому доступі у вигляді фреймворків або бібліотек. Для прикладу можливо навести наступні варіанти:

- taopq;
- libpq;

Враховуючи усі описані факти, такі як рівень швидкодії та часу який витрачається на процес компіляції програм, рівень оптимізації та підтримки різних парадигм програмування, кросплатформенність з можливістю обрати найбільш зручний та доречний компілятор для коду, а також наявність великої кількості різноманітних фреймворків та бібліотек необхідних для успішного написання даного дипломного проекту, можна зробити висновок, що мова програмування C++ є достойним кандидатом на вибір мови програмування для написання серверної частини системи і у повній мірі відповідає усім її потребам.

2.4 Вибір мови програмування iOS додатку

Операційна система iOS, яка встановлена на кожному iPhone, підтримує обмежену кількість мов програмування, а саме ті, які проектувались та розроблялись компанією Apple спеціально для вирішення проблеми розробки програмного забезпечення для їх власної платформи.

На даней момент існує всього два варіанти вибору:

- Objective-C;
- Swift;

Вони обидва мають зовсім різний синтаксис та специфіку використання. Проте, на перший погляд, зовсім простий вибір з двох варіантів у випадку неправильного рішення може спричинити значні обмеження в можливостях.

У випадку, якщо додаток стане успішним і виникне необхідність у розширенні функціоналу, то потрібно бути заздалегідь готовим до цього та мати чіткий план дій на цей випадок. Це означає, що є надзвичайно важливим попереднє та довгострокове визначення планів дій та вибір тої мови програмування, яка дозволить найлегше їх виконання.

Приклад iOS додатку наведено на рис.2.8.

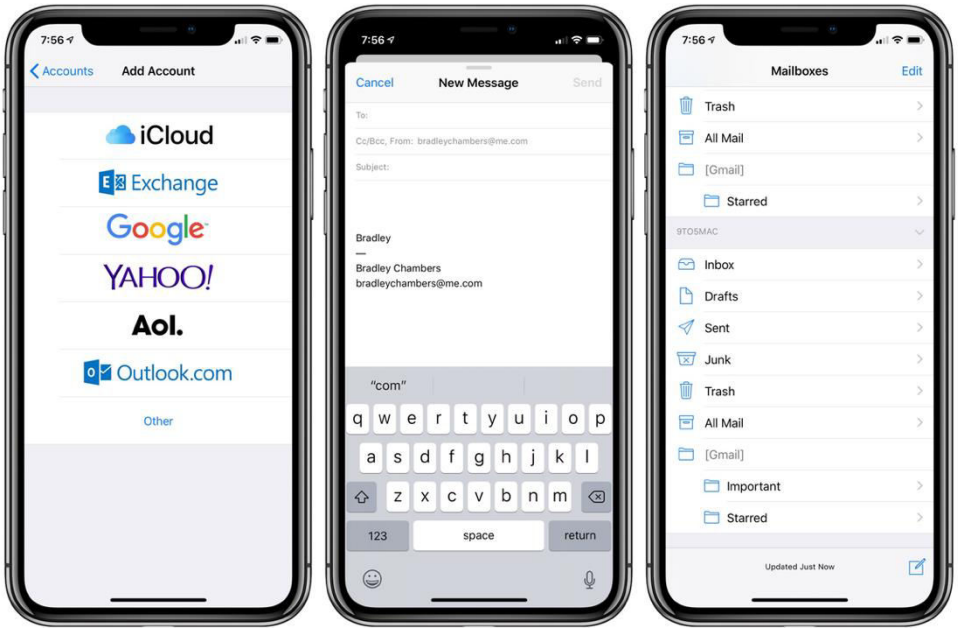

Рисунок 2.8 –iPhone з увімкненим стандартним додатком «Mail»

Цей вибір є надзвичайно важливим та вирішальним, оскільки для того щоб люди хотіли завантажувати та використовувати ваш додаток необхідно надавати максимально можливу його якість.

2.4.1 Мова програмування Swift

Swift – це молода мова програмування яка була представлена у 2014 році. Вона має простий синтаксис, що робить її дуже легкою для читання та легкою в освоєнні.

Розробники, для зручності, додали до неї автоматичний механізм підрахунку посилань на об’єкти, що являє собою «garbage collector», подібний

до якого наявний у мові програмування Java. Об’єкт вважається потрібним до тих пір, поки на нього існує хоч одне посилання і лічильник не опускається до позначки нуля. Як тільки це стається, то цей об’єкт починає вважатися непотрібним і при першій нагоді буде видалений з пам’яті задля збільшення ефективності програми.

Проте, такий механізм несе в собі і небажані недоліки. Існує варіант коли два об’єкти утримують посилання один на одного, що називається циклічною залежністю. У такому випадку вони ніколи не вийдуть з пам’яті пристрою до поки програма не буде зупинена.

Також був доданий такий механізм зручності використання як «optional». Він вирішує проблему покажчиків які посилаються на неіснуючий об’єкт. Для того, щоб позначити для компілятора що значення даної змінної не буде гарантовано присутнє у певний момент часу, потрібно додати в кінці типу змінної додатковий символ «?», наприклад «Int?».

Приклад коду написаного на мові програмування Swift наведено на рис.2.9.

```
import UIKit

class ViewController: UIViewController {

    @IBOutlet weak var myLabel: UILabel!

    @IBAction func myButton(sender: AnyObject) {

        myLabel.text = "It worked!"
    }
    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.

        print("Hello world!")
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }


}
```

Рисунок 2.9 – Приклад коду написаного на мові програмування Swift

Swift є мовою програмування з відкритою кодовою базою, тож при нагоді є можливість зрозуміти з середини принципи його роботи та покращити власні навички роботи з нею.

Важливо зазначити, що Swift має можливість використовувати код мови програмування Objective-C. Тобто будь який функціонал написаний на Objective-C можна передати компілятору разом з функціоналом написаним на Swift, котрий через доступний інтерфейс зможе його використовувати через власний синтаксис. Це можливо зробити тільки з Objective-C. З іншими мовами програмування Swift сумісності не має.

Головною проблемою Swift є відсутність оберненої сумісності з минулими його версіями. Тобто компілятор Swift версії три не зможе успішно зібрати проект написаний на версії під номером два. Залишити все як є і продовжувати використовувати старіші його версії також неможливо, оскільки новіші версії програмного забезпечення iOS потребують тих змін, які поступово вносяться в Swift. Сам процес переходу кодової бази, якщо вона досить велика, не є тривіальним процесом та у ході якого з’являється можливість появи нових помилок присутність яких може залишитись непоміченою у процесі розробки[13].

2.4.2 Мова програмування Objective-C

Objective-C – це мова програмування яка була розроблена компанією «Apple» більше тридцяти років туму назад. Використовує статичний формат типізації даних. Усі ці роки компанія сумлінно розвивала цю мову програмування, покращувала її та насичувала новим функціоналом. За статистикою, до моменту появи Swift у 2014 році, майже 90% усіх розробників додатків для платформи iOS використовували Objective-C в якості мови програмування[14].

В результаті роботи команди розробників «Apple» було створено та виведено у відкритий доступ велику кількість документації та курсів, проглянувши які програміст може отримати знання та навички необхідні для написання якісних iOS додатків.

Код написаний на мові програмування Objective-C здатний використовувати C та C++. Це означає, що величезна кодова база цих мов, усі фреймворки та бібліотеки, розробник має можливість використати при написанні свого додатку. А оскільки було вирішено, що написання серверної частини проекту буде проводитись з допомогою мови програмування C++, то отримується можливість використання спільної кодової бази для вирішення однакових проблем присутніх в обох частинах системи.

Приклад коду написаного на мові програмування Objective-C наведено на рис.2.10.

