МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

«До захисту допущено»
Т.в.о. завідувача кафедри

_____Світлана ВАЩЕНКО

____ _____ 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня бакалавра

зі спеціальності 122 «Комп'ютерні науки»,

освітньо-професійної програми «Інформаційні технології проектування»

на тему: Мобільний додаток інформування про проведення заходів

Здобувача групи ___ІТ-91_____Касьяненка Данила Романовича_____
(шифр групи) (прізвище, ім'я, по батькові)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ ___Данил КАСЬЯНЕНКО___
(підпис) (Ім'я та ПРІЗВИЩЕ здобувача)

Керівник кандидат технічних наук, доцент Яна ЧИБІРЯК_____
(посада, науковий ступінь, вчене звання, Ім'я та ПРІЗВИЩЕ) (підпис)

Суми-2023

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра інформаційних технологій
Спеціальність 122 «Комп'ютерні науки»
Освітньо-професійна програма «Інформаційні технології проектування»

ЗАТВЕРДЖУЮ
Т. в. о. зав. кафедри ІТ

_____Світлана ВАЩЕНКО
«_____» _____2023 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА СТУДЕНТУ

Касьяненко Данил Романович

1 Тема роботи,

керівник роботи __*Чибіряк Яна Іванівна, к.т.н., доцент*,

затверджені наказом по університету від «*29*» 05 2023 р. №0588-VI

2 Строк подання студентом роботи « 12 » червня 2023 р.

3 Вхідні дані до роботи ____функціонал мобільного додатку, системні вимоги

4 Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) вступ, аналіз предметної області, порівняння аналогів, вибір засобів реалізації, постановка задачі, моделювання програмної реалізації, моделювання варіантів використання, проектування процесного алгоритму, програмна реалізація додатку, тестування та працездатність, висновки, список літературних джерел, додатки.

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) _____

6. Консультанти розділів роботи:

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Аналіз предметної області	*Чибіряк Я. І.*		
Постановка задачі	*Чибіряк Я. І.*		
Моделювання та проектування	*Чибіряк Я. І.*		

7.Дата видачі завдання_____

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Оформлення планування робіт	До 19.02.2023	
2	Оформлення технічного завдання	До 28.02.2023	
3	Аналіз предметної області	До 10.03.2023	
4	Проектування та моделювання	До 30.03.2023	
5	Розробка мобільного додатку	До 15.05.2023	
6	Тестування додатку	До 21.05.2023	
7	Оформлення пояснювальної записки	До 06.06.2023	

Студент _____ Данил КАСЬЯНЕНКО
(підпис)

Керівник роботи _____ к.т.н., доц. Яна ЧИБІРЯК
(підпис)

РЕФЕРАТ

Тема роботи «Розробка мобільного додатку інформування про проведення заходів»

Пояснювальна записка містить вступ, розділ «Аналіз предметної області», розділ «Моделювання та проектування», розділ «Розробка ігрового додатку», висновки, список літературних джерел та додатки. Вона включає 101 сторінка, 27 рисунка, 21 літературне джерело та 11 таблиць.

Перший розділ «Аналіз предметної області» включає: огляд останніх досліджень і публікацій, аналіз програмних продуктів-аналогів та постановку задачі.

Другий розділ «Моделювання та проектування» включає: моделювання програмної реалізації, моделювання варіантів використання, проектування моделі бази даних.

Третій розділ «Розробка мобільного додатку» включає архітектуру програмного додатку, програмну реалізацію, тестування та працездатність.

Результатом дипломної роботи є готовий мобільний додаток інформування про проведення заходів.

Ключові слова: мобільний додаток, React Native, Node.JS, Expo, Web Socket, Firebase.

ЗМІСТ

ВСТУП

У сучасному оточенні значну увагу приділяють інформатизації регіонів, особливо розвитку інформаційного ринку. Прогрес у сфері техніки та технологій вимагає поліпшення комунікативних можливостей мобільних пристроїв у суспільстві. Це призводить до появи різноманітних інформаційних продуктів, які покращують якість життя та виконують різні завдання у різних сферах діяльності. Один з найбільш перспективних продуктів цього виду є мобільні додатки (mobile apps), які представляють собою окреме програмне забезпечення для певної операційної системи мобільного пристрою.[1]

Значною перевагою додатка «Connect People» є широкі можливості, що вона надає, серед таких: свобода від місця проживання, можливість знайти нові знайомства та цікавих людей; проводити заходи які важко було б зробити не маючи 100 знайомих; брати участь у цікавих вам заходам; заводити собі нових друзів.

Отже, метою даного дослідження є створення мобільного додатку «Connect People» для покращення комунікації з іншими людьми, який дозволить оптимізувати процес інформування користувачів щодо проведення заходів, а як наслідок знаходження нових знайомств. Також він дозволяє створювати так звані ‘events’ за допомогою яких ви можете як знайти людей для вашого заходу так і долучитися до цікавого вам.

Для досягнення мети проекту необхідно виконати наступні задачі:

- визначити актуальність роботи, дослідити предметну область та провести аналіз аналогів мобільних додатків;
- розробити та реалізувати структуру та функціонал мобільного додатку;
- виконати тестування мобільного додатку.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд останніх досліджень і публікацій

Мобільні пристрої надзвичайно швидко проникають в усі сфери життя як в Україні, так і в решті світу. У минулому році кількість проданих смартфонів перевищила кількість проданих персональних комп'ютерів. Експерти передбачають, що вже в наступному році продажі мобільних телефонів та планшетів перевершать продажі традиційних настільних комп'ютерів. Світові тенденції поширення мобільних пристроїв та мережевих технологій суттєво впливають на розвиток програмного забезпечення. (табл. 1.1). [1]

Таблиця 1.1 - Прогноз основних напрямків розвитку мережевих технологій

Показники розвитку	**2015-2020 роки**
1. Обсяг світового мобільного трафіку передачі даних	збільшився в 8 разів
2. Кількість користувачів мобільних пристроїв	5 млрд. осіб, або 70 % населення землі
3. Кількість пристроїв і з'єднань, готових до підключення до мобільних мереж	11,6 млрд.
4. Відео складає велику частину світового мобільного трафіку передачі даних.	75 %
5. Смартфони, планшети та ноутбуки генерують понад 90% мобільного трафіку даних.	смартфони, планшети, ноутбуки
6. Кількість портативних пристроїв становить.	від 97 до 600 млн.

На початку свого існування мобільні додатки були популярні переважно для швидкої перевірки електронної пошти. Проте, високий попит на них призвів до розширення їх функціональності й застосування в інших галузях, таких як ігри для мобільних телефонів, GPS-навігація, спілкування, перегляд відео та користування Інтернетом. Сучасний ринок мобільних додатків вже дуже розвинений і продовжує стійко зростати.

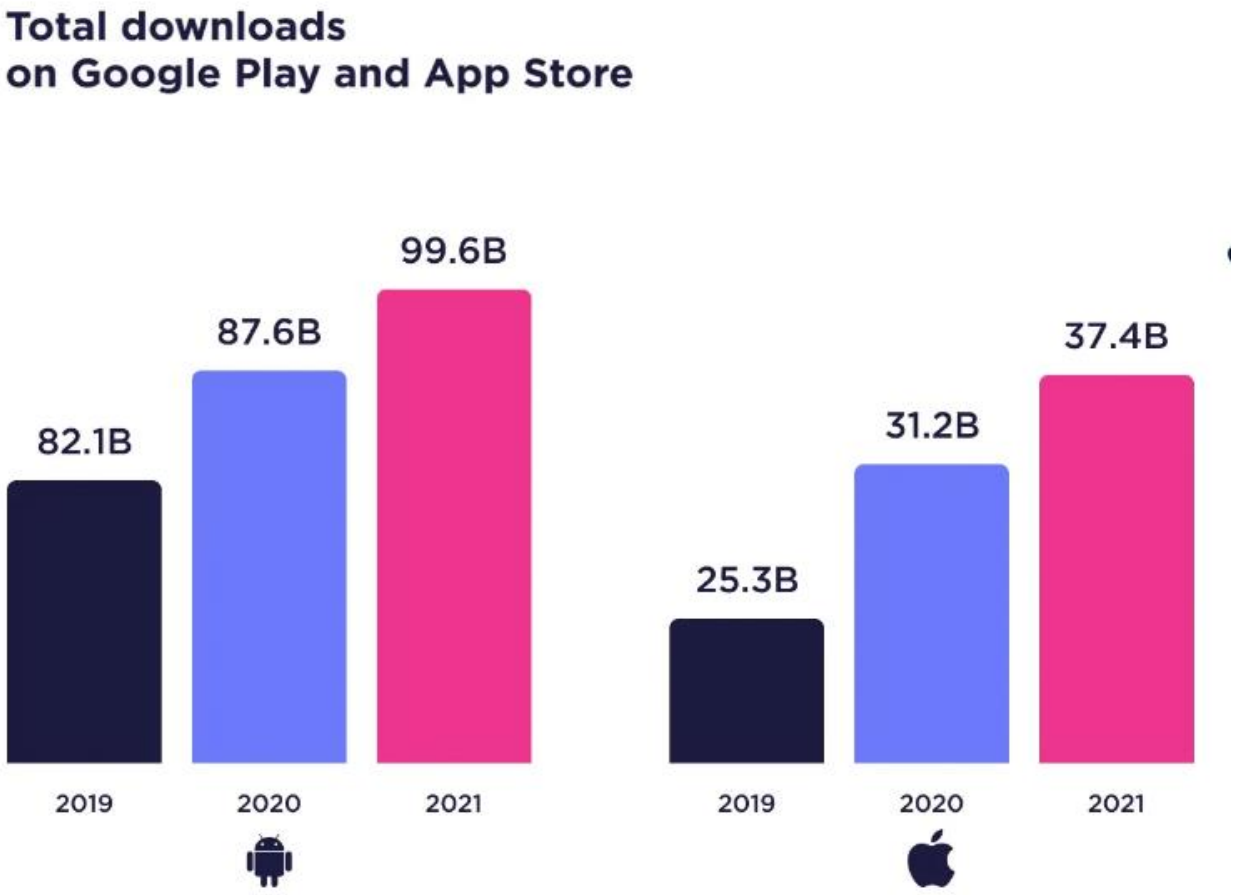


Рисунок 1.1 – Графік завантажень мобільних додатків [2]

Сьогодні складно виділити окремі категорії мобільних додатків так як один застосунок може бути сукупністю декількох типів. Тому треба оцінювати популярність всіх додатків та виділяти категорію чи їх комплекс. Так, наприклад за дослідженням компанії Kantar найбільш популярних в Україні мобільних додатків за вересень 2021 року(рис. 1.2), популярнішими стали Telegram, WhatsApp, Fb Messenger та багато інших додатків. [3]

Рисунок 1.2 – Графік завантажень мобільних додатків [3]

За графіком дослідження бачимо, що Telegram, який є месенджером перегнав Instagram за популярністю, та на кожний додаток соціальної мережі знаходиться застосунок чату, який є популярнішим та використовується більшою часткою користувачів.

Під час виконання преддипломної практики було проведено аналіз предметної області та встановлено, що є необхідність у створенні мобільного додатку, який буде месенджером з можливістю спілкуватися через мережу Інтернет та редагувати профіль.

1.2 Аналіз програмних продуктів-аналогів

На сьогоднішній день існує безліч мобільних додатків месенджерів. Розглянемо найпоширеніші з них та наближених до вихідного продукту.

1.1 Telegram

Згідно даних у месенджері Telegram станом на січень 2021 року кількість активних користувачів перевищує 500 мільйонів щомісяця. [4]

Компанія "Telegram" просуває свій додаток як месенджер, в якому всі особисті дані користувачів і їхні переписки є конфіденційними.

Месенджер Telegram дозволяє користувачам швидко обмінюватися текстовими повідомленнями, зображеннями, аудіофайлами (у тому числі голосовими повідомленнями), відеозаписами, відео повідомленнями та іншими файлами, які не є виконуваними і не перевищують розмір 2 ГБ. Крім того, додаток надає можливість передавати інформацію про місцезнаходження користувача за допомогою супутників GPS через трансляції або поштовхові повідомлення..

Вигляд додатку (рис 1.3):

Рисунок 1.3 – Telegram

Telegram використовує хмарні сховища та клієнти абонентів для зберігання даних, що свідчить про те, що архітектура цього месенджера є клієнт серверною..

Telegram використовує розроблений ним протокол MTProto для захисту даних. Передача повідомлень відбувається шифруванням симетричним ключем з використанням протоколу Діффі-Хеллмана, який обчислюється з використанням криптосистеми RSA. Для верифікації цілісності повідомлень використовується хеш-функція MD5. В рамках протоколу MTProto також використовується протокол TLS версії 1.3, що забезпечує безпечну передачу повідомлень та додатковий рівень безпеки під час обміну даними.[5]

1.2 Viber

Viber - VoIP-застосунок для дзвінків і обміну повідомленнями.[6]

Вигляд додатку (рис 1.4):

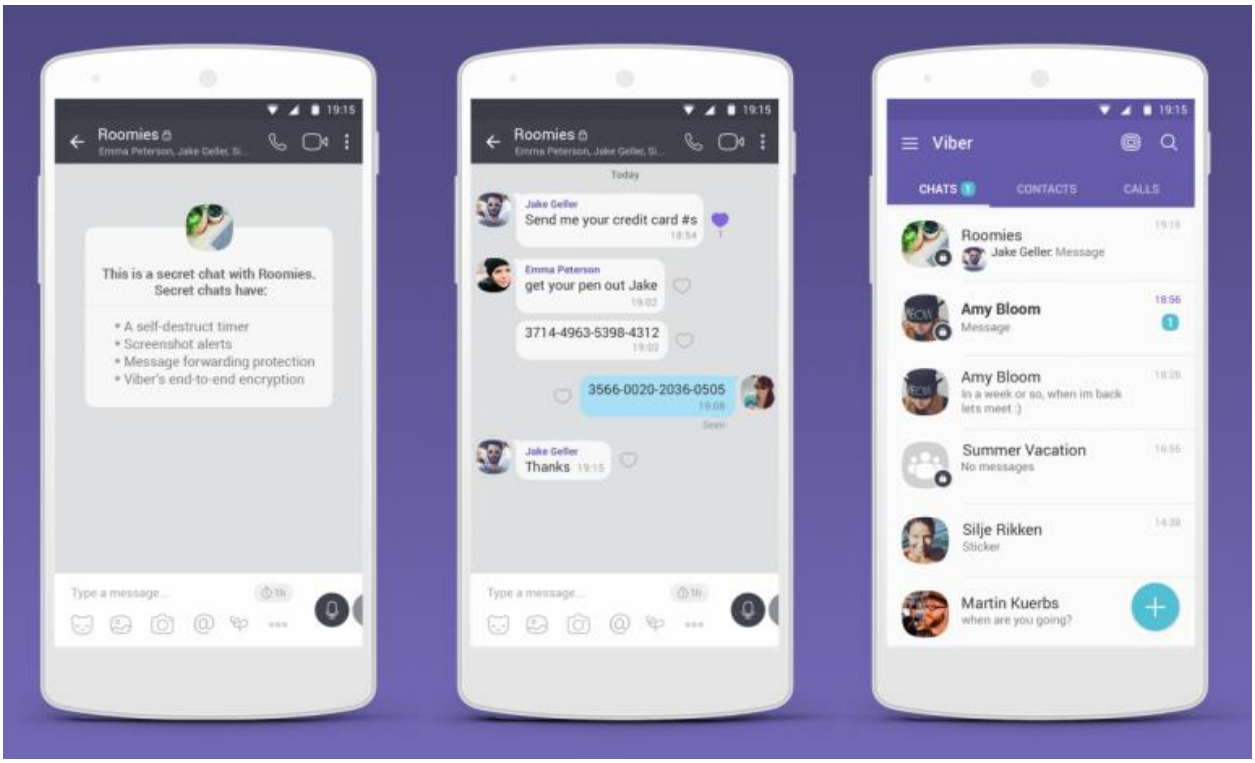


Рисунок 1.4 – Viber

Функціональність месенджера Viber в багатьох аспектах співпадає з функціональністю додатка Telegram. Проте існують суттєві відмінності:

• Viber дозволяє користувачам здійснювати банківські операції через чат ботів;

• Після надсилання повідомлення адресату, копія повідомлення на сервері видаляється, і єдиним місцем знаходження повідомлень є клієнти абонентів;

• Viber підтримує тільки поштучне надсилання геолокації клієнта;

• Є функція надсилання секретних повідомлень з автоматичним видаленням на всіх клієнтах учасників після певного часу.