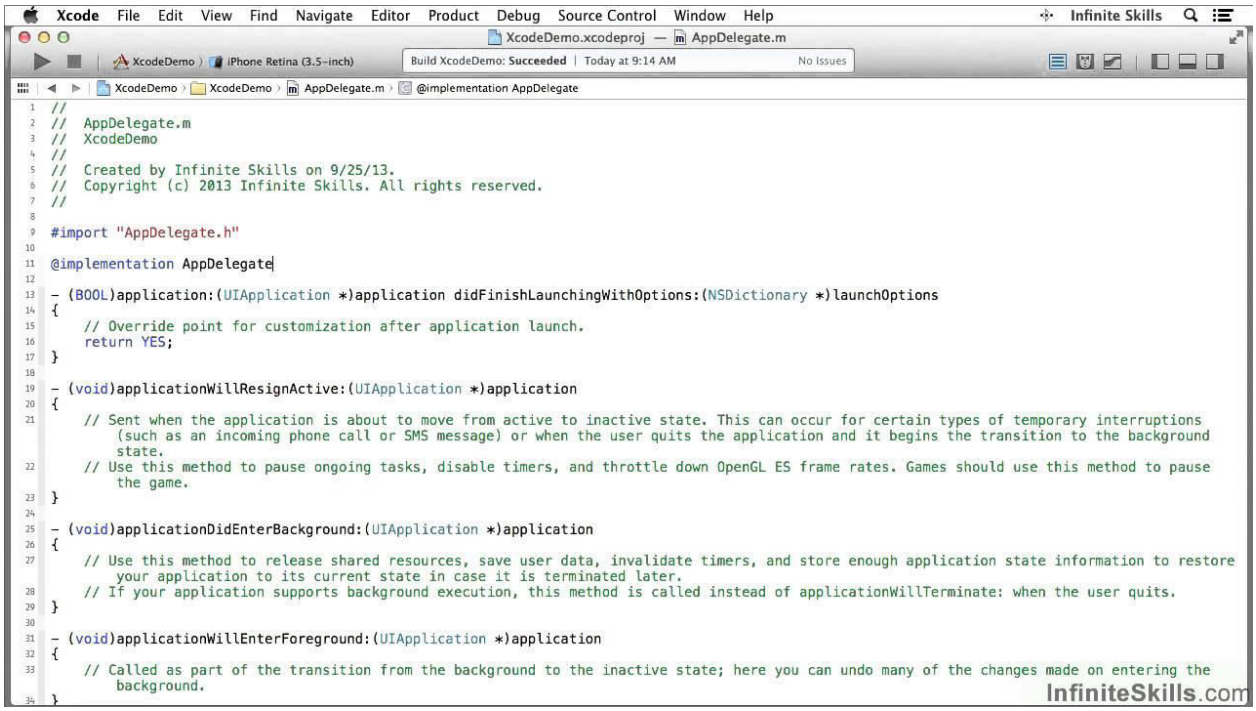

Рисунок 2.10 – Приклад коду написаного на мові програмування Objective-C

Код написаний на Objective-C є надійним, оскільки проведена велика кількість часу над його оптимізацією та відмово стійкістю. Також важливо

зазначити, що оскільки ми можемо використовувати також C та C++, які є мовами низько рівня, то це дає нам доступ до системного рівня операційної системи, що несе за собою більше варіантів та можливостей оптимізації роботи програми.

Objective-C, як і Swift, вимагають для розробки спільне середовище програмування під назвою «Xcode», яке розроблялось поступово з еволюцією Objective-C, враховуючи усі його потреби. Підтримка ж Swift з'явилась значно пізніше і була додана з урахуванням уже наявного функціоналу, що вилилось в не дуже ідеальні характеристики роботи, в порівнянні з випадком використання Objective-C.

Приклад вікна програми Xcode наведено на рис.2.11.

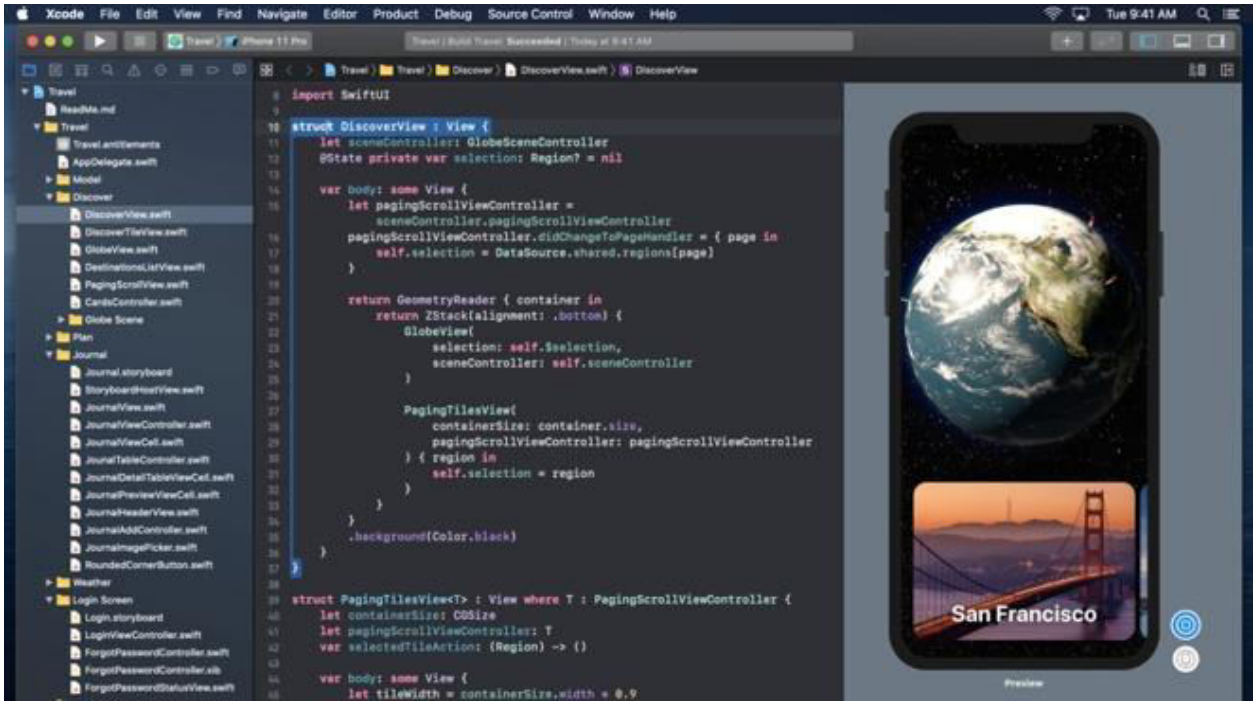

Рисунок 2.11 – Приклад вікна програми Xcode

Аналогічно до Swift, Objective-C має автоматичну систему підрахунку посилань і, при необхідності, видаляє об'єкти які більше не використовуються з пам'яті пристрою.

Objective-C має велику кількість бібліотек готових до використання в будь який момент, що, в поєднанні з кодовою базою мов програмування C та

C++ відчиняє перед розробником цілий «колодязь» варіантів вибору інструменту для вирішення кожної задачі яка постане під час написання коду програми.

Враховуючи усі наведені вище факти можна зробити висновок, що Objective-C задовольняє усі потреби для написання iOS додатку і може бути успішно використаний для його розробки приводячи до максимально ефективного фінального результату.

3. РОЗРОБКА КОРИСТУВАЦЬКОГО ІНТЕРФЕЙСУ IOS ДОДАТКУ

Більшість розробок програмного забезпечення які містять користувацький інтерфейс починаються з дизайну.

Дизайн є візитною картою додатку. В кінцевому результаті він має відповідати сучасним стандартам вигляду та привертати увагу з першого погляду.

Більш важливим його аспектом є зручність користування. Перед початком роботи дизайнеру необхідно уявити кінцевий результат роботи, бо інакше він не зможе скласти чіткої картини фінального вигляду додатку, що може призвести до незрозумілого або неінтуїтивно спроектованого інтерфейсу.

Отже, фінальний варіант повинен бути привабливим для ока користувача додатку, мати приємний користувацький інтерфейс, усі функції та властивості повинні бути інтуїтивно зрозумілі та не занадто складними в імплементації зі сторони розробника.

3.1 Графічний редактор Figma

Figma – це редактор який базується на використанні векторного типу графіки, тобто на побудові візуальних зображень за допомогою геометричних фігур, таких як: точки, лінії, полігони, тощо.

Великою зручністю використання Figma є те, що всі файли та макети по стандарту зберігаються у хмарному сховищі. Це дозволяє всім, кому необхідно їх побачити, просто перейти за посиланням та отримати доступ до синхронізованого та найбільш актуального на певний момент контенту[15].

Також така система поширення файлів дозволяє декільком дизайнерам одночасно працювати над одним і тим же проектом паралельно синхронізувавши свою роботу, що підвищує рівень швидкості розробки дизайну користувацького інтерфейсу в рази.

Приклад інтерфейсу графічного редактору Figma наведено на рис.3.1.

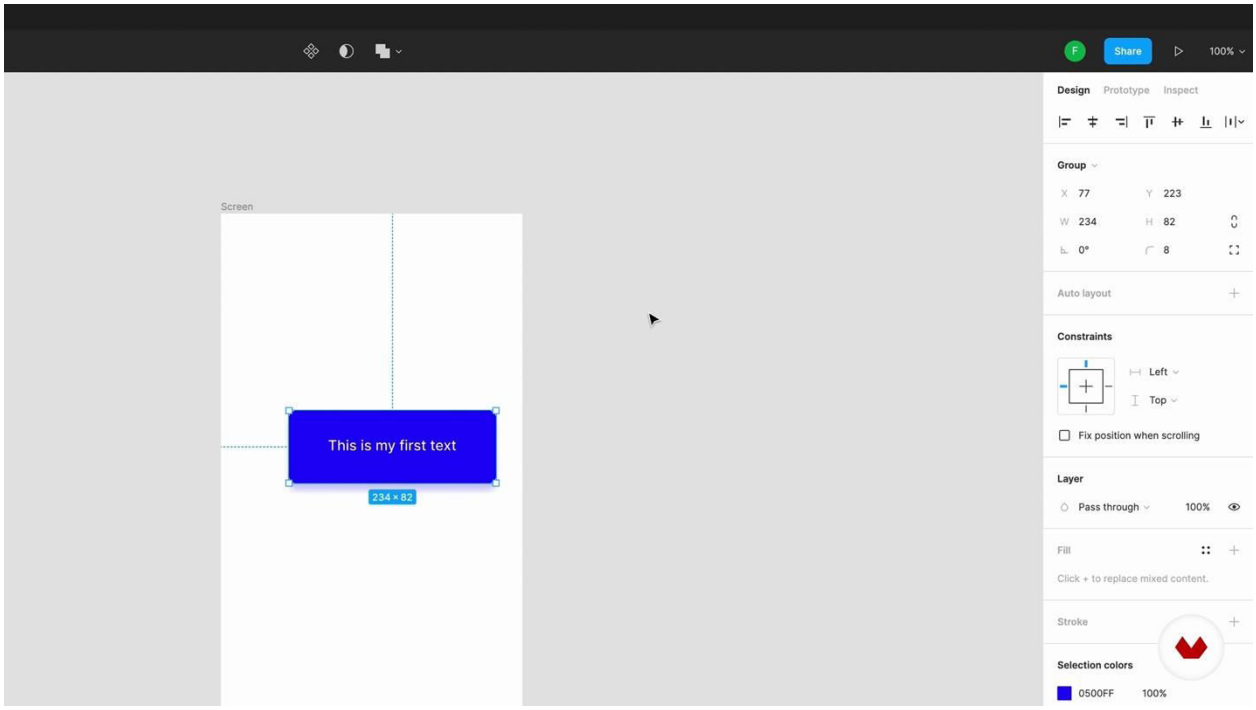

Рисунок 3.1 – Інтерфейс графічного редактору Figma

Figma має змогу впровадження системи компонентів, тобто стандартизації окремих елементів дизайну, які можуть повторюватися на різних екранах дизайну додатка.

Отже, Figma є зручними, надійним та перевіреним інструментом розробки користувацького інтерфейсу майбутнього iOS додатку, що в повній мірі задовольняє потрібності для вирішення цієї задачі.

3.2 Розробка користувацького інтерфейсу iOS додатку

В ході проектування користувацького інтерфейсу була обрана загальна назва системи для пошуку та організації колективних фізичних заходів – «MoveJoin».

Були окреслені загальні характеристики майбутнього дизайну та позначені основні необхідні екрани:

- початковий екран який першим зустрічає користувача, якщо він ще не має облікового запису, та дозволяє увійти в аккаунт або створити новий;
- екран який дозволяє увійти в уже існуючий аккаунт;
- екран створення нового аккаунту;
- головний екран з переліком усіх наявних оголошень про спортивні події з можливістю їх фільтрації;
- екран профілю користувача з можливістю вийти з аккаунту та повернутися на привітальну сторінку;
- екран додавання нового особистого оголошення про спортивну подію;

Після визначення послідовного плану дій була почата робота по створенню користувацького інтерфейсу, в результаті якої були розроблені усі необхідні для функціонування додатку елементи дизайну. Приклад привітального екрану зображений на рис.3.2.

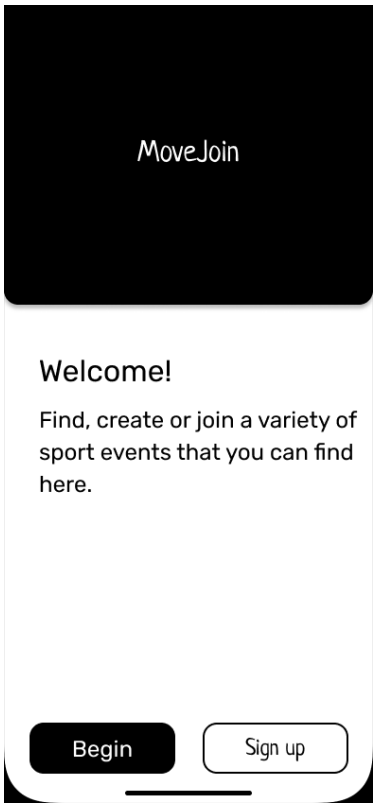

Рисунок 3.2 – Дизайн привітального екрану розробленого в редакторі Figma

На даному екрані наявна привітальна інформація для користувача та дві інтерактивні кнопки, які дають змогу або перейти на екран створення аккаунту, у випадку натискання на «Sign up», або перейти на екран входження в уже існуючий, у випадку з «Begin». Приклад екрану реєстрації користувача наведено на рис.3.3.

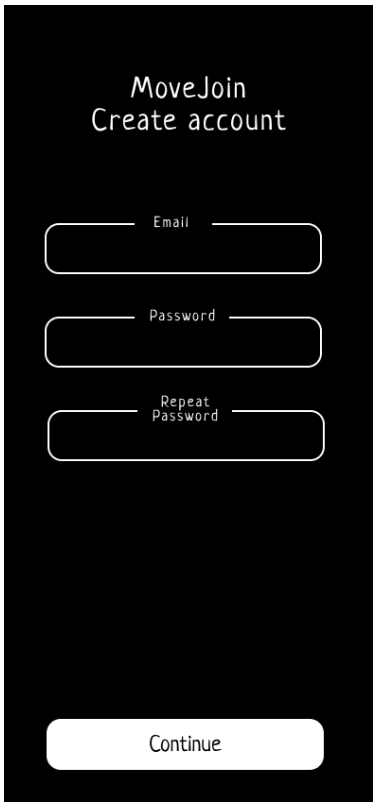

Рисунок 3.2 – Дизайн екрану реєстрації розробленого в редакторі Figma

У процесі, основною кольоровою палітрою для дизайну було вирішено обрати чорно-білу.

Фінальний варіант розробленого користувацького інтерфейсу відповідає усім вимогам визначеним у розділі 3. З точки зору імплементації, зі сторони написання програми iOS додатку, результат є збалансованим у відношенні складності та зручності використання, що, в свою чергу, вже дозволяє почати розробку та написання коду усіх сегментів системи.