Архітектура месенджера Viber подібна до Telegram за відповідними параметрами. Однак, на відміну від Telegram, розробники Viber стверджують, що не зберігають листування користувачів на своїх серверах після доставки повідомлення адресату. Після видалення додатку на всіх пристроях абонента і повторному встановленні, відновлюється інформація про участь абонента у бесідах, але зміст бесід та чати з абонентом повністю видаляються.

Захист даних в Viber здійснюється за допомогою розробленого компанією протоколу. Під час експерименту з прослуховуванням трафіку виявлено використання протоколу TCP та TLS версії 1.2, проте пакети не містили назву протоколу захисту.

1.3 Signal

Signal — це багатоплатформова служба зашифрованих миттєвих повідомлень, розроблена Signal Foundation та Signal Messenger LLC. Вона використовує Інтернет для надсилання індивідуальних та групових повідомлень, які можуть містити файли, голосові нотатки, зображення та відео.

Вигляд додатку (рис 1.4):

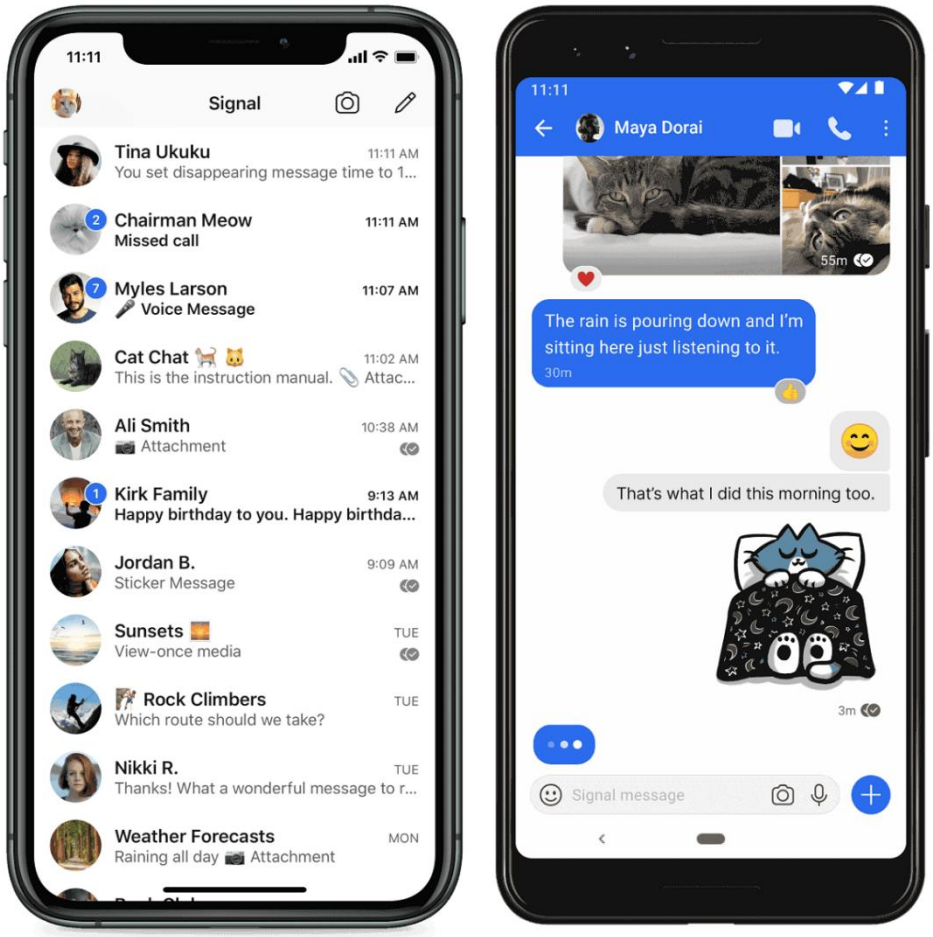


Рисунок 1.5 – Signal

Месенджер Signal пропонує багато функціональних можливостей, подібних до месенджера Viber. Він підтримує обмін файлами, геолокацію, аудіо, відео та текстові повідомлення. Однак, Signal не підтримує створення відео-повідомлень, і це є відмінністю в порівнянні з іншими месенджерами. Одна з важливих особливостей Signal полягає в тому, що користувачам заборонено створювати скріншоти під час використання додатку. Також, Signal має функцію "інкогніто-клавіатура", яка запобігає персоналізованому навчанню клавіатурного прогнозування. Signal забезпечує додатковий рівень конфіденційності за допомогою ретрансляції інтернет-дзвінків через свої сервери. Це допомагає уникнути розкриття реальних IP-адрес учасників дзвінка зловмисникам.

Архітектура Signal включає сервер для прийому повідомлень від відправників та доставки їх адресатам. Загалом, архітектура Signal подібна до

інших месенджерів, що були згадані раніше. Розробники Signal стверджують, що не зберігають на своїх серверах шифрограми листування після доставки повідомлень адресатам. Дані листування зберігаються на пристроях користувачів у вигляді шифрограм, які створюються з використанням парольної фрази користувача та шифрів довгострокового типу.

Signal використовує широкий спектр криптографічних протоколів для захисту даних. Використовуються протокол Діффі-Хелмана для створення симетричних ключів шифрування, AES256 для симет.

Було створено порівняльну таблицю в ході аналізу месенджерів. (Таблиця 1.2):

Таблиця 1.2 – Порівняння за функціональністю та захищеністю даних

Назва/ властивості додатку	Telegram	Viber	Signal
Резервне збереження даних	так	за вибором	за вибором
Комп’ютерна версія	так	так	так
Мобільна версія	так	так	так
Реєстрація за номером телефону	так	так	так
Назва/ властивості додатку	Telegram	Viber	Signal
Шифрування даних в ході їх збереження на пристрої користувачів	так	так	так
Наскрізне шифрування	за вибором	за вибором	так
Приховування факту одностороннього видалення повідомлення	так	ні	ні

Продовження таблиці 1.2 – Порівняння за функціональністю та захищеністю даних:

Назва/ властивості додатку	Telegram	Viber	Signal
Використання парольної фрази	за вибором	ні	так
Функція трансляції геолокації	так	ні	ні
Чат-боти	так	так	ні

Таблиця 1.2 надає можливість зосередитися на функціональних розширеннях, які можна впровадити під час розробки, а також на проблемах, які треба вирішити.

На основі проведеного аналізу було сформовано ряд вимог до власного додатку:

- Аворизація користувачів;
- Є можливість спілкування у чаті;
- Резервне збереження даних користувача та повідомлень у чатах;
- Можливість видалень повідомлень у чатах;
- Викорситання авторизації для більшої надійності даних;
- Реєстрація за електронною поштою, так як це зручніше ніж за номером телефону.

1.3 Постановка задачі

Мета цього дослідження полягає в створенні мобільного застосунку для інформування про проведення заходів, який матиме авторизацію та чати з учасниками заходів.

Таким чином, на цьому етапі для додатку «Connect People» були сформовані наступні твердження:

- додаток – це звичайний месенджер;
- присутня авторизація у додаток;
- в додатку є чати з іншими користувачами;
- можливий перегляд користувачів та відображення їх статусу(онлайн/офлайн).

Для досягнення основної мети проекту необхідно виконати такі завдання:

- оцінити актуальність проекту, визначити цільову аудиторію і провести дослідження предметної області;
- проаналізувати існуючі платформи комунікації, виявити їх переваги і недоліки;
- створити модель і структуру мобільного додатку;
- вибрати необхідні технології для розробки;
- розробити прототип мобільного додатку;
- реалізувати структуру додатку;
- розробити функціонал мобільного додатку;
- Провести тестування мобільного додатку.

У додатку А міститься повне технічне завдання на розробку продукту у всіх його аспектах.

2 МОДЕЛЮВАННЯ ТА ПРОЕКТУВАННЯ

Після аналізу предметної області, визначення мети та завдань проекту, формулювання вимог до функціоналу та вибору засобів реалізації, наступним етапом є моделювання та проектування мобільного додатку.

2.1 Моделювання програмної реалізації

Контекстна діаграма IDEF0 використовується як графічне зображення методології функціонального моделювання. Цей метод аналізу та документації бізнес-процесів був розроблений у 1970-х роках для виробничих галузей, але з розвитком технологій та появою комп'ютерів використання контекстних діаграм IDEF0 розширилося. Сьогодні ці діаграми застосовуються в різних сферах життєдіяльності, зокрема в сфері інформаційних технологій [8].

Для графічного представлення систем використовуються функції, вхідні та вихідні дані, механізми та об'єкти управління. У діаграмі IDEF0 функція отримує вхідні дані, а управління через механізми створює умови для отримання очікуваних вихідних даних. Процеси можуть бути розбиті на менші підпроцеси, що називається декомпозицією. Діаграма IDEF0 складається з блоків і стрілок, які використовуються з таким принципом:

1. Блоки представляють функції;
2. Стрілки, що входять зліва до блоку, вказують на вхідні дані;
3. Стрілки, що входять зверху до блоку, представляють управління;
4. Стрілки, що входять знизу до блоку, вказують на механізми;
5. Стрілки, що виходять справа від блоку, представляють вихідні дані [9].

Після отримання теоретичного уявлення про структуру діаграми і основний процес "Інформування про проведення заходів" визначаються такі параметри з погляду користувача рис.2.1:

- Вхідні дані: потреба користувача щодо участі в заході;

- Управління: функціонал мобільного додатку, системні вимоги;
- Механізми: мобільний додаток і користувач;
- Вихідні дані: долучення до заходу.

Контекстна діаграма мобільного додатку в нотації IDEF0 представлена на рисунку 2.1.

Рисунок 2.1 – Контекстна діаграма IDEF0

На рисунку 2.2 зображена декомпозиція діаграм реалізації мобільного додатку:

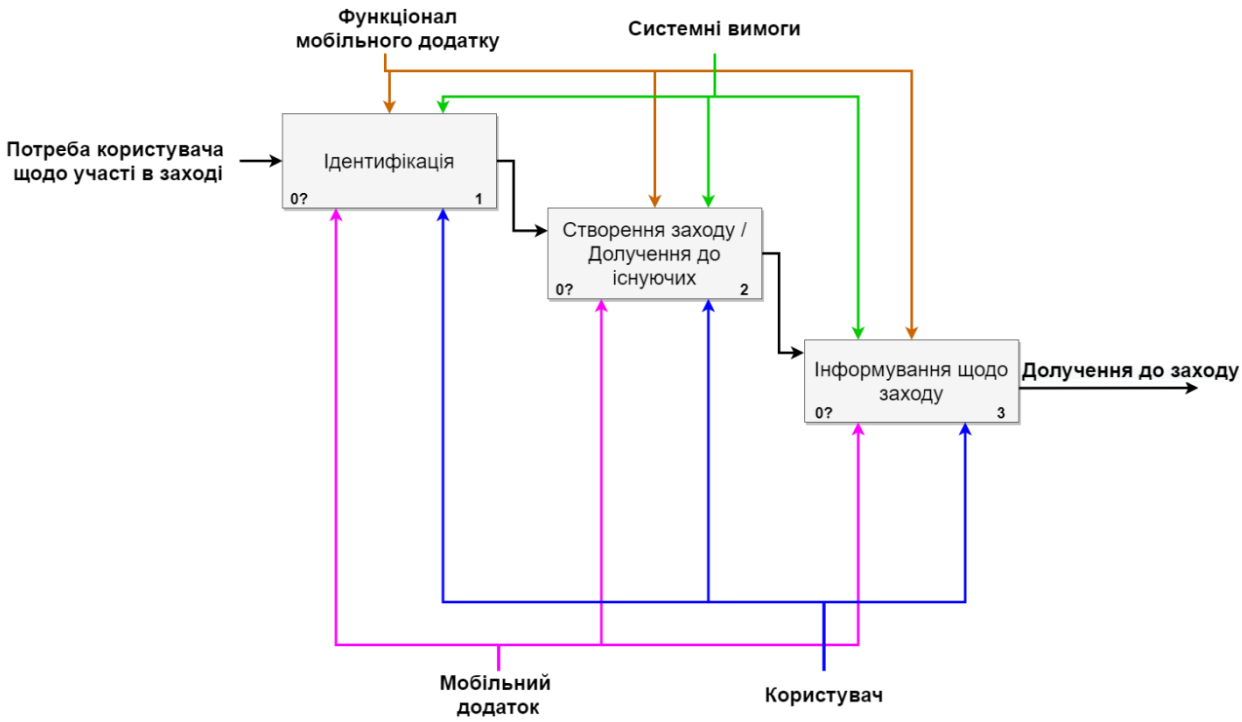


Рисунок 2.2 – Декомпозиція діаграми IDEF0

2.2 Моделювання варіантів використання

Варіанти використання діаграми, яка є найпростішою серед поведінкових діаграм UML, є надзвичайно корисними при поясненні функціональних особливостей програми людям, які не мають глибоких знань в галузі IT, наприклад, замовникам програмного забезпечення. Ця діаграма використовується для опису цілей, які користувачі вашої програми, будь то людина або інша програма, прагнуть досягти. Щоб бути точним, діаграма варіантів використання використовується для опису функціональних вимог до програми, її підсистем або інших об'єктів. Вона надає загальну уяву про те, як ваша програма буде використовуватись.[10]

На основі технічного завдання, ми встановлюємо акторів і варіанти використання у системі нашої ігрової програми.

Актори, що виступають у системі, включають:

- Користувач:
- База даних.

Варіанти використання в системі:

- Ідентифікація – процедура розпізнавання користувача в системі;
- Сторінка профілю –інформація про користувача;
- Редагування персональних даних – змінює персональні дані;
- Сторінка заходів – відображає список заходів;
- Чат заходу – відображає повідомлення користувачів;
- Учасники заходу – відображає список учасників;
- Створення заходу – створює новий захід.

На рисунку 2.5 зображена діаграма варіантів використання мобільного додатку.

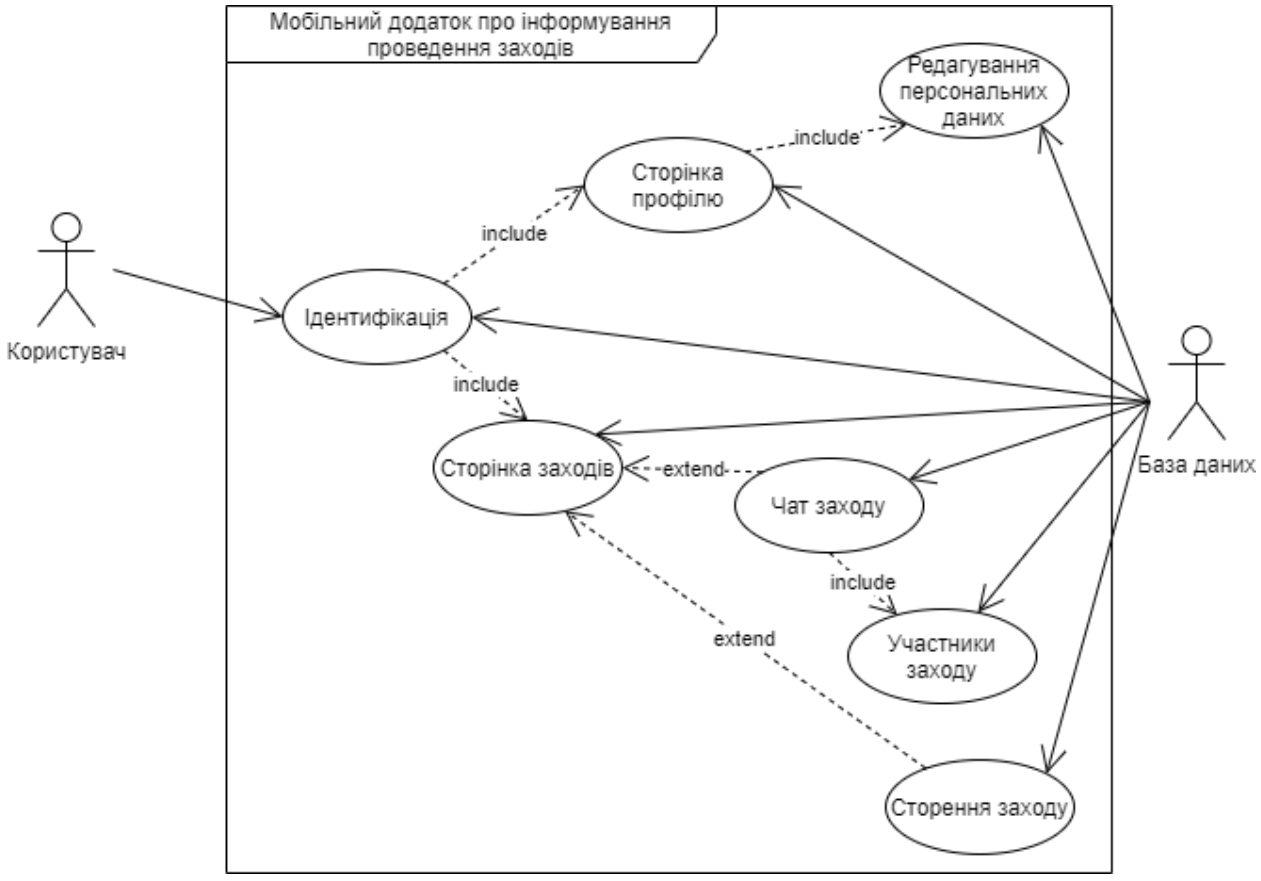


Рисунок 2.3 – Діаграма варіантів використання UML

2.3 Проектування моделі бази даних

В процесі проектування бази даних, ключовим елементом є створення моделі даних - системи концепцій і правил, які використовуються для відображення структури, стану і динаміки проблемної області в базі даних. З часом, різні моделі даних, такі як ієрархічна, мережева і реляційна, отримали значний розвиток.

На рисунку 2.4 зображено реляційна модель бази даних мобільного додатку:

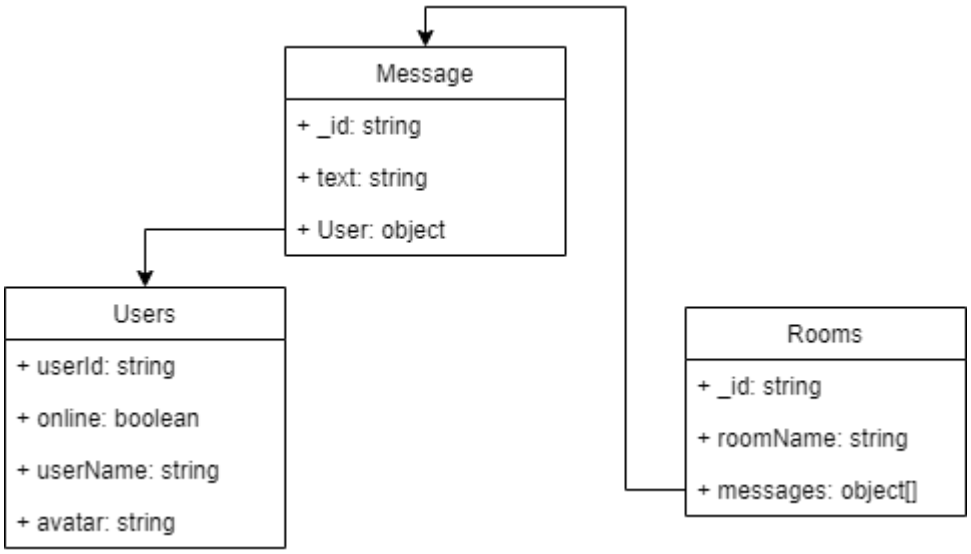


Рисунок 2.4 – Модель бази даних

3 РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ

3.1 Архітектура програмного додатку

Архітектура програмного додатку ділиться на чотири основних компонента:

1. Мобільний пристрій – додаток є об’єкт, вузли:
   1.1.з користувачем це сенсорний екран, за допомогою якого здійснюється управління;
   1.2.з web server це HTTPS та Web socket connection за допомогою якого здійснюється обмін інформацією.
2. Web server – додатками є інтерфейс, інтерфейс ідентифікації та бази даних, вузли:
   2.1.з Firebase це HTTPS connection за допомогою якого здійснюється обмін інформацією;
   2.2. з JSON це зчитування та запис інформації до файлів.
3. JSON – об’єктом є різні файли з інформацією;
4. Firebase – об’єктом є Firebase Authentication.

На рисунку 3.1 зображено діаграма розгортання архітектури мобільного додатку:

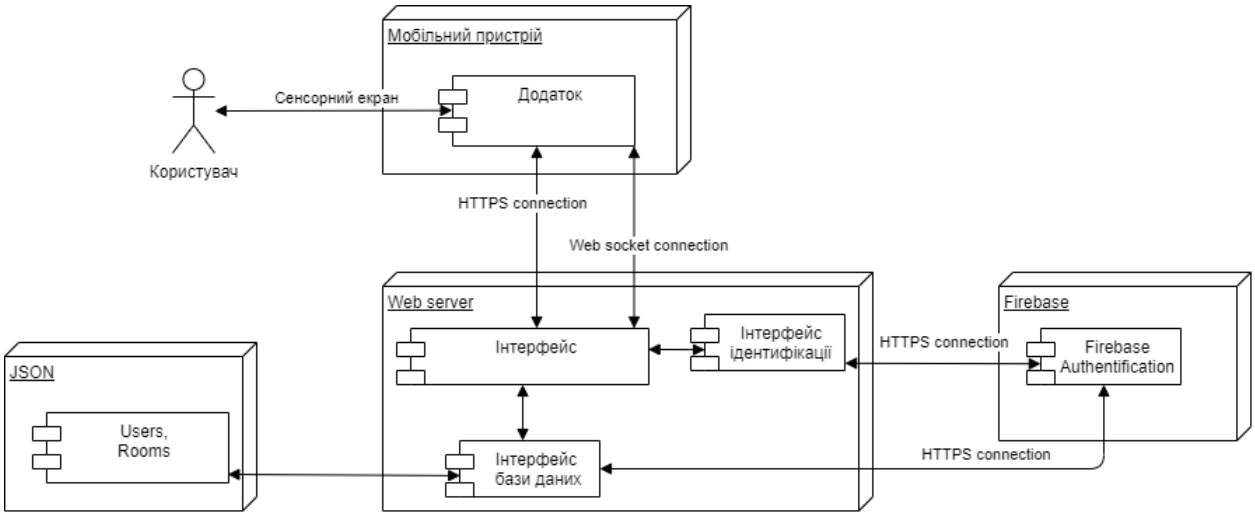


Рисунок 3.1 – Діаграма архітектури мобільного додатку

3.2 Програмна реалізація

Почнемо з того, що створив проект firebase

Переходимо на сайт Firebase , і створюємо проект. Далі переходимо в Authentication і натискаємо Get Started . Зі списку вибираємо Email/Password і активуємо перемичку, Google також активуємо перемичку і вибираємо пошту для допомоги. З налаштуванням firebase ми закінчили.

Наступним кроком буде створено проект

Дамо йому назву project , перезавантажено папку в VS Code. Відкриваємо консоль і пишемо наступні команди git init, npm install,npm і concurrently

Concurrently дозволить нам запустити сервер і клієнт з однієї директорії.

Далі нам потрібно буде створити наш package.json для одночасного використання . Для запуску сервера потрібно буде прописати — npm run dev --prefix server Для запуску клієнта потрібно буде прописати — npm run start — prefix client

"dev:client": "npm run start — prefix client",

"dev:server": "npm run dev — prefix server",

"dev": "concurrently \"npm run dev:client\" \"npm run dev:server\""

Наступним кроком буде створена структури проекту.

Створюємо папки Client і Server , відповідно в папці Client у нас буде наш проект React-native, а в Server буде проект Node.js. Так само додаємо файл .gitignore і прописуємо наступне node_modules

Далі переходимо до структури папки Сервер Структура сервера буде представлена наступним чином (рис. 3.2):

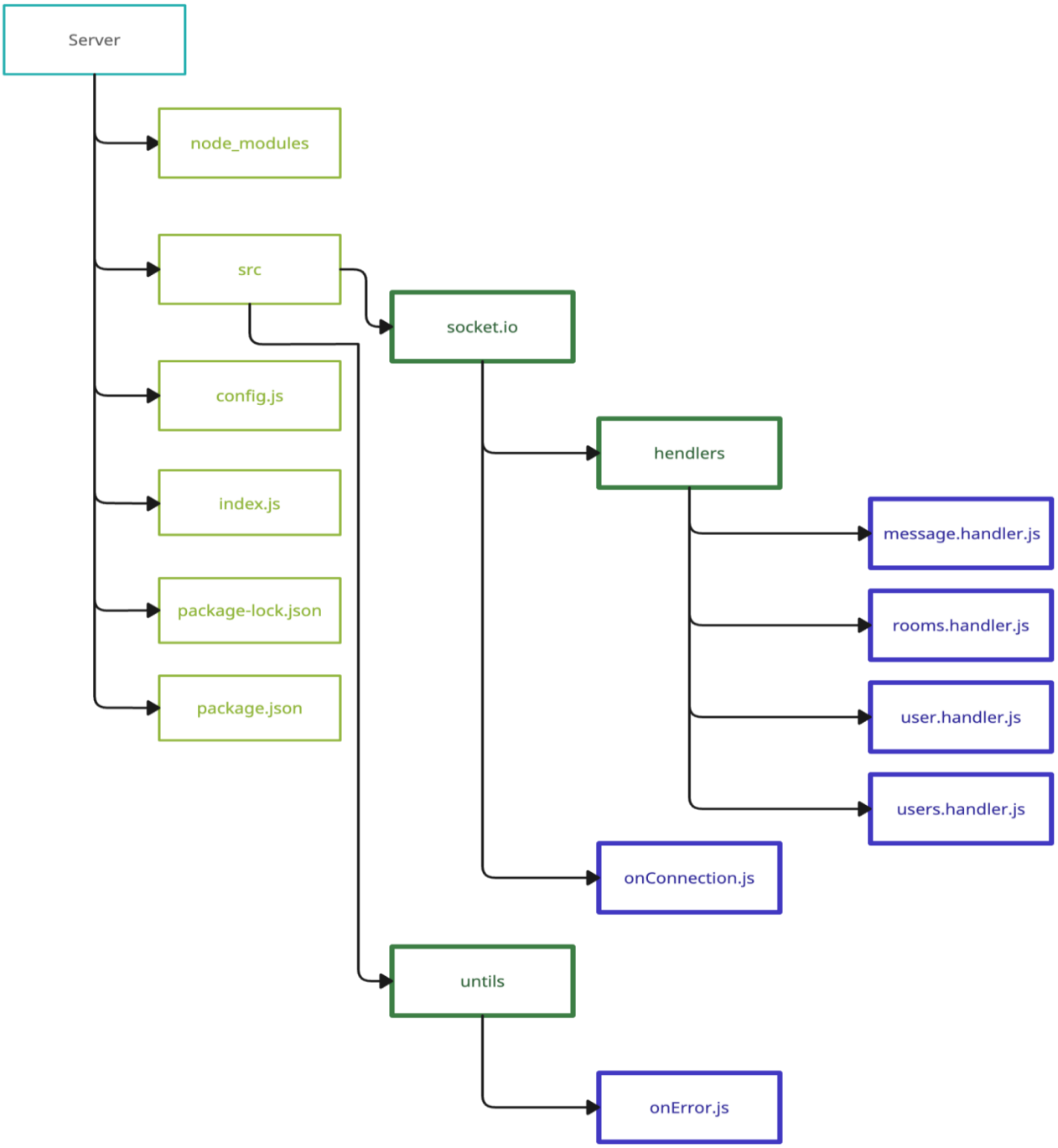


Рисунок 3.2 – Структура сервера

Відкриваємо консоль, за допомогою команди cd .\Server\ переходимо до папки нашого сервера. И прописуємо команду npm init Встановлюємо потрібні модулі npm i cors express lowdb socket.io

Використовуємо «lowdb», за допомогою цієї бібліотеки будемо імітувати нашу базу даних.

Переходимо до нашого package.json та додаємо наступні властивості(рис 3.3):

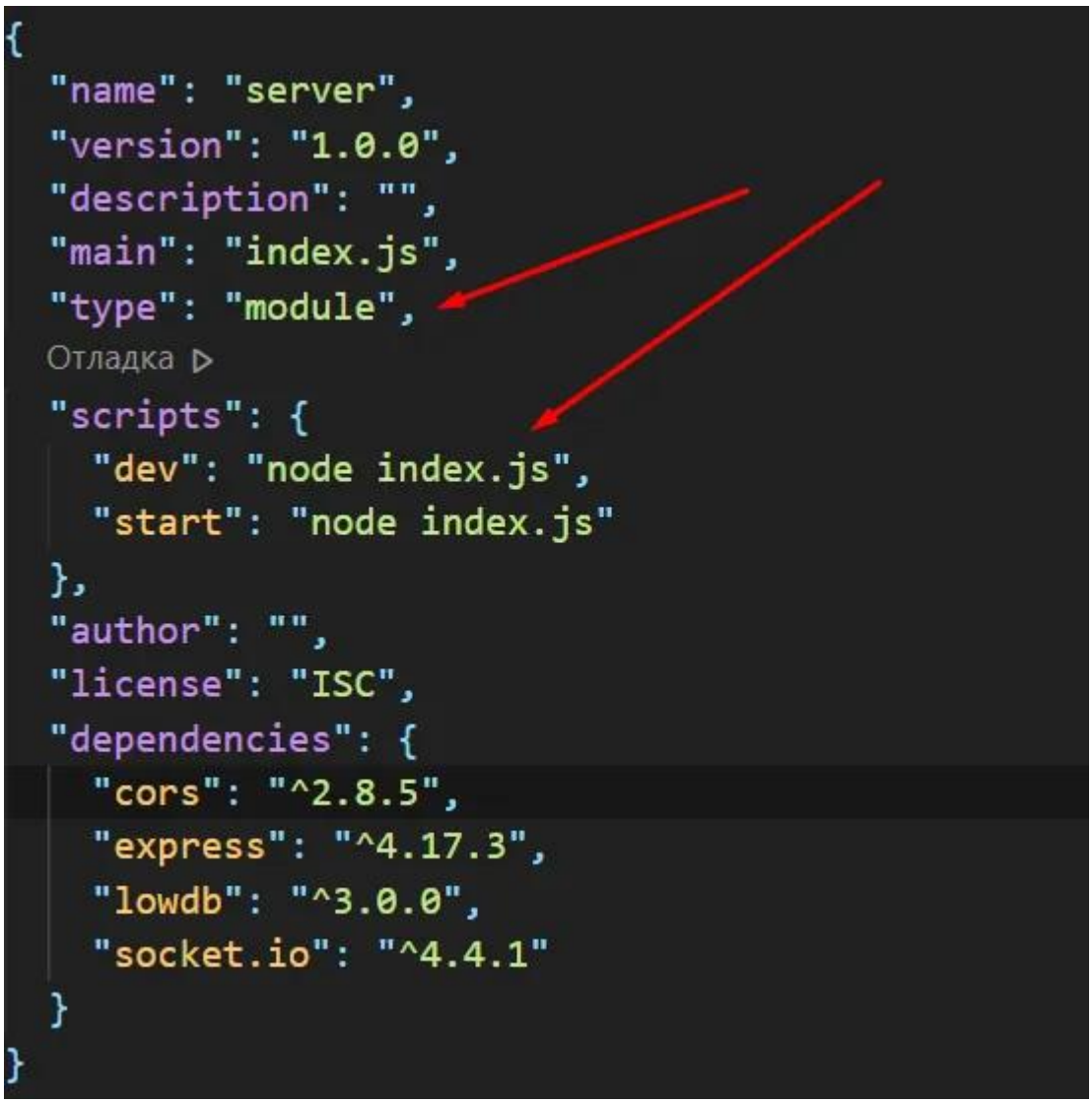


Рисунок 3.3 – Додання властивостей

Створюємо файл index.js в корневій папці сервера і прописуємо код, наданий у додатку В.