4. ТЕОРЕТИЧНІ ЗАСАДИ ДЛЯ ВИРІШЕННЯ ПОСТАВЛЕНОЇ ЗАДАЧІ

4.1 Загальна характеристика системи

Фінальний варіант системи повинен відповідати певним критеріям та правилам побудови як серверної частини, так і iOS додатку. Повинні виконуватись наступні пункти:

- необхідне досягання максимально можливої швидкодії усіх компонентів системи;
- усі компоненти системи повинні мати змогу комунікувати один між одним за допомогою мережі;
- серверна частина повинна мати змогу отримувати та обробляти усі надіслані їх запити від клієнта;
- iOS додаток повинен мати зручний та зрозумілий інтрерфейс;
- усі паролі які надходять до або від сервера повинні мати зашифрований вигляд;
- користувач за допомогою клієнта повинен мати змогу переглядати увесь доступний функціонал системи;

Для побудови грамотної моделі бази даних необхідно, для початку, чітко окреслити майбутні функціональні здатності користувача у системі в залежності від його стану.

4.2 Роль користувача у системі

Розмежування користувачів за типом доступу необхідне для правильного відокремлення доступного для кожного з них функціоналу додатку. Оскільки в нинішньому варіанті клієнту системи відсутні такі ролі як «Модератор» та такі явища як «платні підписки», то на даний момент будуть присутні усього два типи користувачів, а саме:

- користувач-гість;

– користувач-авторизований;

Роль певного користувача залежить від того чи пройшов він процедуру авторизації від моменту встановлення додатку на свій особистий пристрій. У випадку якщо додаток не може визначити наявну авторизацію він переходить у режим роботи «гостя» і відображає привітальне початкове вікно.

Можливості користувача у режимі «гість» наведено на рис.4.1.

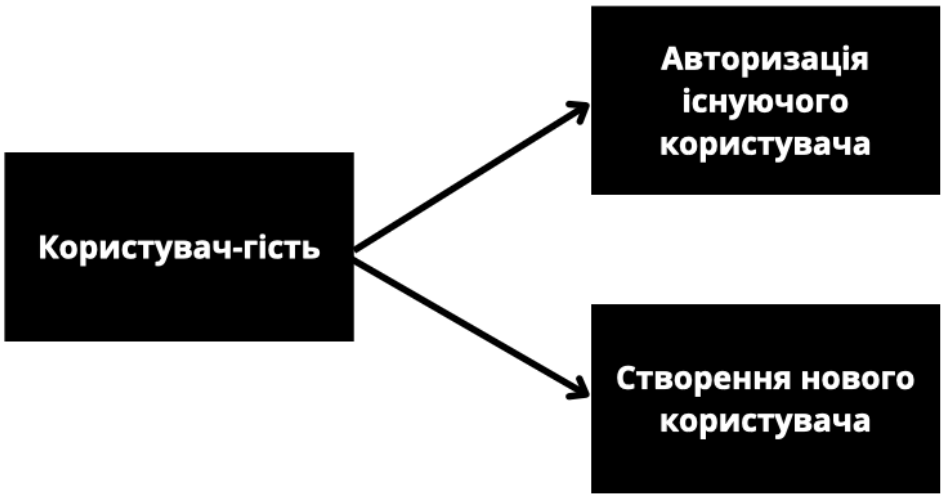

Рисунок 4.1 – Можливості користувача у режимі «гість»

Як видно з рисунку наведеного вище, користувачі з таким типом доступу не отримують доступу до основного функціоналу системи. Для розширення наданих можливостей необхідно пройти авторизацію за допомогою одного з двох доступних методів.

Можливості користувача у режимі «авторизований» наведено на рис.4.2.

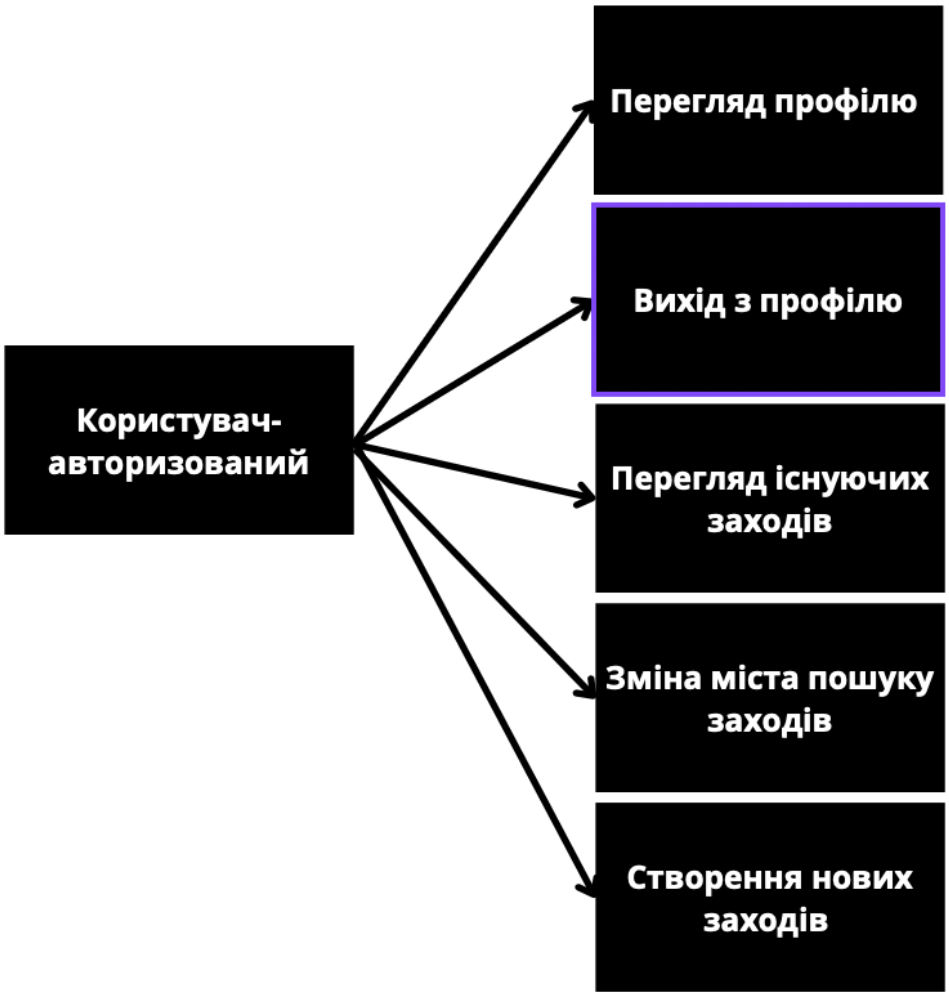

Рисунок 4.2 – Можливості користувача у режимі «авторизований»

Даний тип розширює доступ користувача до максимально можливого, на даний момент, рівня доступу до функціональних можливостей додатку і дає змогу використовувати його в повній мірі. Отримується доступ до управління власним профілем, перегляду або створення нових фізичних заходів.

Визначивши основні функціональні можливості усіх доступних типів користувачів можна починати розпочати розробку та проектування бази даних системи «MoveJoin».

4.3 Проектування бази даних

Обрана «клієнт-серверна» модель системи дозволяє клієнту не залежати від конкретного місця розташування бази даних, тобто вона може бути

розміщена як локально, так і віддалено. Усі операції з нею відбуваються на стороні сервера як наслідок на певний запит від клієнта. Клієнт також обробляє відповіді надіслані базою даних і форматує їх у певний вигляд який очікує клієнт.

Обрана раніше система керування базою даних PostgresQL є реляційною. Це означає, що усі дані всередині неї структуруються та зберігаються у вигляді таблиць, кожне поле яких відповідає одному конкретному значенню. Поля таблиць, в свою чергу, також можуть бути пов’язані між собою за допомогою певних правил відношень.

В результаті проектування була реалізована база даних яка представляє собою модель «Сутність - Зв’язок». Сутність – це суб'єкт, місце, річ, подія або поняття, що містять інформацію. Точніше, сутність – це набір (об'єднання) об'єктів, званих екземплярами. Кожен екземпляр сутності володіє набором характеристик. У логічної моделі БД всі ці характеристики називаються атрибутами сутності[16].