При успішному підключенні клієнта до нашого сервера викликаємо функцію onConnection. Так само в корні нашого проекту створюємо файл config.js, в якому прописуємо export const ALLOWED_ORIGIN = “http://localhost:4000".

Далі створіть папку src і в ній 2 папки socket.io і utils Почнемо з папки utils в якій створюємо файл onError.js. Код представлений у додатку В.

У папці socket.io створюємо файл onConection.js — це наша функція, яка буде викликана при успішному з’єднанні сокета з нашим сервером. У функції ми викликаємо обробники для різних подій, і так само обробляємо подію відключення сокета від сервера.

За допомогою socket.on(eventName, listener) додає функцію слухача в кінці масиву слухачів для події з іменем eventName . Якщо є eventName — це можна сказати id , який ми хочемо викликати на клієнті, і буде відпрацьовувати функцію на нашому сервері, так як ми будемо спілкуватися з нашим сервером.

Тепер створимо наші обробники для SOCKET

Створюємо обробники папки в папці socket.io , тут будуть зберігатися наші обробники. Спочатку з message.handlers.js створюємо його в папці обробників.

Тут ми реєструємо 2 події:

- Отримання повідомлень
- Додавання повідомлень

Так само ми тут використовуємо lowdb, про який я згадав вище. Ми будемо використовувати 3 методи db.read(), db.write(), db.data db.read() — потрібні для того, щоб встановити вміст db.data b.write () — потрібен для того, щоб записати дані в наш json db.data — потрібен для того, щоб отримати наші дані.

Так само ми використовуємо метод io.in(“roomName”).emit(eventName, data). Який передає підписникам наші дані (повідомлення)

Наступний розробник, який ми створили rooms.handlers.js

Тут ми реєструємо 2 події:

- Отримання кімнат
- Додавання кімнати

Тут ситуація схожа з message.handlers.js. Наступним обробником буде user.handler.js. Він відповідає за те, щоб наш підключений сокет зумів покинути та приєднатися до кімнати.

Тут ми використовуємо socket.join(roomId) і socket.leave(roomId).

Останнім обробником буде users.handler.js. Він відповідає за те, щоб ми змогли додати користувача до списку, керує його станом, на клієнті ми реалізуємо функціональність, щоб бачити, які користувачі зараз онлайн і офлайн. Також він відповідає за подію, коли користувач почав і перестав друкувати. Ми теж будемо це показувати нашому клієнту .

Тепер перейдемо до клієнта

Як вже вказувалось клієнта буде написано на React-native + expo + typescript. Він отримав досить великий, так що деякі прості моменти будуть упускатися. Я намагався писати максимально наближено до продакшну.

Переходимо в консоль і прописуємо наступне: expo init Client, потім зі списку обираємо blank typescript. Продемонстровано на рисунку 3.4:

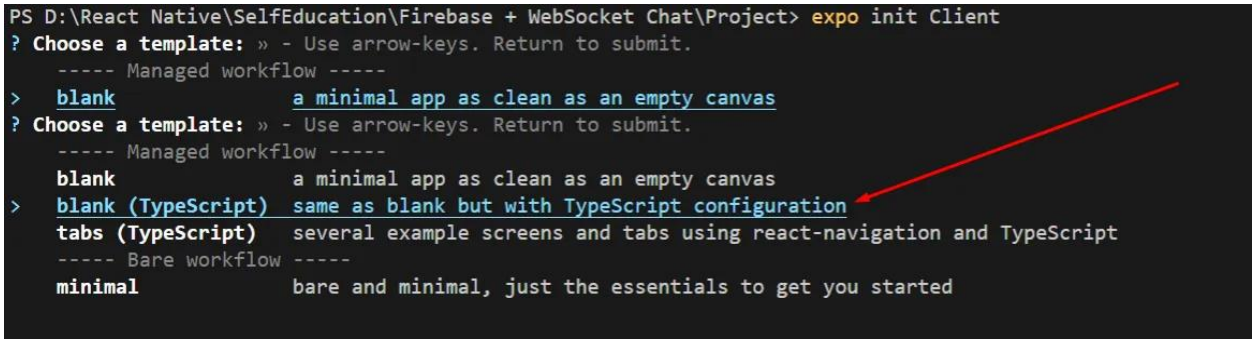


Рисунок 3.4 – Запуск клієнта

Далі необхідно внести деякі зміни в package.json і App.tsx. Переходимо в package.json і в поле «main» прописуємо App.tsx. Далі переходимо в App.tsx і реалізуємо код, наданий у додатку В.

Це все потрібно, щоб сказати нашому додатку, що App.tsx це вхід у наші додатки

Структура нашого додатка буде виглядати так(рис 3.5):

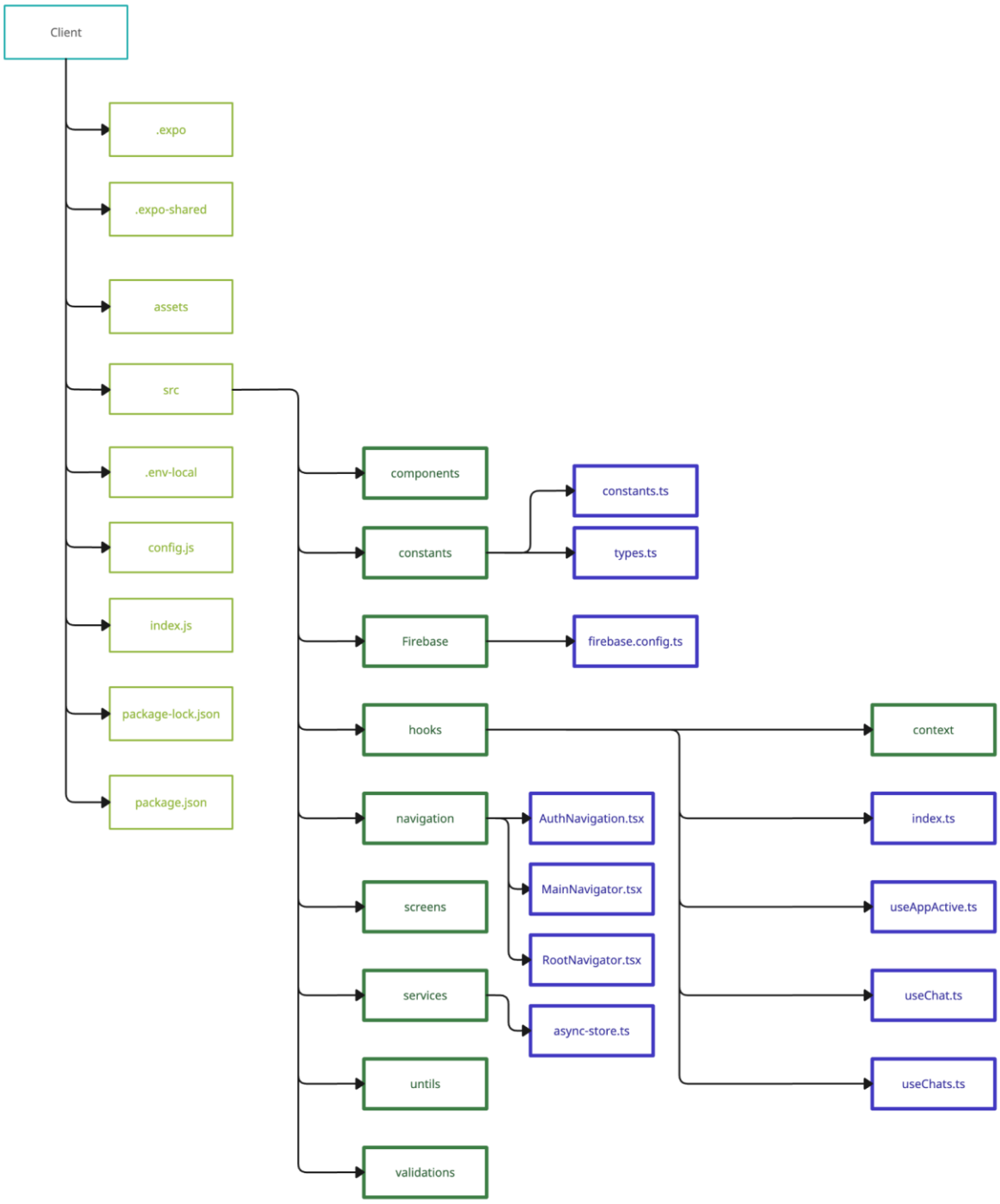


Рисунок 3.5 – Структура додатка

Переходимо до установки пакетів. Переходимо в консоль і встановлюємо наступні пакети:

expo-auth-session, expo-random, expo-web-browser, firebase, formik, native-base, react-native-dotenv, react-native-get-random-values, react-native-gifted-chat, react-native-safe-area-context, react-native-svg, socket.io-client, yup,

@react-native-async-storage/async-storage, @react-navigation/native,@react-navigation/native-stack

Переходимо на сайт Firebase натискаємо на огляд проекту і вибираємо підтримку веб-сайту(рис 3.6):

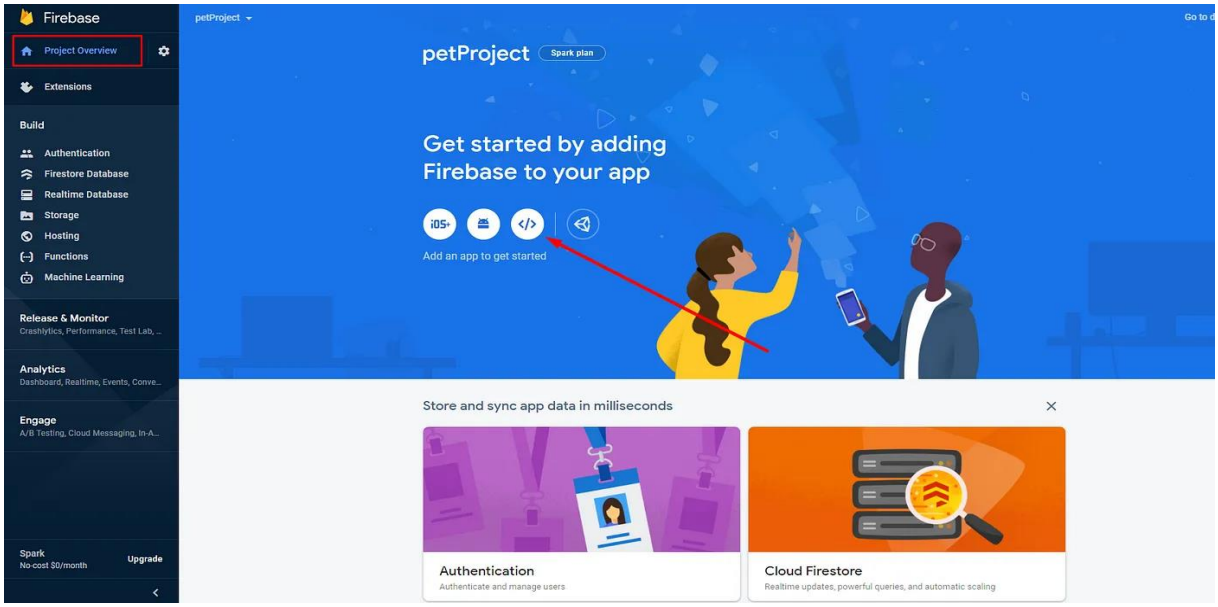


Рисунок 3.6 – Реєстрація продукту

Даємо назву проекту і натискаємо Register app.

У папці клієнта створіть файл .env.local, де будемо зберігати firebaseConfig . Прописуємо данні, за допомогою цих даних ми будемо підключатися до нашого сервера firebase:

```
REACT_APP_FIREBASE_API_KEY=Ваш_API_KEY
REACT_APP_FIREBASE_AUTH_DOMAIN=Ваш_AUTH_DOMAIN
REACT_APP_FIREBASE_PROJECT_ID=Ваш_PROJECT
REACT_APP_FIREBASE_STORAGE_BUCKET=Ваш_STORAGE_BUCKET
REACT_APP_FIREBASE_MESSAGING_SENDER_ID=Ваш_MESSAGING_SENDER
REACT_APP_FIREBASE_APP_ID=Ваш_APP_ID
REACT_APP_FIREBASE_MEASUREMENT_ID=Ваш_MEASUREMENT_ID
REACT_APP_SERVER_URL=http://localhost:5000
```

Щоб завантажити імпортувати значення з env потрібно відкрити файл babel.config.js і додати плагіни. Після цього потрібно оновити кеш, прописуємо командуexpo r -c

Далі створіть папку scr і в ній папку Firebase , у ній створіть файл firebase-config.ts — тут ми будемо ініціалізувати наш Firebase

Тепер перейдемо до частин, де будемо отримувати ключ API для авторизації через google у нашому додатку.

Переходимо на Google Cloud і обираємо наш проект firebase(рис 3.7 – 3.8):