4.3.1 Таблиці бази даних

Сутності представлені у базі даних:

- користувачі;
- міста;
- заходи;
- типи заходів;
- участники заходів;

Ці сутності являють собою певний набір полів з певними типами даних, які в них зберігаються. Проте, кожна з них повинна також мати ті поля, які є унікальними для неї і, по суті, являються ідентифікаторами кожного окремого представленого об’єкту сутності. Такі поля ще називають «первинними». Первинні поля не мають бути константними, оскільки їх зміна приведе до переходу сутності в інший його примірник.

Зазвичай первинними ключами стають поля які описують певний номер об’єкту у таблиці. З кожним додавання нового примірника цей лічильник збільшується на одиницю, що повністю виключає випадки коли ці ключі можуть повторюватися у двох і більше записів одночасно. Також варто зазначити що перший елемент не повинен мати значення нуль. Первинний елемент повинен мати ідентифікатор зі значенням одиниці.

Перелік усіх доступних у межах бази даних сутностей наведе у таблиці 4.1.

Таблиця 4.1 – Список всіх сутностей БД системи «MoveJoin»

Users	Користувачі системи
Cities	Міста проведення заходів
Activities	Спортивні заходи
ActivityTypes	Види спорту спортивних заходу
ActivityParticipants	Учасники спортивних заходів

Наступним кроком необхідно описати усі сутності системи надавши їх необхідні поля та описавши їх первинні ключі. Відомості про таблиці бази даних представлені нижче у вигляді таблиць (табл. 4.2 – 4.6).

Таблиця 4.2 –Місто системи «MoveJoin»

No	Поле	Атрибут	Тип
1	city_id	Ідентифікатор	int
2	city_name	Назва	varchar(50)

Таблиця 4.3 –Користувач системи «MoveJoin»

No	Поле	Атрибут	Тип
1	user_id	Ідентифікатор	int

2	city_id	Ідентифікатор міста проживання	int
3	name	Ім’я	varchar(50)
4	surname	Фімілія	varchar(50)
5	email	Адреса електронної пошти	varchar(50)
6	password	Зашифрований пароль	varchar(250)

Таблиця 4.4 –Вид спорту системи «MoveJoin»

No	Поле	Атрибут	Тип
1	activity_type_id	Ідентифікатор	int
2	activity_type_name	Назва	varchar(50)

Таблиця 4.5 –Фізичний захід системи «MoveJoin»

No	Поле	Атрибут	Тип
1	activity_id	Ідентифікатор	int
2	activity_type_id	Назва	varchar(50)
3	creator_id	Ідентифікатор власника заходу	int
4	title	Назва	varchar(50)
5	city_id	Ідентифікатор міста проведення	int
6	description	Опис	varchar(250)
7	participants_count	Кількість учасників	smallint

8	date	Дата створення	timestamp

Таблиця 4.6 – Учасник заходу системи «MoveJoin»

No	Поле	Атрибут	Тип
1	activity_participant_id	Ідентифікатор	int
2	activity_id	Ідентифікатор заходу	int
3	participant_user_id	Ідентифікатор учасника	int

Відношення таблиць бази даних системи «MoveJoin» зображено на рис.4.3.

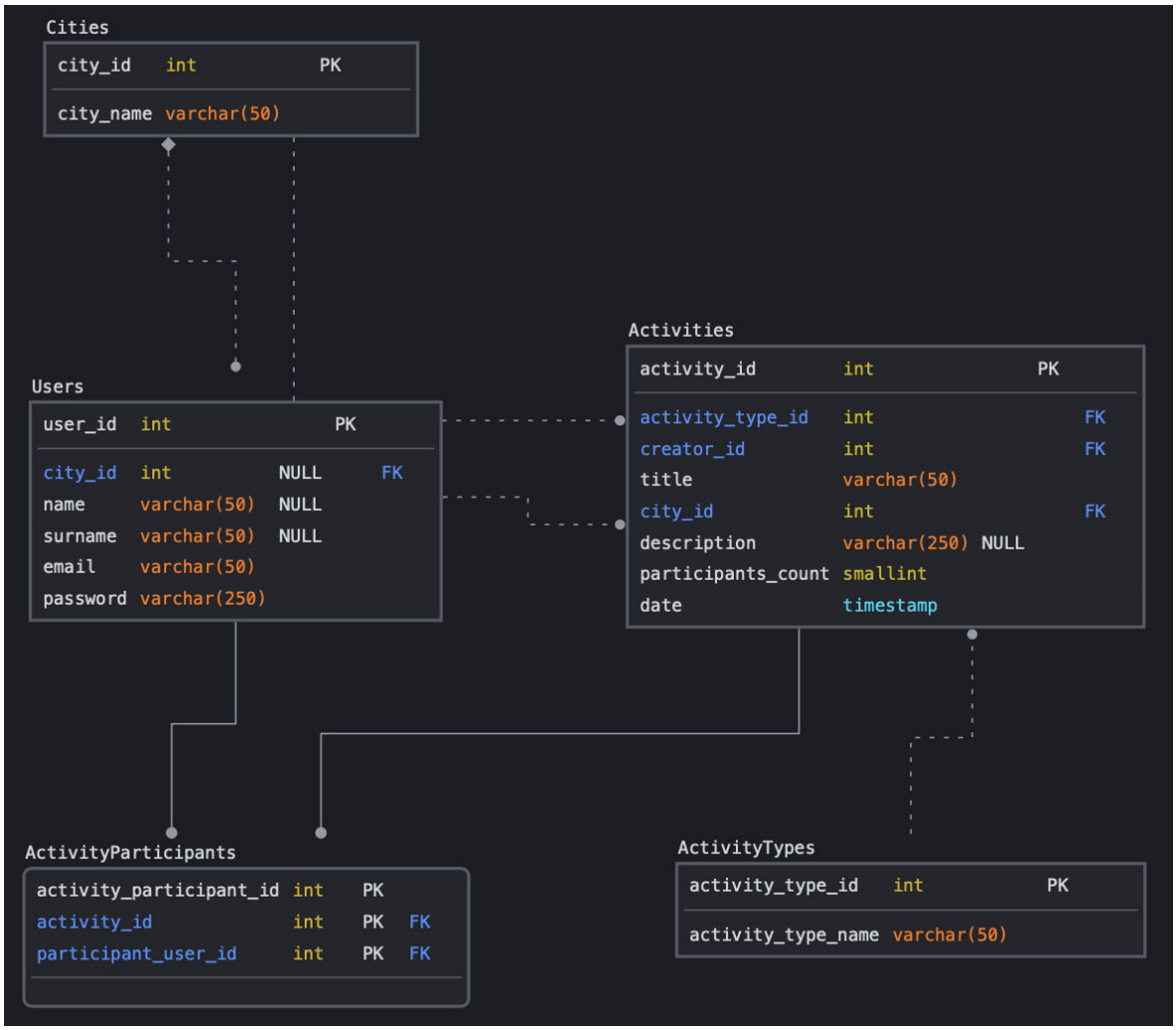

Рисунок 4.3 – Відношення таблиць бази даних системи «MoveJoin»

Розроблена база даних відповідає усім поставленим переде нею вимогам і може буди успішно використана у роботі системи. Наступним кроком реалізації системи повинна стати програмна реалізація серверної та клієнтської частин системи.

5. ПРОГРАМНА РЕАЛІЗАЦІЯ

5.1 Принцип роботи системи

Система реалізовану у вигляді клієнтської та серверної частин, кожна з яких займається своєю відокремленою задачею.

Серверна частина на початку своєї роботи встановлює зв'язок з базою даних та налаштовує усі доступні маршрути по яким клієнт матиме змогу робити запити. Отримавши запит, сервер обробляє усю присутню у ньому інформацію та виконує визначені дії. В результаті він повертає їх результат назад до клієнта у певному формату, яка в нашому випадку має вигляд байтового представлення згенерованого за допомогою функціоналу Protocol Buffers.

Клієнт, в свою чергу, займається відображенням певної корисної інформації для користувача, її зміною та взаємодією з нею. Все це відбувається в чіткій послідовності як результат певних дій людини, яка користується мобільним додатком. Натискання по екрану в певному його місці завжди веде за собою певний набір команд, які буде виконано і, залежно від результату, відображено їх наслідок. Наприклад, якщо натиснути на кнопку додавання нового заходу на головному екрані, то у ста відсотках випадків відкриється новий контролер з відповідним функціоналом, і нічого більше.

Приклад екрану додавання заходу наведено на рис.5.1.

Рисунок 5.1 –Екран додавання нового заходу iOS додатку «MoveJoin»

Заповнивши всі доступні поля та натиснувши на інтерактивну кнопку «Add» ми змусимо клієнт відправити серверу новий запит на створення нового об’єкту фізичного заходу у базі даних. Якщо це вийде успішно зробити, то сервер відправить назад відповідь з присутньою у ньому інформацією про щойно створений запис.

5.2 Навігація всередині додатку

Навігація всередині iOS додатку «MoveJoin» між різними частинами його функціоналу відбувається за допомогою панелі з відповідними доступними сегментами. На даний момент доступно лише три з них:

- розташований з лівого боку головний екран зі списком наявних фізичний заходів та можливістю зміни поточного міста їх пошуку;
- розташований по центру екран додавання нового фізичного заходу;
- розташований по правому краю екран профіля користувача;

Для того, щоб переключитись на один з них, користувачу достатньо лише натиснути на відповідну картинку розташовану у нижній частині додатку.

Приклад головного екрану додатку «MoveJoin» з присутнім меню навігації наведено на рис.5.2.

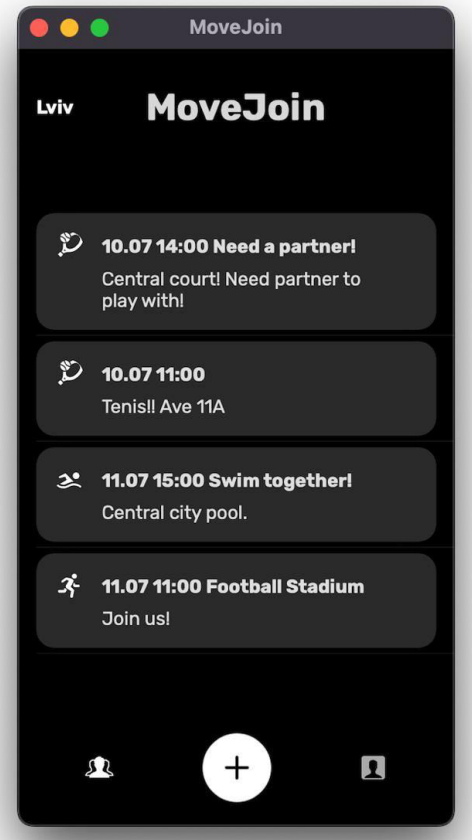

Рисунок 5.2 –Екран фізичних заходів iOS додатку «MoveJoin»

Для відображення інформації про поточного користувача використовується окремий екран, за допомогою якого можна дізнатись таку

інформацію як ім’я, фамілію та адрес електронної пошти. Також за його допомогою можливо вийти з поточного аккаунту та повернутись на привітальне меню додатку.

Екран профілю наведено на рис.5.3.

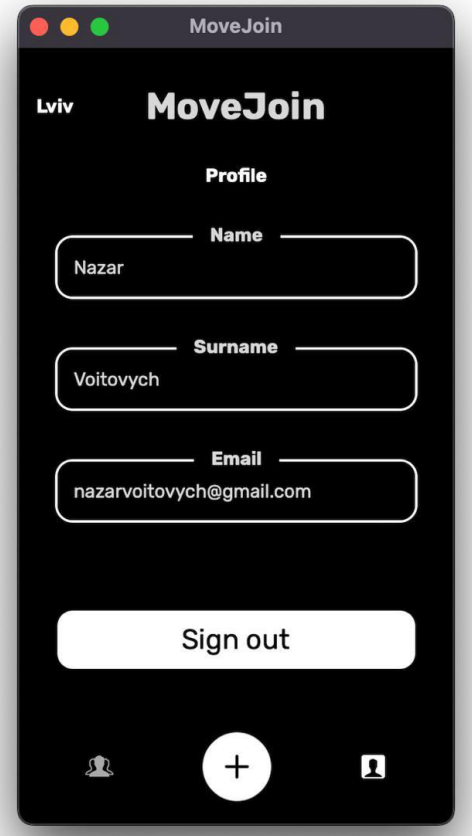

Рисунок 5.3 –Екран профілю iOS додатку «MoveJoin»

Розроблена система організації та пошуку спортивних фізичних заходів відповідає усім окресленим раніше вимогам та сучасним стандартам по розробці програмного забезпечення. Серверна частина має високі показники рівня швидкодії та ступеня безвідмовності. iOS додаток створений з зрозумілим користувацьким інтерфейсом та надає змогу отримати від нього повний обсяг доступного функціоналу.

ВИСНОВОК

У ході виконання дипломної роботи були виконані наступні етапи розробки системи для пошуку та організації колективних фізичних заходів:

1. окреслено основну тематику роботи, означено її необхідність та доречність;
2. проведено літературний огляд за темою роботи з подальшим викладом його результатів;
3. обрано інструменти необхідні для розробки кожного з компонентів системи.
4. розроблено користувацький інтерфейс iOS додатку за допомогою редактору Figma.
5. спроектовано базу даних системи.
6. розроблено програмне забезпечення усіх компонентів системи.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. У чому сила групових занять спортом? [Електронний ресурс] – Режим доступу до ресурсу: http://readonline.com.ua/items/32142-u-chomu-sila-grupovih-zanyat-sportom/.

2. Account deletion requirement starts June 30 [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/news/?id=12m75xbj.

3. Sport.ly App Page [Електронний ресурс] – Режим доступу до ресурсу: https://apps.apple.com/us/app/sport-ly-sport-meetup-pickup/id1185955518.

4. Sidelyne: Sports Events [Електронний ресурс] – Режим доступу до ресурсу: https://www.sidelyne.app/index.php - slide.

5. Вступ до використання JSON [Електронний ресурс] – Режим доступу до ресурсу: https://support.apple.com/uk-ua/guide/shortcuts/apd0f2e057df/ios.

6. JSON.- що це? [Електронний ресурс] – Режим доступу до ресурсу: https://apix-drive.com/ru/blog/useful/chto-takoe-json.

7. Protocol Buffers [Електронний ресурс] – Режим доступу до ресурсу: https://developers.google.com/protocol-buffers/docs/overview.

8. Порівняння MySQL та PostgresQL [Електронний ресурс] – Режим доступу до ресурсу: https://hyperhost.ua/info/ru/sravnenie-mysql-i-postgresql

9. MySQL [Електронний ресурс] – Режим доступу до ресурсу: https://dirask.com/posts/MySql-get-row-position-with-SELECT-query-1wwnJ1

10. Мова програмування Java [Електронний ресурс] – Режим доступу до ресурсу: https:/ru.hexlet.io/blog/posts/yazyk-programmirovaniya-java-osobennosti-populyarnost-situatsiya-na-rynke-truda

11. Java [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Java

12. Недоліки та переваги c++ [Електронний ресурс] – Режим доступу до ресурсу: http://naukam.triada.in.ua/index.php/konferentsiji/48-visimnadtsyata-vseukrajinska-praktichno-piznavalna-internet-konferentsiya/400-preimushchestva-i-

nedostatki-c-kak-pervogo-yazyka-programmirovaniya-dlya-nachinayushchego-razrabotchika

13. Недоліки мови програмування Swift [Електронний ресурс] – Режим доступу до ресурсу: https://www.quora.com/What-are-the-disadvantages-of-using-Swift-language

14. Swift vs Objective-C [Електронний ресурс] – Режим доступу до ресурсу: https://wnfx.ru/razrabotka-pod-ios-swift-vs-objective-c/

15. Figma [Електронний ресурс] – Режим доступу до ресурсу: https://prodesign.in.ua/2018/07/chomu-figma-zaminyla-sketch-v-dyzajn-komandi-templatemonster/