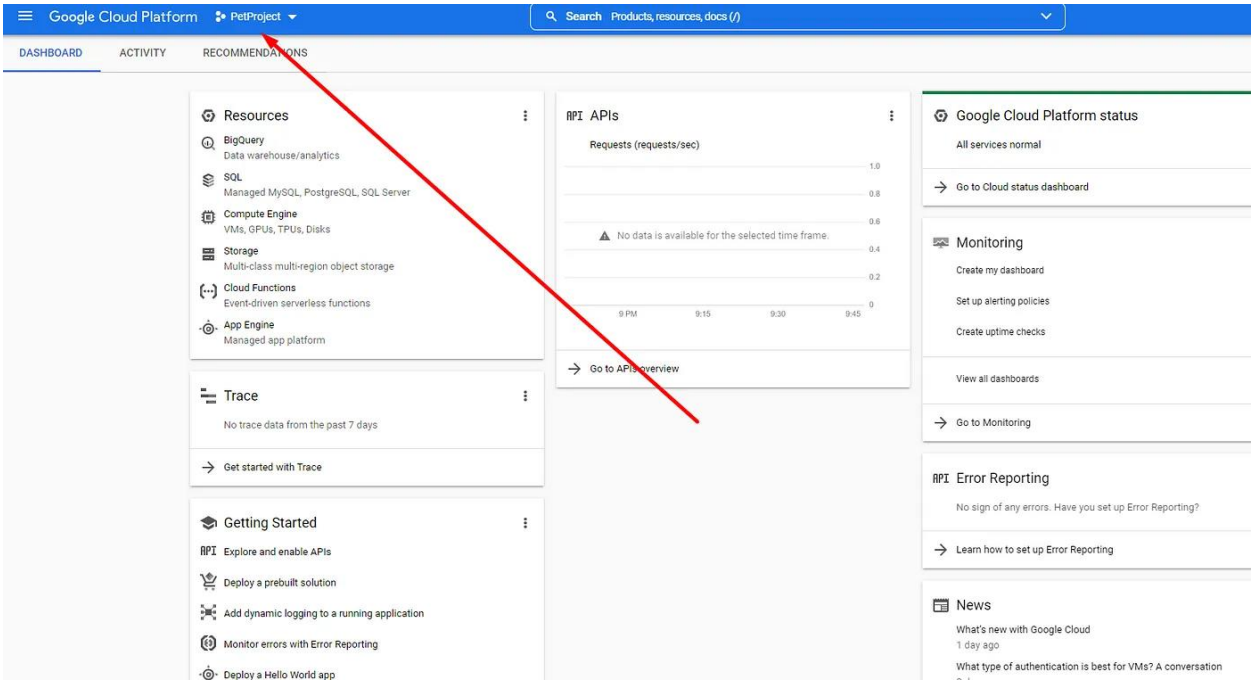


Рисунок 3.7 – Проекти на Google Cloud

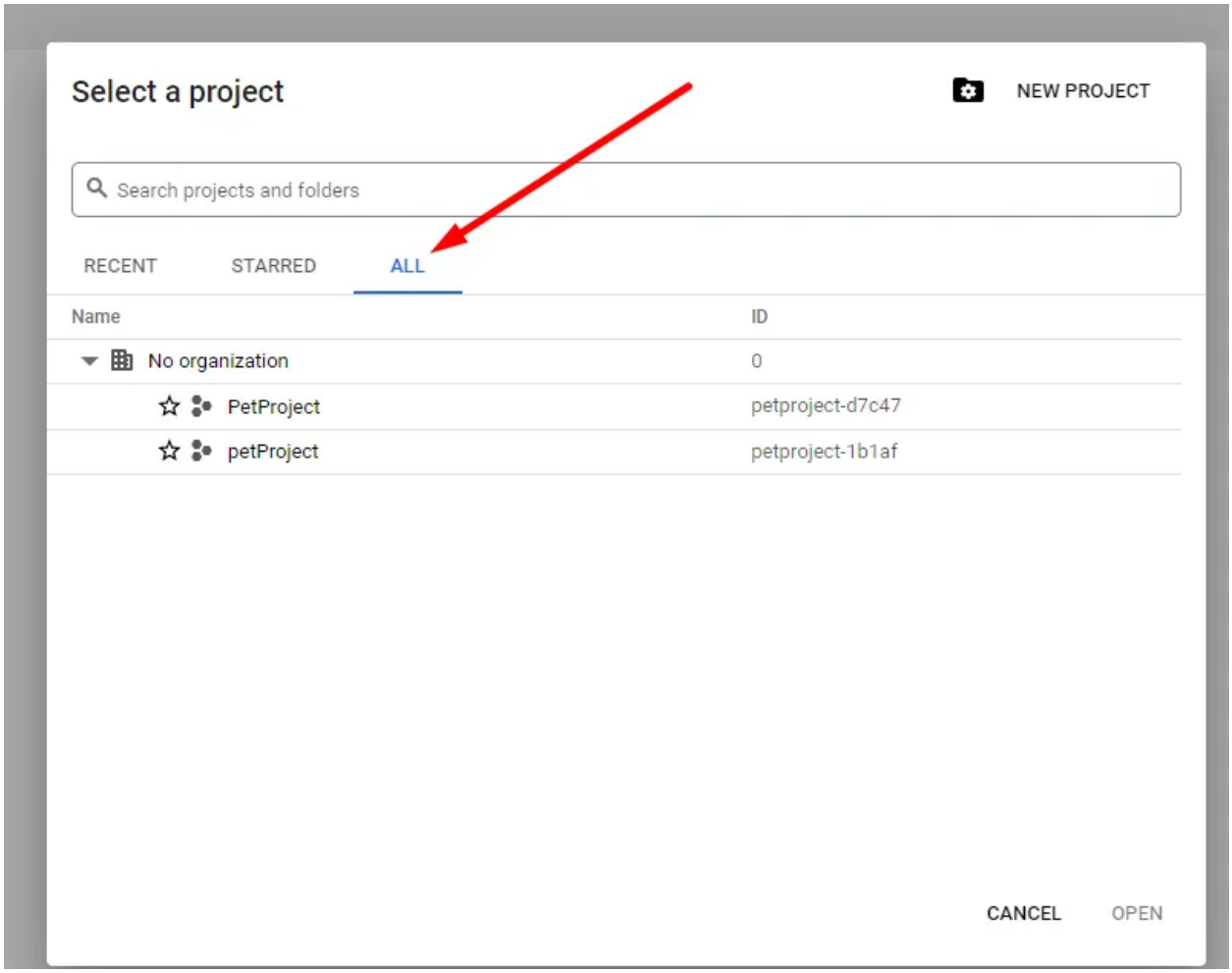


Рисунок 3.8 – Вибір проекта на Google Cloud

Тепер можна перейти в облікові дані(рис 3.9):

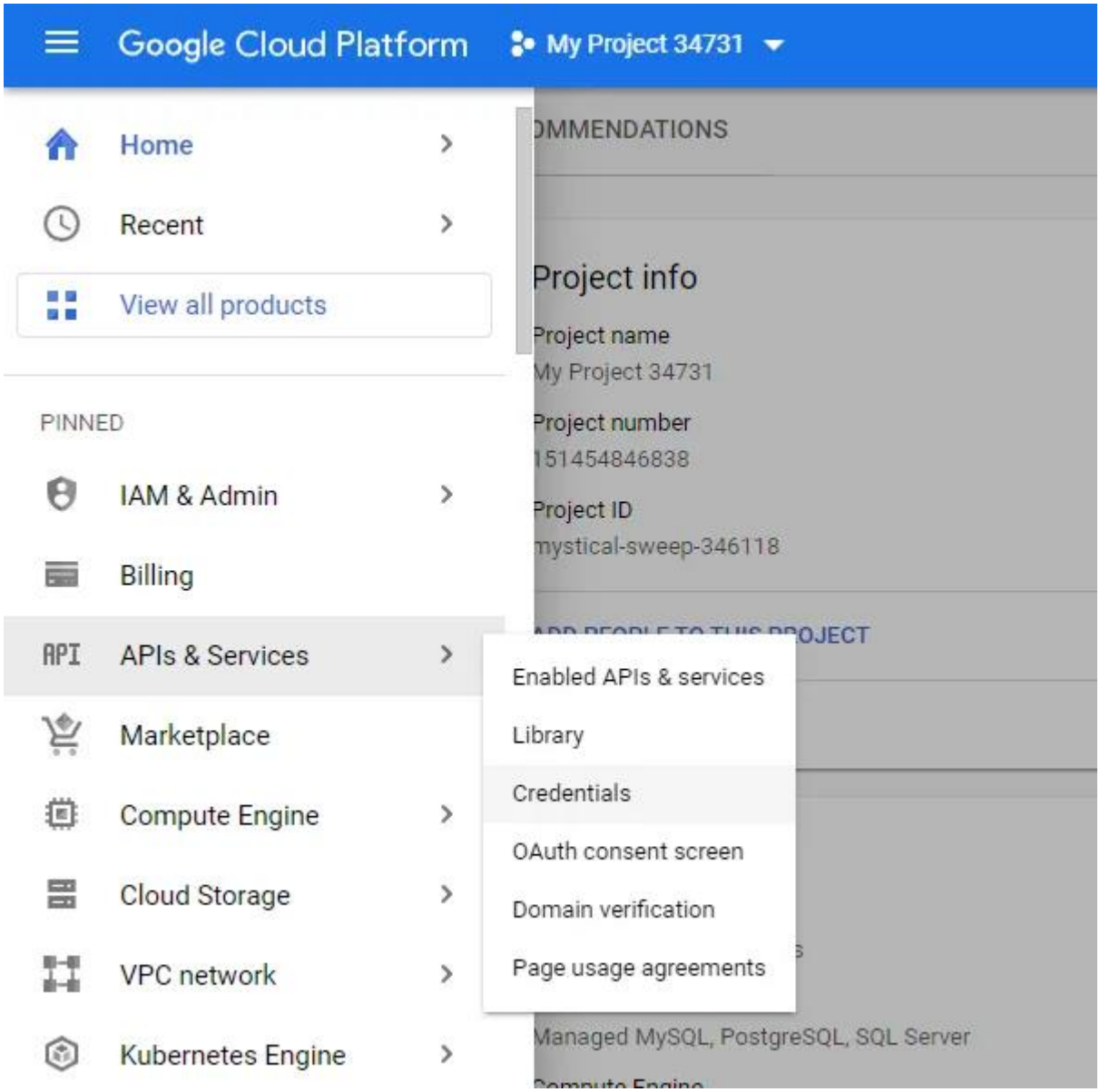


Рисунок 3.9 – Облікові дані Google Cloud

Створюємо нові облікові дані(рис 3.10):

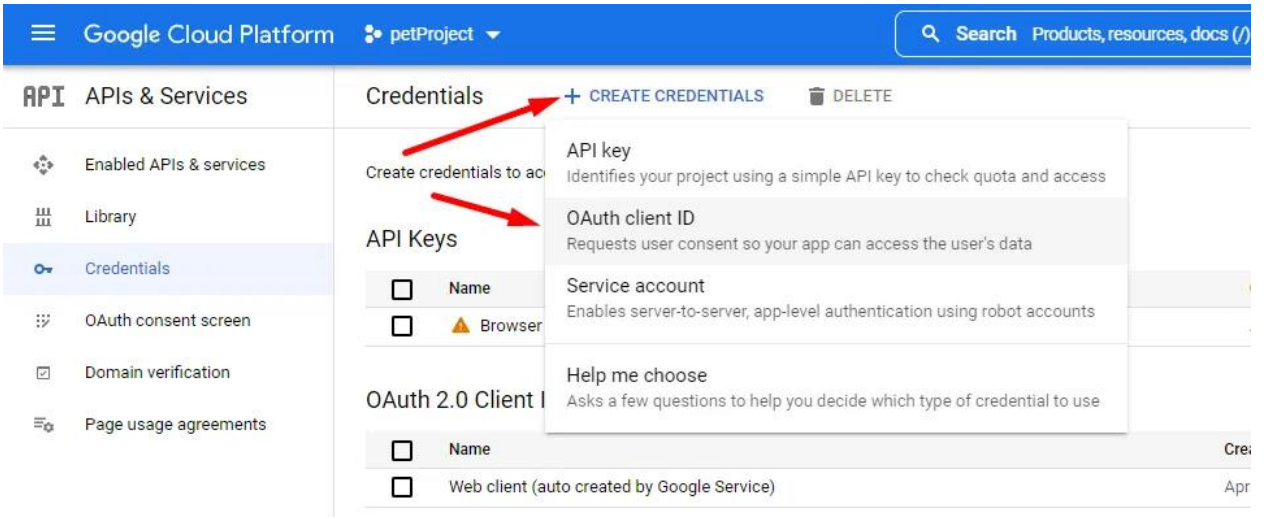


Рисунок 3.10 – Створення нових даних Google Cloud

Вибираємо веб-додаток , даємо йому назву Expo Go Proxy. В Авторизовані джерела JavaScript, додаємо https://auth.expo.io. В Авторизовані URI перенаправлення, додаємо https://auth.expo.io/@your-username/your-project-slug.

- @your-username — це ваше ім’я облікового запису в expo
- your-project-slug — це поле slug у файлі app.json

Контекст авторизації:

Переходимо до папки хуків, в ній створіть контекст папки У контексті папки створіть файл auth-context.tsx — цей контекст буде відповідати для зберігання нашого користувача, і так само там будуть усі функції, які пов’язані з ним. Код надано у додатку В.

WebSocketContext:

У цій же папці створюємо WebSocketContext.tsx, у якому будемо зберігати наш сокет і функції підключення, відключення від сервера та від кімнати.

URL для підключення до нашого сервера:

Щоб підключитися до нашого сервера достатньо вказати http://localhost:4000. Так це працює для веб-клієнта, але у цьому випадку використовуємо емулятор. Потрібно замість localhost вказати IP-адресу. Ip можна дізнатися про запис у консолі команди ipconfig. Без цього не можливо підключитися локально до сервера.

Тепер перейдемо до створення 3 хуків

1. useAppActive.ts — відповідає за стан нашого додатка (активно або додатків у фоновому режимі)

2. useChat.ts — відповідає за обробку та доставку наших повідомлень і так само отримання користувачів

3. useChats.ts — відповідає за отримання кімнат і їх створення і за стан користувача (онлайн, офлайн)

Також створивши index.ts, для зручного імпорту наших файлів. Код надано у додатку В.

Якщо проект з використанням typescript, то створюємо файл types.ts у папці констант, де будемо зберігати наші типи.

Бібліотека socket.io-client підтримує типізацію обробників, що нам на руку. Ми створюємо два інтерфейси — це ClientToServerEvents, другий — ServerToClientEvents. Так ми можемо протипірувати, що ми отримуємо від сервера і що відправляємо.

Компонент GiftedChat приймає:

1. messages — це наш масив повідомлень

2. onSend — функція для виправлення повідомлень

3. user — наш користувач

4. scrollToBottom — додає стрілочку в нижньому правому куті, щоб можна було проскролити наш чат до кінця

5. renderUsernameOnMessage — у вікні повідомлення з’являється ім’я. відправника

6. onInputTextChanged — потрібна нам функція, яка відповідає за відповідь на сервер події, користувач друкує

7. renderFooter — необхідний для відображення користувач, який друкує

З основними моментами закінчено, весь інший код розміщено у додатку В.

Додаток розміщуємо на сервер Heroku. Але Heroku раз в день очистить файли json до вихідного стану, так що наступного дня всі повідомлення зникнуть. Також можна підключити сторонню базу даних для зберігання повідомлень.

Переходим на сайт Heroku реєструємось і створюємо проект. Після цього нам потрібно встановити героя на наш комп'ютер. Переходимо по посиланню на сайт Heroku та виконуємо інструкції зображені на рисунку 3.11.

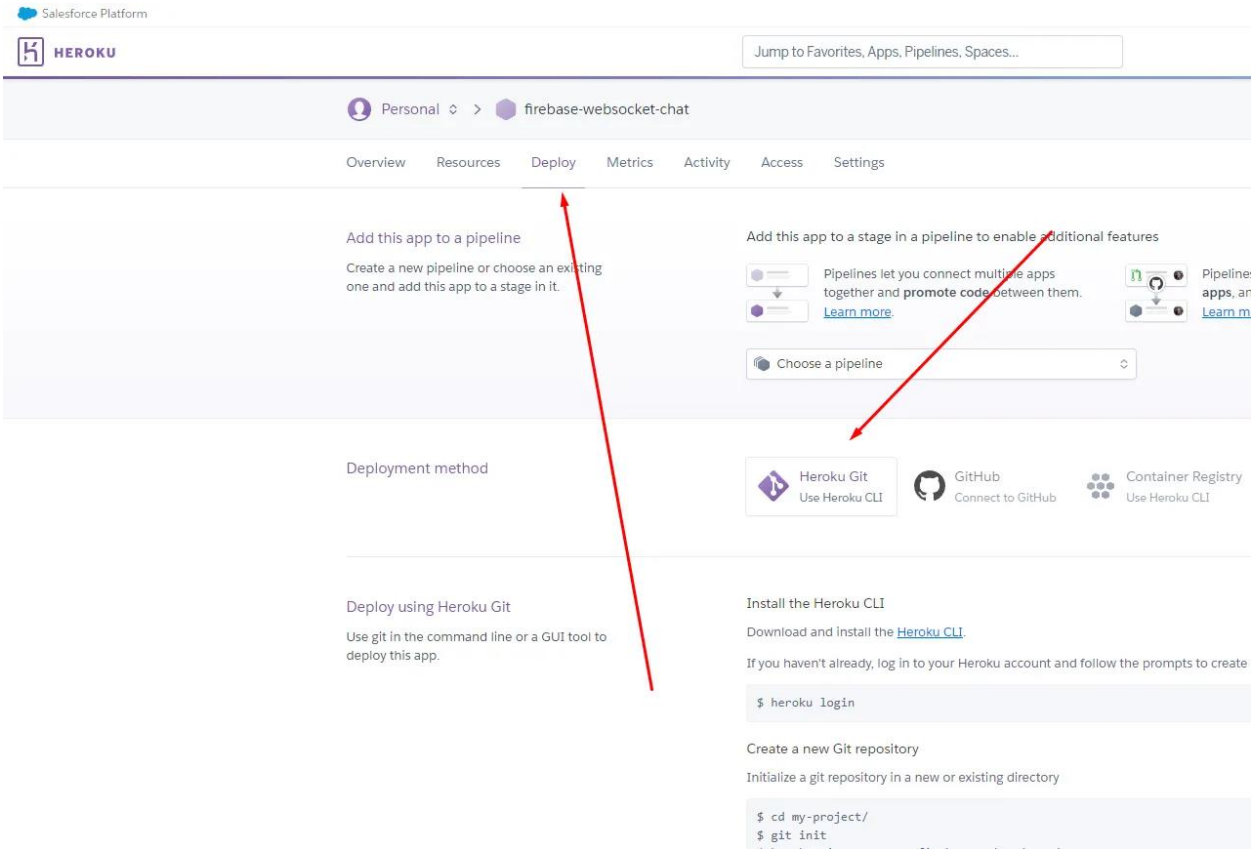


Рисунок 3.11 – Хостинг на Heroku

У проекті відкриваємо папку сервера в консолі та наступні інструкції. Після завантаження коду на сервер прописуємо команду, heroku logs --tail щоб переконатися, що все пройшло добре. Залишається тількт змінити посилання у файлі WebSocketContext.tsx, яку отримуємо після команди git push heroku master.