16. Основи автоматизованого проектування складних об’єктів і систем [Електронний ресурс] – Режим доступу до ресурсу: http://biblio.umsf.dp.ua/xmlui/bitstream/handle/123456789/4330/Навч_пос_ОАП_10_2018.pdf?sequence=1&isAllowed=y

ДОДАТКИ

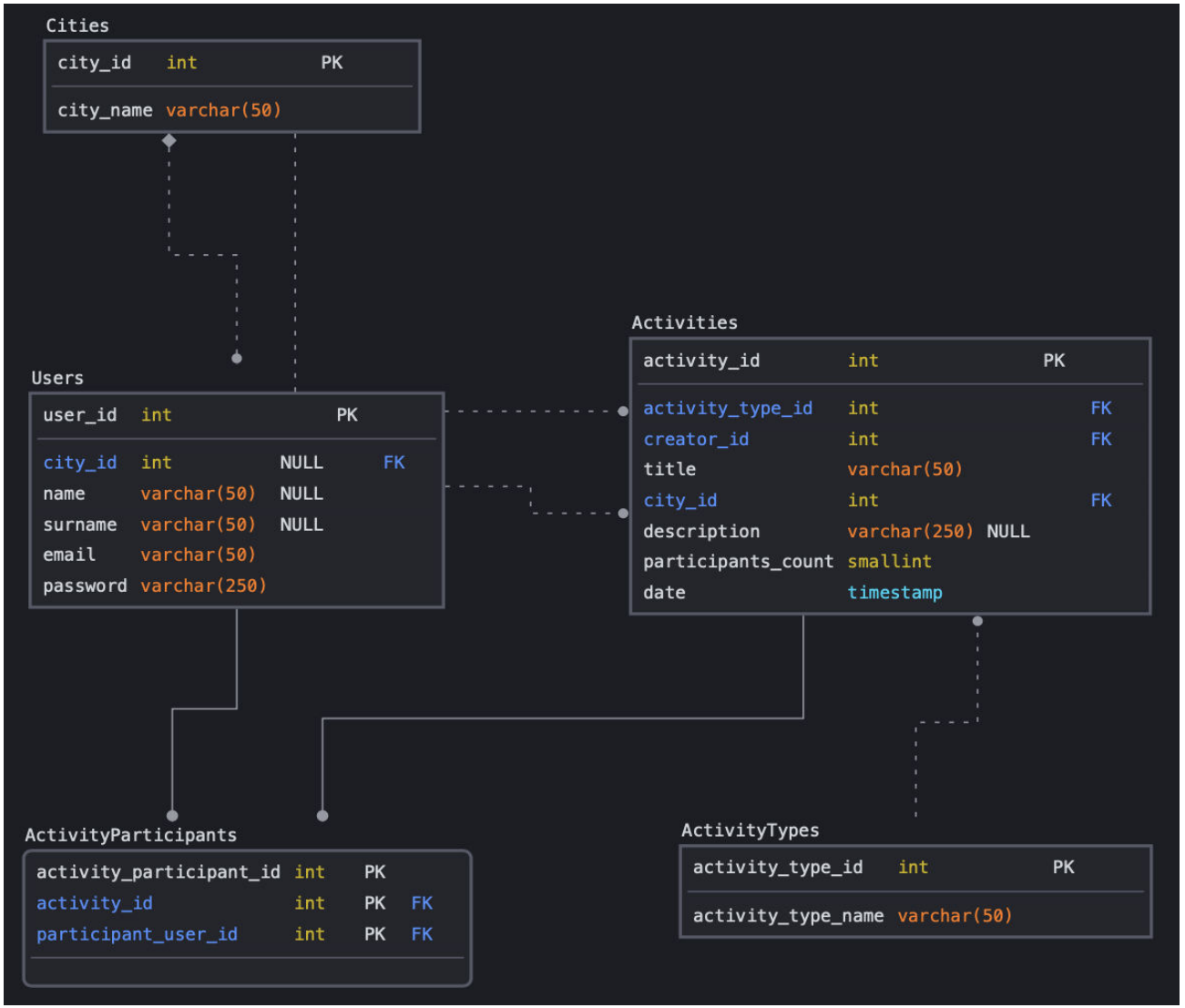

Рисунок А.1 – Відношення таблиць бази даних системи «MoveJoin»