Останнім кроком буде зроблена пакування додатка в expo. Переходимо в консоль директорії клієнта і пишемо наступну команду «expo login». Після того, як ми отримали посилання на скачування apk. На ios завантажуємо додаток Expo Go і авторизуємось в обліковий запис Expo. Далі в консолі директорії клієнта прописуємо «expo publish», отримуємо Qr-код. Скануємо його за допомогою камери Iphone.

3.3 Тестування та працездатність

Під час розробки виникла потреба у тестуванні сервера. Для цього потрібно в консолі директорії нашого сервера прописати наступну команду: «node index.js». При правильному спрацюванні отримуємо наступне повідомлення «Server started on port 4000». Якщо так сталось то сервер працює.

Якщо є потреба у запуску сервера з директорії всього проекту, потрібно написати в консоль «npm run dev — prefix server».

ВИСНОВКИ

Під час виконання преддипломного проекту було досліджено розробку інтерактивного мобільного додатку для інформування користувачів про проведення заходів, який дозволить оптимізувати процес комунікації учасників заходу через мережу Інтернет.

Під час цього процесу було успішно виконано наступні завдання/задачі:

- дослідження предметної області;
- були визначені завдання проекту і основні функції в рамках цього дослідження;
- актуальність роботи;
- аналіз аналогів;
- спроектована модель та структура додатку;
- технології для розробки;
- сформована мета та постановка задачі.

Було проаналізовано розвиток месенджерів як різновиду нових медіа, їх частка у глобальному ринку мобільних додатків та взагалі популярність мобільних додатків у суспільстві.

Також було проведено аналіз обраних засобів для реалізації додатку. Основними з них є: JS, React Native.

Було встановлено, що всі досліджені додатки належать до одного жанру, але вони значно відрізняються за технологіями реалізації, можливостями та інтерфейсом.

У результаті проведеної практики були сформульовані завдання для поточного мобільного додатка з метою задоволення всіх вимог. Крім того, отримано нові знання щодо розробки програмних продуктів для мобільних систем.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Корпанюк Т. М., Мулик Я. І. Застосування мобільних додатків в бізнесі та їх облік. Ефективна економіка. [Електронний ресурс] – Доступ до ресурсу: http://www.economy.nayka.com.ua/pdf/3_2018/59.pdf (дата звернення: 11.02.2023);
2. Одноroженко Т. Світові економічні тенденції та ринок мобільних додатків. asomobile.net [Електронний ресурс] – Доступ до ресурсу: https://asomobile.net/uk/blog/svitovi-ekonomichni-tendenczi%D1%97-ta-rinok-mobilnih-dodatkiv/ (дата звернення: 12.02.2023);
3. Кулеш С. Рейтинг найпопулярніших в Україні мобільних додатків за вересень 2021 року. https://itc.ua/ [Електронний ресурс] – Доступ до ресурсу: https://itc.ua/news/rejting-mobilnih-dodatkiv-za-veresen-2021/ (дата звернення: 12.02.2023);
4. Adorno J. Telegram reaches 1 billion downloads globally. 9to5mac.com [Електронний ресурс] – Доступ до ресурсу: https://9to5mac.com/2021/08/30/telegram-reaches-1-billion-downloads-globally/ (дата звернення: 16.02.2023);
5. MTProto Mobile Protocol. core.telegram.org [Електронний ресурс] – Доступ до ресурсу: https://core.telegram.org/mtproto (дата звернення: 16.02.2023);
6. Viber. www.viber.com [Електронний ресурс] – Доступ до ресурсу: https://www.viber.com/ua/ (дата звернення: 17.02.2023);
7. Signal. signal.org [Електронний ресурс] – Доступ до ресурсу: https://signal.org/uk/ (дата звернення: 18.02.2023);
8. What is IDEF - Definition, Methods, and Benefits/Edraw. [Електронний ресурс] – Режим доступу до ресурсу: URL:https://www.edrawsoft.com/what-is-idef.html (дата звернення: 18.03.2023);

9. Моделювання процесу управління стратегічною гнучкістю підприємства/Електронний журнал «Ефективна економіка». [Електронний ресурс] – Режим доступу до ресурсу: URL: http://www.economy.nayka.com.ua/?op=1&z=1729#:~:text=IDEF0%20%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C%20%D1%81%D0%BA%D0%BB%D0%B0%D0%B4%D0%B0%D1%94%D1%82%D1%8C%D1%81%D1%8F%20%D0%B7,%D1%82%D0%B0%20%D0%B0%D1%81%D0%BE%D1%86%D1%96%D0%B9%D0%BE%D0%B2%D0%B0%D0%BD%D1%96%20%D0%B7%20%D0%BD%D0%B8%D0%BC%D0%B8%20%D0% (дата звернення: 19.03.2023);
10. Застосування UML в дипломних роботах dut.edu.ua [Електронний ресурс] – Доступ до ресурсу: https://dut.edu.ua/ua/news-1-626-7758-zastosuvannya-uml-v-diplomnih-robotah_kafedra-kompyuternih-nauk-ta-informaciynih-tehnologiy (дата звернення: 16.05.2023)
11. React Native Documentation. reactnative.dev [Електронний ресурс] – Доступ до ресурсу: https://reactnative.dev/docs/getting-started (дата звернення: 11.04.2023)
12. Expo Documentation. docs.expo.dev [Електронний ресурс] – Доступ до ресурсу: https://docs.expo.dev/?utm_source=google&utm_medium=cpc&utm_content=search&gclid=CjwKCAjw1YCkBhAOEiwA5aN4ATB0SHp9Y3vMPD4SJB_ukCJVu6ur_NeyNVbgWhsj6gOPSSzJ74ra7BoCxZkQAvD_BwE (дата звернення: 13.04.2023)
13. TypeScript Documentation. typescriptlang.org [Електронний ресурс] – Доступ до ресурсу: https://www.typescriptlang.org/docs/ (дата звернення: 15.04.2023)
14. Firebase документація. firebase.google.com [Електронний ресурс] – Доступ до ресурсу: https://firebase.google.com/docs?hl=ua (дата звернення: 18.04.2023)

15. Socket.io Documentation. socket.io [Електронний ресурс] – Доступ до ресурсу: https://socket.io/docs/v4/ (дата звернення: 19.04.2023)
16. Node.js Documentation. nodejs.org [Електронний ресурс] – Доступ до ресурсу: https://nodejs.org/en/docs (дата звернення: 19.04.2023)
17. Express Documentation. expressjs.com [Електронний ресурс] – Доступ до ресурсу: https://expressjs.com/en/guide/routing.html (дата звернення: 22.04.2023)
18. Діаграма розгортання. studfile.net [Електронний ресурс] – Доступ до ресурсу: https://studfile.net/preview/5010027/page:6/ (дата звернення: 29.03.2023)
19. Архітектура програмного забезпечення. wezom.com.ua [Електронний ресурс] – Доступ до ресурсу: https://wezom.com.ua/ua/blog/arhitektura-programmnogo-obespecheniya (дата звернення: 02.04.2023)
20. Архітектура та проектування програмного забезпечення. dspace.wunu.edu.ua [Електронний ресурс] – Доступ до ресурсу: http://dspace.wunu.edu.ua/jspui/bitstream/316497/24194/1/%D0%BE%D0%BF%D0%BE%D1%80%D0%BD%D0%B8%D0%B9%20%D0%BA%D0%BE%D0%BD%D1%81%D0%BF%D0%B5%D0%BA%D1%82%20%D0%BB%D0%B5%D0%BA%D1%86%D1%96%D0%B9.pdf (дата звернення: 02.04.2023)
21. Папроцький І. А. Огляд сучасних підходів до розробки мобільних застосунків для бізнес застосувань. ekmair.ukma.edu.ua [Електронний ресурс] – Доступ до ресурсу: https://ekmair.ukma.edu.ua/server/api/core/bitstreams/9db68fe2-66e3-473d-a5b4-257123f93785/content (дата звернення: 12.03.2023)

ДОДАТОК А.

МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Технічне завдання на створення мобільного додатку інформування про проведення заходів

ПОГОДЖЕНО:

Доцент кафедри комп’ютерних наук

_____ Чибіряк Я. І.

Студент групи ІТ-91

_____ Касьяненко Д. Р.

2023

1 Призначення й мета створення мобільного додатку

1.1 Призначення мобільного додатку

Мобільний додаток має реалізувати повне функціонування поставлених задач та надавати можливість спілкуватися онлайн.

1.2 Мета створення мобільного додатку

Метою даного дослідження є розробка мобільного додатку для інформування щодо проведення заходів, який матиме авторизацію та чати з користувачами.

1.3 Цільова аудиторія

До цільовою аудиторії додатку можна віднести людей майже всіх вікових груп, які вже вміють користуватися мобільним пристроєм.

2 Вимоги до мобільного додатку

2.1 Вимоги до мобільного додатку в цілому

2.1.1 Вимоги до структури й функціонування мобільного додатку

Мобільний додаток має бути доступним за допомогою файла завантаження в мережі Інтернет. Месенджер повинен складатися із взаємозалежних розділів із чітко розділеними функціями.

2.1.2 Вимоги до персоналу

Персоналом є технічний розробник, який має володіти особливими навичками у розробці та досвід з проектами з нуля і підтримки в production. Технічні навички ES6, JS, React-Native, hooks, Redux, Firebase та досвід з IOT.

2.1.3 Вимоги до збереження інформації

Уся інформація надана у додатку буде зберігатися у базі даних реалізованій засобами платформою для розробки мобільних додатків Firebase.

2.1.4 Вимоги до розмежування доступу

Розроблюваний додаток має бути загальнодоступним для завантаження у мережі Інтернет. Права доступу до інформації розмежовані за групами користувачів: технічний розробник та користувачі. Технічний розробник має необмежений доступ до даних бази даних, вихідного коду з можливістю їх зміни/видалення/додання.

Користувач додатку має доступ до всіх вікон та можливостей додатку, з можливістю листуватися з іншими користувачами, змінити персональні дані та перегляд статусу інших користувачів.

Розробник має доступ до програмного коду, бази даних користувачів з можливістю змінювати, додавати чи видаляти їх.

2.2 Структура мобільного додатку

2.2.1 Загальна інформація про структуру мобільного додатку

Структура мобільного додатку являє собою набір сторінок. Такими розділами є: Головна – на сторінці зображені чати з іншими користувачами та можливістю перейди до вікна чату.

Чат – на сторінці можна переглянути раніше створені повідомлення, написати нове, перегляд кількості користувачів та їх профілю

Початкова сторінка/Профіль – сторінка з персональними даними користувача та можливістю їх зміни чи оновлення

2.2.2 Навігація

Основна навігація реалізована на початковій сторінці за допомогою кнопок вибору дій. Також на головній сторінці та сторінці чату є повернення до початкового екрану.

2.2.3 Наповнення мобільного додатку

Для управління контентом додатку буде використана система React Native.

Заповнення та редагування контенту мобільного додатку має бути через програмний код, використовуючи інформацію з бази даних.

Всю інформацію для наповнення мобільного додатку має надавати керівник проекту, включаючи всі структурні та стилістичні рішення.

2.2.4 Дизайн та структура додатку

Стиль мобільного додатку було обрано у мінімалістичний дизайн, кольори обрані поєднувані між собою, приємні для ока та чіткі у розборі повідомлень.

Розташування елементів на головній та початковій сторінці додатку показано на рисунках А.1-А.2.

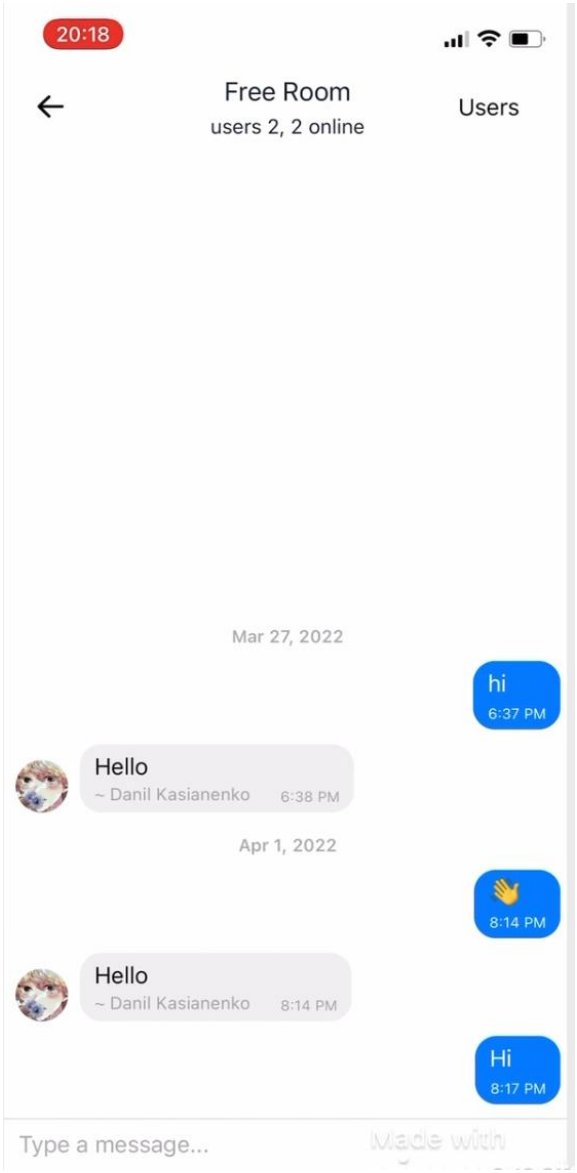


Рисунок А.1 – Вигляд вікна чату

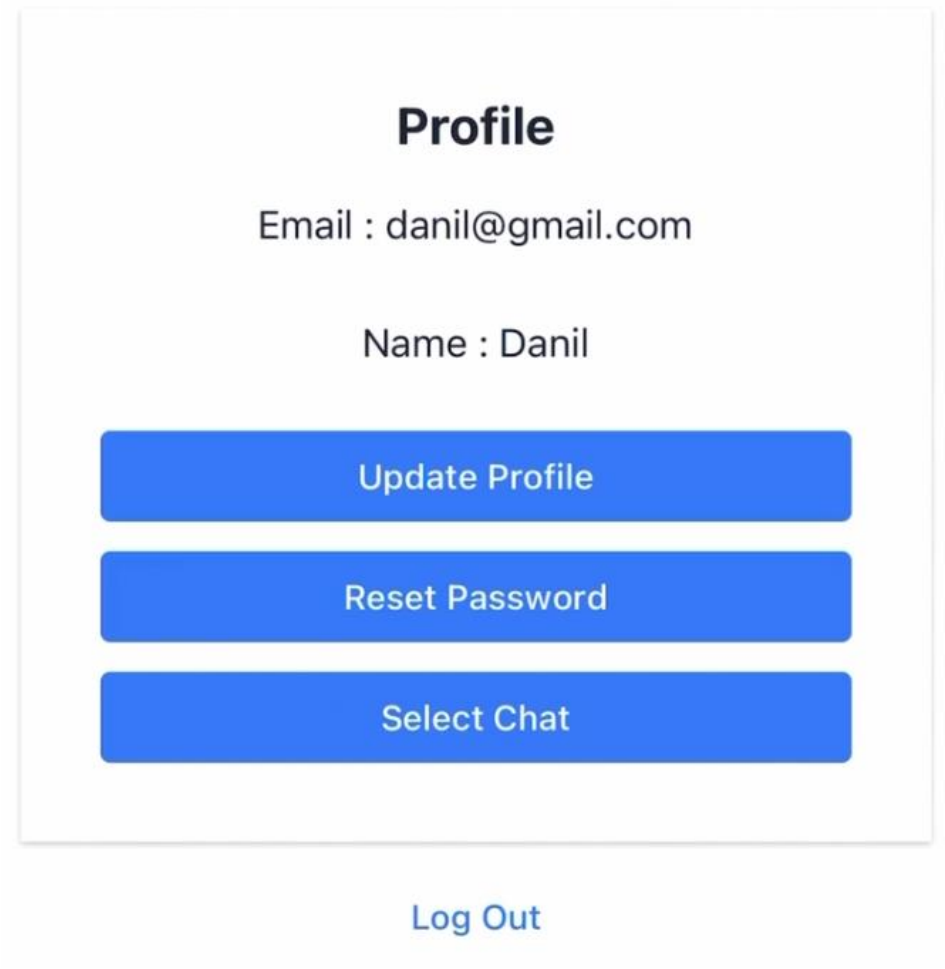


Рисунок А.2 – Вигляд вікна профілю

2.2.5 Система навігації

Карта мобільного додатку зображена на рисунку А.3.

Рисунок А.3 – Карта додатку

2.3 Вимоги до функціонування системи

2.3.1 Потреби користувача

Потреби користувача наведені у таблиці А.1.

Таблиця А.1 – Потреби користувача

ID	Потреби користувача	Джерело
UN-01	Функціонування авторизації	Користувач
UN-02	Працездатність чату	Користувач
UN-03	Зміна чи оновлення персональних даних	Користувач
UN-04	Перегляд чатів та їх участників	Користувач
UN-05	Відображення статусу участників чату	Користувач
UN-06	Зміна, видалення та додання програмного коду	Технічний розробник
UN-07	Перегляд, зміна, видалення та додання інформації у БД	Технічний розробник

2.3.2 Функціональні вимоги

Проаналізувавши потреби користувачів платформи дистанційного навчання, було визначено наступні вимоги:

- Реєстрація та авторизація користувачів;
- Відображення персональних даних та можливість їх зміни;
- Можливість листування у вікні чату;
- Відображення всіх доступних чатів;
- Можливість перегляду участників чату.

2.3.3 Системні вимоги

Даний розділ визначає, розподіляє та вказує на системні вимоги, визначені розробником. Їх перелік наведений в таблиці А.2.

Таблиця А.2 – Системні вимоги

ID	Системні вимоги	Пріоритет	Опис
SR-01	Вікно авторизації користувача	S	Надає користувачу можливість користування додатком після звірення даних щодо користувача за допомогою БД
SR-02	Чат користувачів	M	Повне функціонування функції листування, відображення повідомлень
SR-03	Персональні данні	S	Відображення даних користувача та можливість їх зміни
SR-04	Вікно перегляду всіх чатів	M	Відображення всіх чатів у вікні з можливістю перейти в них
SR-05	Вікно перегляду користувачів та їх інформації	S	Відображення користувачів у чаті з їх іменем та статусом

Умовні позначення в таблиці А.2:

Must have (M) – вимоги, які повинні бути реалізовані в системі;

Should have (S) – вимоги, які мають бути виконані, але вони можуть почекати своєї черги;

Could have (C) – вимоги, які можуть бути реалізовані, але вони не є центральною ціллю проекту.

2.4 Вимоги до видів забезпечення

2.4.1 Вимоги до інформаційного забезпечення

Для створення мобільного додатку використовується:

- React-native;
- Expo client;
- TypeScript;
- Firebase;
- Socket.io;
- Node.js;
- Express.

2.4.2 Вимоги до лінгвістичного забезпечення

Мобільний додаток використовує англійську мову.

2.4.3 Вимоги до програмного забезпечення

Програмне забезпечення клієнтської частини повинне задовольняти наступним вимогам:

- Android версії 4.4 та вище;
- IOS 7+.

3 Склад і зміст робіт зі створення web-додатку

Послідовність створення ігрового додатку наведена в таблиці А.3.

Таблиця А.3 – Етапи створення ігрового додатку

№	Склад і зміст робіт	Строк розробки
1	Постановка цілей необхідних для досягнення певного результату	2 дні
2	Складання технічного завдання	2 дні
3	Створення структури проекту	5 днів
4	Створення сервера	2 дні
5	Створення обробника повідомлень	7 днів
6	Створення чатів	5 днів
7	Тестування серверу	2 дні
8	Створення аутентифікації клієнта	4 дні
9	Підключення до БД	1 день
10	Створення інтерфейса додатку	6 днів
11	Beta-тестування	8 днів
12	Alpha-тестування	5 днів
13	Перевірка працездатності	2 дні
14	Написання супровідної документації	2 дні
15	Реліз моібльного додатку	1 день
	Загальна тривалість робіт	68 днів

4 Вимоги до складу й змісту робіт із введення мобільного додатку в експлуатацію

Продукт розробляється ітеративно із урахуванням принципів та технологій уніфікованого процесу розроблення програмного забезпечення. Додаток повинен бути розроблений з використанням мови JavaScript, а саме стандарт ES6, додатковим використанням фреймворку React-Native та платформи для створення мобільних додатків Firebase. Клієнт повинен мати повне функціонування визначених можливостей та працездатність вимог. Для того щоб користувачі могли використовувати клієнт необхідно розмістити додаток на сервері з параметрами необхідними для коректної працездатності.

ДОДАТОК Б.

Планування робіт

Деталізація мети проекту методом SMART. Продуктом дипломного проекту є мобільний додаток «Connect People».

Результати деталізації мети даного проекту представлено в таблиці Б.1.

Таблиця Б.1 – Деталізація мети проекту методом SMART

Specific	Створити додаток месенджеру, для спілкування людей через мережу Інтернет
Measurable	Результатом роботи проекту є збільшення кількості користувачів до 50 тис в день.
Achievable	Мета досяжна, є узгоджена тема проекту та навички у роботі з середовищами розробки мобільних додатків
Relevant	Додаток дозволить користувачу ознайомитись з клієнтом та його можливостями. Розвиток власних навичок розробника у розробці мобільних додатків
Time-framed	30 травня 2023 року.

Планування змісту структури робіт. Структурна декомпозиція робіт (work breakdown structure, WBS) – це ієрархічна структура робіт, побудована з метою логічного розподілу усіх робіт з виконання проекту і подана у графічному вигляді. Це сукупність декількох рівнів, кожний з яких формується в результаті розподілу роботи поперед-нього рівня на її складові.

Елементом найнижчого рівня є група робіт, або так званий робочий пакет (work package). WBS, створення гри «Shoot Them Up» представлений на рис. Б.1

Планування структури організації, для впровадження готового проекту (OBS). Наступним кроком розробки структури проекту є визначення організаційної структури (OBS) проекту. Організаційна структура проекту (OBS) – є графічним відображенням учасників проекту (фізичних та юридичних осіб) та їхніх відповідальних осіб, залучених до реалізації проекту. На верхньому рівні OBS проекту знаходиться керівник та команда управління проектом; на наступному рівні – виконавці. Останнім рівнем OBS-структури є відповідальні особи виконавців. Це не обов’язково повинні бути керівники, а ті співробітники, яким доручено безпосередньо організовувати і відповідати перед виконавцем за виконання конкретного елемента WBS-структури. OBS, Діаграма OBS представлена на рис. Б.2. Список виконавців, що функціонують в проекті знаходиться в табл. Б.2.

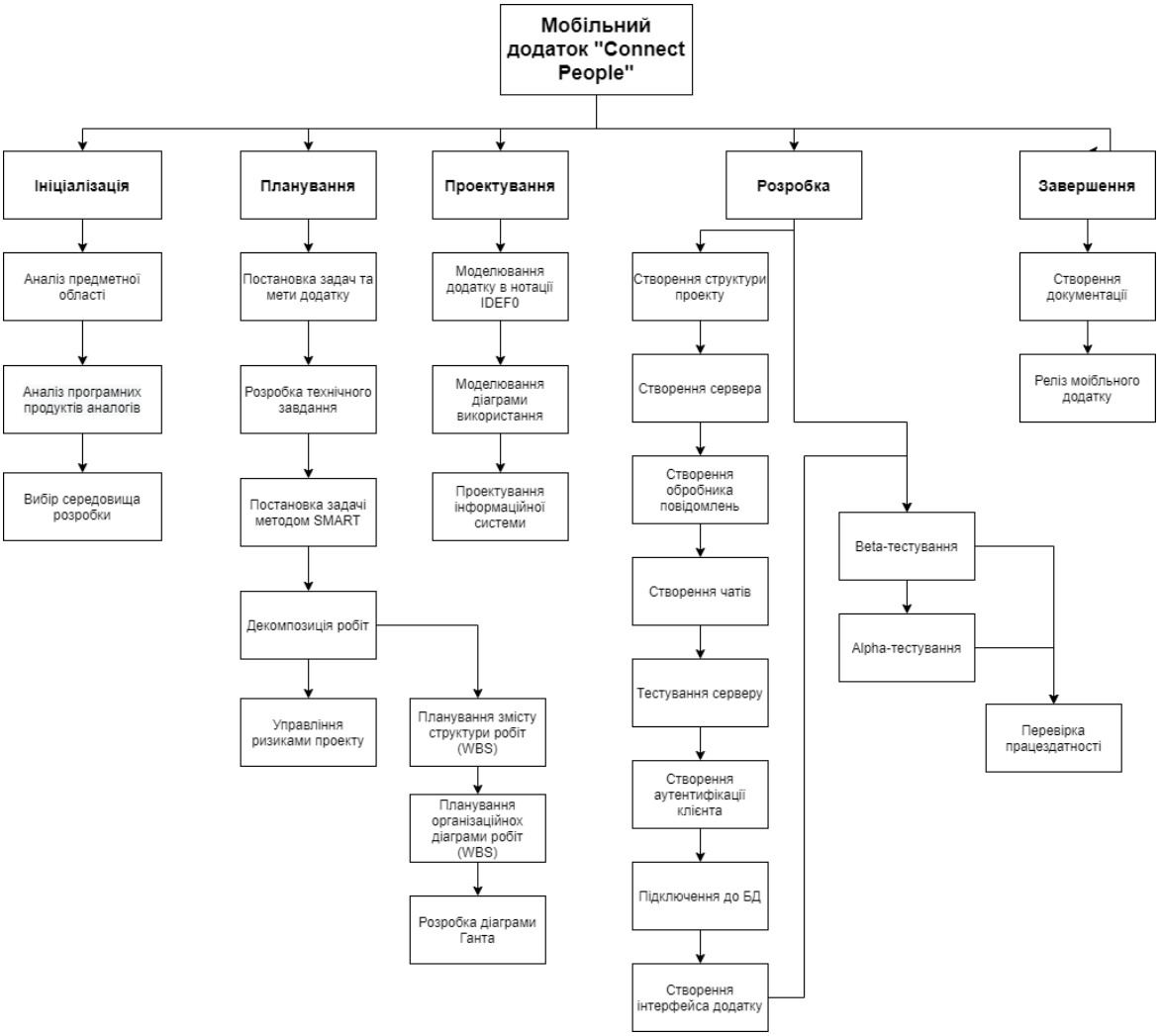


Рисунок Б.1 - WBS-структура проекту

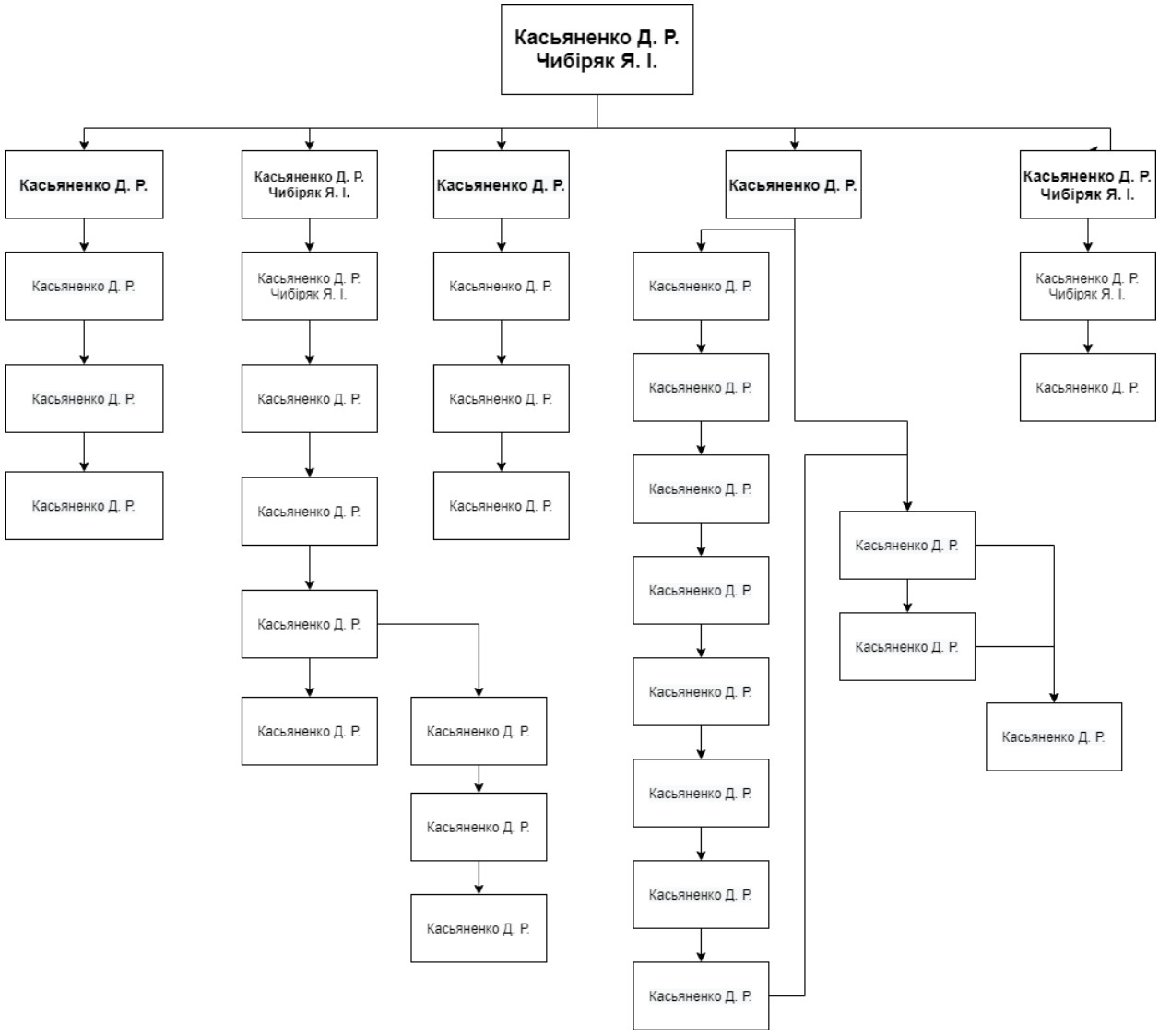


Рисунок Б.2 - OBS-структура проекту

Таблиця Б.2 – Виконавці проекту

Роль	**Ім’я**	**Проектна роль**
Розробник	Касьяненко Д. Р.	Виконує розробку додатку
Проектувальник	Касьяненко Д. Р.	Виконує проектування структури додатку
Тестувальник	Касьяненко Д. Р.	Відповідає за тестування додатку

Продовження таблиці Б.2 – Виконавці проекту

Керівник проекту	Чибіряк Я. І.	Формує завдання проекту
Менеджер проекту	Касьяненко Д. Р.	Виконує аналіз даних та відповідає за розподіл ресурсів, стежить за виконанням термінів