ДОДАТОК Б ОСНОВНІ КОДИ ПРОГРАМНИХ МОДУЛІВ СИСТЕМИ

main.cpp

```
#ifndef MJ_BACKEND_BUILD

#include "http/httplib/httplib.h"
#include "postgres/include/tao/postgres.hpp"

#include "models/pb/cpp/city.pb.h"
#include "models/pb/cpp/user.pb.h"
#include "models/pb/cpp/activity_type.pb.h"
#include "models/pb/cpp/activity.pb.h"
#include "models/pb/cpp/activity_participant.pb.h"

using namespace httplib;

std::string result_to_cities(const tao::postgres::result& result) {
    Cities cities;

    for (const auto& row : result) {
        City *city = cities.add_cities();
        city->set_city_id(row["city_id"].as<int32_t>());
        city->set_city_name(row["city_name"].as<std::string>());
    }

    std::string target;
    cities.AppendToString(&target);

    return target;
}

std::string result_to_users(const tao::postgres::result& result) {
    Users users;

    for (const auto& row : result) {
        User *user = users.add_users();
        user->set_user_id(row["user_id"].as<int32_t>());
        user->set_city_id(row["city_id"].as<int32_t>());
        user->set_email(row["email"].as<std::string>());
        user->set_password(row["password"].as<std::string>());
        user->set_name(row["name"].as<std::string>());
        user->set_surname(row["surname"].as<std::string>());
    }

    std::string target;
    users.AppendToString(&target);

    return target;
}

std::string result_to_activity_types(const tao::postgres::result&
result) {
    ActivityTypes activity_types;

    for (const auto& row : result) {
        ActivityType *type = activity_types.add_activity_types();
        type->set_activity_id(row["activity_id"].as<int32_t>());
        type->set_activity_name(row["activity_name"].as<std::string>());
    }

    std::string target;
```

```
        activity_types.AppendToString(&target);

        return target;
    }

    Activities result_to_activities(const tao::postgres::result& result) {
        Activities activities;

        for (const auto& row : result) {
            Activity *activity = activities.add_activities();
            activity->set_activity_id(row["activity_id"].as<int32_t>());
            activity-
>set_activity_type_id(row["activity_type_id"].as<int32_t>());
            activity->set_creator_id(row["creator_id"].as<int32_t>());
            activity->set_title(row["title"].as<std::string>());
            activity->set_city_id(row["city_id"].as<int32_t>());
            activity->set_description(row["description"].as<std::string>());
            activity-
>set_participants_count(row["participants_count"].as<int32_t>());
            activity->set_date(row["date"].as<std::string>());
        }

        return activities;
    }

    std::string result_to_activities_string(const tao::postgres::result&
result) {
        Activities activities = result_to_activities(result);

        std::string target;
        activities.AppendToString(&target);

        return target;
    }

    ActivityParticipants result_to_activitity_participants(const
tao::postgres::result& result) {
        ActivityParticipants activity_participants;

        for (const auto& row : result) {
            ActivityParticipant *activity_participant =
activity_participants.add_activity_participants();
            activity_participant-
>set_activity_id(row["activity_id"].as<int32_t>());
            activity_participant-
>set_activity_participant_id(row["activity_participant_id"].as<int32_t>());
            activity_participant-
>set_participant_user_id(row["participant_user_id"].as<int32_t>());
        }

        return activity_participants;
    }

    std::string result_to_activitity_participants_string(const
tao::postgres::result& result) {
        ActivityParticipants activity_participants =
result_to_activitity_participants(result);

        std::string target;
        activity_participants.AppendToString(&target);

        return target;
    }
```

```
      int main(void)
      {
          const auto conn = tao::postgres::connection::create( "host=localhost
port=5432 dbname=movejoin connect_timeout=10" );
          conn->prepare("insert_user", "INSERT INTO Users (city_id, name,
surname, email, password) values ($1, $2, $3, $4, $5)");
          conn->prepare("insert_activity", "INSERT INTO Activities
(activity_type_id, creator_id, title, city_id, description,
participants_count) values ($1, $2, $3, $4, $5, $6)");
          conn->prepare("insert_activity_participant", "INSERT INTO
ActivityParticipants (activity_id, participant_user_id) values ($1, $2)");
          conn->prepare("update_participants_count", "update Activities set
participants_count = participants_count + 1 where activity_id = $1");
          conn->prepare("update_user_name", "update Users set name = $1,
surname = $2 where user_id = $3");

          if (!conn->is_open()) {
              return -1;
          }

          Server svr;

          svr.Get("/api/cities", [&](const Request &req, Response &res) {
              const auto result = conn->execute("SELECT * from Cities");
              res.set_content(result_to_cities(result), "application/json");
          });

          svr.Get(R"(/api/activities_list/(\d+))", [&](const Request &req,
Response &res) {
              auto city_id = req.matches[1].str();
              const auto result = conn->execute("SELECT * from Activities
where city_id = $1 and participants_count < 10", city_id);
              res.set_content(result_to_activities_string(result),
"application/json");
          });

          svr.Get(R"(/api/users/(\S+)/(\S+))", [&](const Request& req,
Response& res) {
              auto user_email = req.matches[1].str();
              auto user_password = req.matches[2].str();
              const auto result = conn->execute("SELECT * FROM Users where
email = $1 and password = $2", user_email, user_password);
              res.set_content(result_to_users(result), "application/text");
          });

          svr.Get(R"(/api/authorize/(\d+)/(\S+)/(\S+))", [&](const Request&
req, Response& res) {
              fprintf(stderr, "test: ");
              auto user_id = req.matches[1].str();
              auto user_name = req.matches[2].str();
              auto user_surname = req.matches[3].str();
              fprintf(stderr, "test: %s %s %s", user_id.c_str(),
user_name.c_str(), user_surname.c_str());
              const auto result = conn->execute("update Users set name = $1,
surname = $2 where user_id = $3", user_name, user_surname, user_id);

              res.set_content("1", "application/text");
          });

          svr.Post("/api/users", [&](const Request &req, Response &res, const
ContentReader &content_reader) {
              std::string body;
              content_reader([&](const char *data, size_t data_length) {
                  body.append(data, data_length);
```

```
                return true;
            });

            Users users;
            users.ParseFromString(body);
            User user = users.users(0);

            const auto email_users = conn->execute("SELECT * FROM Users
where email = $1", user.email());
            if (!email_users.empty()) {
                res.set_content("0", "application/text");
                return;
            }

            const auto tr = conn->transaction();
            tr->execute("insert_user", user.city_id(), user.name(),
user.surname(), user.email(), user.password());
            tr->commit();

            const auto user_result = conn->execute("SELECT * FROM Users
where user_id in (select last_value from users_seq)");
            res.set_content(result_to_users(user_result),
"application/text");
        });

        svr.Get("/api/activity_types", [&](const Request &req, Response
&res) {
            const auto result = conn->execute("SELECT * from
ActivityTypes");
            res.set_content(result_to_activity_types(result),
"application/json");
        });

        svr.Get(R"(/api/activities/(\d+))", [&](const Request& req,
Response& res) {
            auto activity_id = req.matches[1].str();
            const auto result = conn->execute("SELECT * FROM Activities
where activity_id = $1", activity_id);
            res.set_content(result_to_activities_string(result),
"application/text");
        });

        svr.Post("/api/activities", [&](const Request &req, Response &res,
const ContentReader &content_reader) {
            std::string body;
            content_reader([&](const char *data, size_t data_length) {
                body.append(data, data_length);
                return true;
            });

            Activities activities;
            activities.ParseFromString(body);
            Activity activity = activities.activities(0);

            {
                const auto tr = conn->transaction();
                const auto result = tr->execute("insert_activity",
activity.activity_type_id(), activity.creator_id(), activity.title(),
activity.city_id(), activity.description(), 1);
                tr->commit();
            }

            const auto activity_result = conn->execute("select * from
Activities where activity_id in (select last_value from activity_sequence)");
```

```
            activity = result_to_activities(activity_result).activities(0);

            {
                const auto tr = conn->transaction();
                const auto result = tr-
>execute("insert_activity_participant", activity.activity_id(),
activity.creator_id());
                tr->commit();
            }

            res.set_content(result_to_activities_string(activity_result),
"application/text");
        });

        svr.Get(R"(/api/activity_participants/(\d+))", [&](const Request&
req, Response& res) {
            auto activity_id = req.matches[1].str();
            const auto result = conn->execute("SELECT * FROM
ActivityParticipants where activity_id = $1", activity_id);

res.set_content(result_to_activitity_participants_string(result),
"application/text");
        });

        svr.Get(R"(/api/join/(\d+)/(\d+))", [&](const Request& req,
Response& res) {
            auto activity_id = req.matches[1].str();
            auto user_id = req.matches[2].str();

            const auto curr_result = conn->execute("SELECT * FROM
ActivityParticipants where activity_id = $1 and participant_user_id = $2",
activity_id, user_id);
            if (!curr_result.empty()) {
                res.set_content("0", "application/text");
                return;
            }

            {
                const auto tr = conn->transaction();
                tr->execute("insert_activity_participant", activity_id,
user_id);
                tr->execute("update_participants_count", activity_id);
                tr->commit();
            }

            const auto joined_result = conn->execute("SELECT * FROM
ActivityParticipants where activity_id = $1", activity_id);

res.set_content(result_to_activitity_participants_string(joined_result),
"application/text");
        });

        svr.listen("localhost", 1234);
    }

    #endif // MJ_BACKEND_BUILD

    MJUser.mm

    #import "MJUser.h"

    #include "models/pb/cpp/user.pb.h"

    @interface MJUser ()
```

```
@property (nonatomic, assign) ::std::shared_ptr<User> user;

@end

@implementation MJUser

- (instancetype)init {
    self = [super init];
    if (self) {
        self.user = ::std::shared_ptr<User>(new User);
    }
    return self;
}

+ (MJUser *)deserializeFromArrayData:(NSData *)data {
    Users users;

    users.ParseFromArray([data bytes], (int)[data length]);
    if (users.users_size() <= 0) {
        return nil;
    }

    MJUser *mjUser = [[MJUser alloc] init];
    mjUser.user->CopyFrom(users.users(0));
    return mjUser;
}

+ (MJUser *)deserializeFromData:(NSData *)data {
    MJUser *mjUser = [[MJUser alloc] init];

    mjUser.user->ParseFromArray([data bytes], (int)[data length]);
    return mjUser;
}

- (NSData *)serializedData {
    std::string stringData;
    self.user->AppendToString(&stringData);
    return [NSData dataWithBytes:stringData.data()
length:stringData.length()];
}

- (NSData *)serializedArrayData {
    Users users;
    User *array_user = users.add_users();

    array_user->set_email(self.user->email());
    array_user->set_password(self.user->password());

    std::string stringData;
    users.AppendToString(&stringData);
    return [NSData dataWithBytes:stringData.data()
length:stringData.length()];
}

- (BOOL)isEmpty {
    return self.user->email().empty();
}

- (void)setEmail:(NSString * _Nullable)email {
    self.user->set_email([email UTF8String]);
}

- (void)setPassword:(NSString *)password {
```

```
        self.user->set_password([password UTF8String]);
    }

    - (void)setName:(NSString *)name {
        self.user->set_name([name UTF8String]);
    }

    - (void)setSurname:(NSString *)surname {
        self.user->set_surname([surname UTF8String]);
    }

    - (NSInteger)userId {
        return self.user->user_id();
    }

    - (NSString *)password {
        return [NSString stringWithUTF8String:self.user-
>password().c_str()];
    }

    - (NSString *)email {
        return [NSString stringWithUTF8String:self.user->email().c_str()];
    }

    - (NSString *)name {
        return [NSString stringWithUTF8String:self.user->name().c_str()];
    }

    - (NSString *)surname {
        return [NSString stringWithUTF8String:self.user->surname().c_str()];
    }

    @end
```