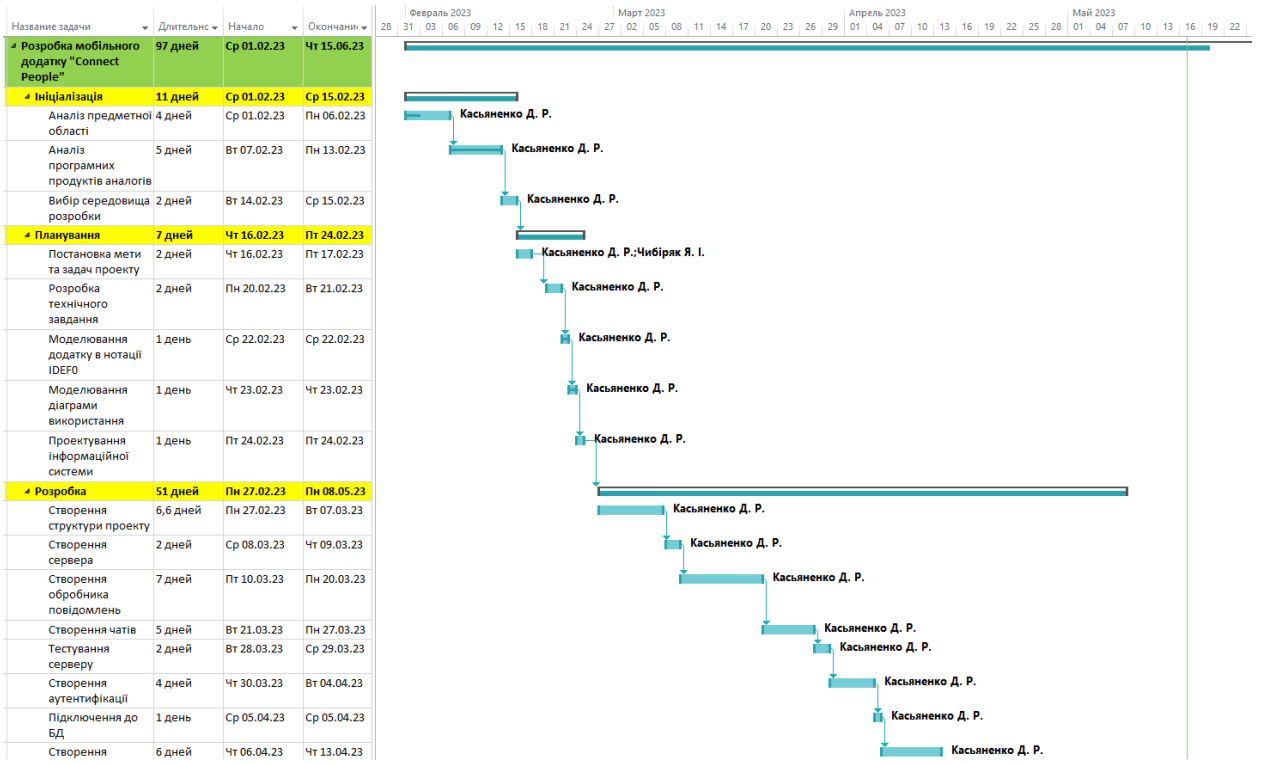
Діаграма Ганта. Діаграма Ганта – це популярний вид діаграми який використовується для планування і контролю виконання проекту. Такий інтерактивний мережевий графік присутній практично у всіх системах управління проектами. На діаграмі відображаються завдання і стадії проекту з урахуванням їх часу виконання. Завдання на діаграмі можуть бути залежними один від одного (наприклад, одна задача може починатися тільки після завершення другого). Крім того, може показуватися відсоток виконання кожного завдання і відповідальний за її виконання. Діаграма Ганта представлена на рис. Б.3-Б6

Название задачи	Длительнс	Начало	Окончани	Предшественн	Названия ресурсов
Розробка мобільного додатку "Connect People"	**97 дней**	**Ср 01.02.23**	**Чт 15.06.23**		**Касьяненко Д. Р.;Чибіряк Я. І.**
Ініціалізація	**11 дней**	**Ср 01.02.23**	**Ср 15.02.23**		**Касьяненко Д. Р.**
Аналіз предметної області	4 дней	Ср 01.02.23	Пн 06.02.23		Касьяненко Д. Р.
Аналіз програмних продуктів аналогів	5 дней	Вт 07.02.23	Пн 13.02.23	3	Касьяненко Д. Р.
Вибір середовища розробки	2 дней	Вт 14.02.23	Ср 15.02.23	4	Касьяненко Д. Р.
Планування	**7 дней**	**Чт 16.02.23**	**Пт 24.02.23**	**5**	**Сніжко А. Я.;Чибіряк Я. І.**
Постановка мети та задач проекту	2 дней	Чт 16.02.23	Пт 17.02.23		Касьяненко Д. Р.;Чибіряк Я. І.
Розробка технічного завдання	2 дней	Пн 20.02.23	Вт 21.02.23	7	Касьяненко Д. Р.
Моделювання додатку в нотації IDEF0	1 день	Ср 22.02.23	Ср 22.02.23	8	Касьяненко Д. Р.
Моделювання діаграми використання	1 день	Чт 23.02.23	Чт 23.02.23	9	Касьяненко Д. Р.
Проектування інформаційної системи	1 день	Пт 24.02.23	Пт 24.02.23	10	Касьяненко Д. Р.
Розробка	**51 дней**	**Пн 27.02.23**	**Пн 08.05.23**	**11**	**Касьяненко Д. Р.**
Створення структури проекту	6,6 дней	Пн 27.02.23	Вт 07.03.23		Касьяненко Д. Р.
Створення сервера	2 дней	Ср 08.03.23	Чт 09.03.23	13	Касьяненко Д. Р.
Створення обробника повідомлень	7 дней	Пт 10.03.23	Пн 20.03.23	14	Касьяненко Д. Р.
Створення чатів	5 дней	Вт 21.03.23	Пн 27.03.23	15	Касьяненко Д. Р.
Тестування серверу	2 дней	Вт 28.03.23	Ср 29.03.23	16	Касьяненко Д. Р.
Створення аутентифікації	4 дней	Чт 30.03.23	Вт 04.04.23	17	Касьяненко Д. Р.
Підключення до БД	1 день	Ср 05.04.23	Ср 05.04.23	18	Касьяненко Д. Р.
Створення	6 дней	Чт 06.04.23	Чт 13.04.23	19	Касьяненко Д. Р.

Рисунок Б.3 – Перелік виконаних робіт

Название задачи	Длительн	Начало	Окончани	Предшественн	Названия ресурсов
Підключення до БД	1 день	Ср 05.04.23	Ср 05.04.23	18	Касьяненко Д. Р.
Створення інтерфейса додатку	6 дней	Чт 06.04.23	Чт 13.04.23	19	Касьяненко Д. Р.
Beta-тестування	8 дней	Пт 14.04.23	Вт 25.04.23		Касьяненко Д. Р.
Alpha-тестування	5 дней	Ср 26.04.23	Вт 02.05.23	21	Касьяненко Д. Р.
Перевірка працездатності	4 дней	Ср 03.05.23	Пн 08.05.23	22	Касьяненко Д. Р.
Завершення	**9 дней**	**Вт 09.05.23**	**Пт 19.05.23**	**23**	**Касьяненко Д. Р.;Чибіряк Я. І.**
Створення документації	8 дней	Вт 09.05.23	Чт 18.05.23		Касьяненко Д. Р.;Чибіряк Я. І.
Презентація проек	1 день	Пт 19.05.23	Пт 19.05.23	25	Касьяненко Д. Р.

Рисунок Б.4 – Продовження переліку виконаних робіт



Название задачи	Длительнс	Начало	Окончани
Розробка мобільного додатку "Connect People"	**97 дней**	**Ср 01.02.23**	**Чт 15.06.23**
Ініціалізація	**11 дней**	**Ср 01.02.23**	**Ср 15.02.23**
Аналіз предметної області	4 дней	Ср 01.02.23	Пн 06.02.23
Аналіз програмних продуктів аналогів	5 дней	Вт 07.02.23	Пн 13.02.23
Вибір середовища розробки	2 дней	Вт 14.02.23	Ср 15.02.23
Планування	**7 дней**	**Чт 16.02.23**	**Пт 24.02.23**
Постановка мети та задач проекту	2 дней	Чт 16.02.23	Пт 17.02.23
Розробка технічного завдання	2 дней	Пн 20.02.23	Вт 21.02.23
Моделювання додатку в нотації IDEF0	1 день	Ср 22.02.23	Ср 22.02.23
Моделювання діаграми використання	1 день	Чт 23.02.23	Чт 23.02.23
Проектування інформаційної системи	1 день	Пт 24.02.23	Пт 24.02.23
Розробка	**51 дней**	**Пн 27.02.23**	**Пн 08.05.23**
Створення структури проекту	6,6 дней	Пн 27.02.23	Вт 07.03.23
Створення сервера	2 дней	Ср 08.03.23	Чт 09.03.23
Створення обробника повідомлень	7 дней	Пт 10.03.23	Пн 20.03.23
Створення чатів	5 дней	Вт 21.03.23	Пн 27.03.23
Тестування серверу	2 дней	Вт 28.03.23	Ср 29.03.23
Створення аутентифікації	4 дней	Чт 30.03.23	Вт 04.04.23
Підключення до БД	1 день	Ср 05.04.23	Ср 05.04.23
Створення	6 дней	Чт 06.04.23	Чт 13.04.23

Рисунок Б.5 – Діаграма Ганта

Рисунок Б.6 – Продовження діаграми Ганта

Аналіз ризиків. Виконано якісну і кількісну оцінку ризиків роботи. При якісній оцінці визначено проблеми, що потребують швидкого реагування. Така оцінка визначить ступінь важливості і дозволить вибрати спосіб реагування. Кількісна оцінка буде виконана для більш повної ідентифікації ризиків та ступеня їхнього впливу на виконання проекту. Кількісна і якісна оцінка можуть використовуватися окремо або разом, залежно від наявного часу і бюджету, необхідності в кількісній або якісній оцінці. У табл. Б.6 знаходиться класифікація ризиків за показниками ймовірності виникнення ризику та величині втрат.

Далі виконано планування реагування на ризики — це розробка методів і технологій зниження негативного впливу ризиків на проект. Визначено ефективність розробки реагування на проект, визначено чи будуть наслідки впливу ризику на проект позитивними або негативним. Оцінено ризики за показниками, що знаходяться в табл. Б.3. На основі оцінки побудовано матрицю ймовірності виникнення ризиків та впливу ризику, що знаходиться в табл. Б.4.

Таблиця Б.3 – Шкала оцінювання ймовірності виникнення та впливу ризику на виконання проекту

Оцінка	Ймовірність виникнення	Вплив ризику
1	Низька	Низький
2	Середня	Середній
3	Висока	Високий

Таблиця Б.4 - Матриця ймовірності виникнення ризиків та впливу ризику

Вплив ризику **Ймовірність виконання**	Низький	Середній	Високий
Висока	-	RS_2	-
Середня	RS_7	RS_3, RS_6	-
Низька	-	RS_1, RS_5	RS_4

- зелений колір – прийнятні ризики;
- жовтий колір – виправданні ризики;
- червоний колір – недопустимі ризики.

На підставі отриманого значення індексу ризики класифікують: за рівнем ризику, що знаходиться в табл. Б.4.

Таблиця Б.5 – Шкала оцінювання за рівнем ризику

№	Назва	Межі	Ризики, які входять (номера)
1	Прийнятні	$0{,}005 \leq R \leq 0{,}05$	5, 7
2	Виправдані	$0{,}05 < R \leq 0{,}16$	1, 3, 4, 6
3	Недопустимі	$0{,}16 < R \leq 0{,}72$	2

Таблиця Б.6 – Оцінка ймовірності виникнення, величини витрат та індексу ризику

ID	Статус ризику	Опис ризику	Ймовірність виникнення	Вплив	Ранг ризику	План А	Тип стратегії реагування	План Б
RS_1	Закритий	Непорозуміння між розробниками та замовником	Низька	Середній	0,08	Налагодження відносин. Дотримання ділового етикету спілкування.	Попередження	Аналіз та обговорення проблем
RS_2	Відкритий	Продукти-конкуренти	Висока	Середній	0,24	Створення більших переваг у кінцевого продукта	Попередження	Обговорення з замовниками щодо популяризації продукту
RS_3	Відкритий	Недоліки завдання на розробку	Середня	Середній	0,15	Обговорення всіх видів вимог з замовником. Контроль замовником етапів розробки	Попередження	Окреслення що було зроблено невірно та виправлення помилок
RS_4	Закритий	Низька кваліфікація розробників	Низька	Великий	0,16	Переконуватися у кваліфікації виконавців	Попередження	Найняти інших людей
RS_5	Відкритий	Відпустки/лікарняні працівників	Низька	Середній	0,04	Внесення коректив у графік	Прийняття	
RS_6	Відкритий	Часте внесення змін в ТЗ	Середня	Середній	0,12	Обговорення всіх майбутніх змін та правок.	Попередження	Поступове розроблення всіх правок
RS_7	Закритий	Неоптимальний розподіл часу	Середня	Низький	0,03	Додання робочих годин розробникам	Попередження	Обговорення з замовником щодо додання деякого часу на розробку

ДОДАТОК В.

Програмний код

index.js:

```
import cors from "cors"
import express from "express"
import { createServer } from "http"
import { Server } from "socket.io"
import { ALLOWED_ORIGIN } from "./config.js"
import onConnection from "./src/socket.io/onConection.js"
import onError from "./src/utils/onError.js"

const app = express()

app.use(
  cors({
    origin: ALLOWED_ORIGIN,
  })
)

app.use(onError)

const server = createServer(app)

const io = new Server(server, {
  // ❗ ПРЕДУПРЕЖДЕНИЕ О БЕЗОПАСНОСТИ. Настройка источника *открывает возможность для фишинговых сайтов имитировать внешний вид вашего сайта
  cors: {
    origin: "*",
  },
})

// Событие коннекта сокета к серверу
io.on("connection", (socket) => {
```

```
  onConnection(io, socket)
})

// Событие не удачного коннекта сокета к серверу
io.on("connect_failed", function () {
  console.log("Connection Failed")
})

const PORT = process.env.PORT || 4000
server.listen(PORT, () => {
  console.log(`🚀 Server started on port ${PORT}`)
})
```

onError.js:

```
export default function onError(err, req, res, next) {
  console.log(err)

  // если имеется объект ответа
  if (res) {
    // статус ошибки
    const status = err.status || err.statusCode || 500
    // сообщение об ошибке
    const message = err.message || "Something went wrong. Try again later"
    res.status(status).json({ message })
  }
}
```

onConection.js:

```
import userHandlers from "./handlers/user.handler.js"
import usersHandlers from "./handlers/users.handler.js"
import messageHandlers from "./handlers/message.handlers.js"
import roomsHandlers from "./handlers/rooms.handlers.js"

export default function onConnection(io, socket) {
  console.log("a user connected")

  // регистрируем обработчики для пользователя
```

```
  userHandlers(io, socket)

  // регистрируем обработчики для пользователей
  usersHandlers(io, socket)

  // регистрируем обработчики для сообщений
  messageHandlers(io, socket)

  // регистрируем обработчики для комнат
  roomsHandlers(io, socket)

  // Прослушивание событий отключения сокета от сервера
  socket.on("disconnect", () => {
    // выводим сообщение
    console.log("User disconnected")
  })
}
```

message.handler.js:

```
// Регистрируем обработку соответствующих событий:
// message:get — получение сообщений
// message:add — добавление сообщения
// message:remove — удаление сообщения

// _id (string) — индентификатор сообщения
// user (object) — пользователь
// text (string) — текст сообщения
// createdAt (date) — дата создания

import { LowSync, JSONFileSync } from "lowdb"
// БД хранится в директории "db" под названием "messages.json"
const adapter = new JSONFileSync("messages.json")
const db = new LowSync(adapter)

// Чтение данных из файла JSON, это установит содержимое db.data
await db.read()
```

```
export default function messageHandlers(io, socket) {
  // обрабатываем запрос на получение сообщений
  const getMessages = async ({ roomId }) => {
    // Чтение данных из файла JSON, это установит содержимое db.data
    await db.read()

    // получаем сообщения из БД
    const messages = db.data.rooms.find((item) => item._id === roomId).messages

    // передаем сообщения пользователям, находящимся в комнате
    // синонимы - распространение, вещание, публикация
    io.in(roomId).emit("messages", messages)
  }

  // обрабатываем добавление сообщения
  // функция принимает объект сообщения и идентификатор комнаты
  const addMessage = async ({ _id, createdAt, text, user, roomId }) => {
    db.data.rooms
      // ищем нашу конмату
      .find((item) => item._id === roomId)
      .messages.push({
        _id,
        createdAt,
        text,
        user,
      })

    await db.write()
    // выполняем запрос на получение сообщений
    getMessages({ roomId })
  }

  // обрабатываем удаление сообщение
  // функция принимает id сообщения
  const removeMessage = (_id) => {
    db.data.message.remove({ _id })
```

```
    db.write()

    // выполняем запрос на получение сообщений
    getMessages()
  }

  // регистрируем обработчики
  socket.on("message:get", getMessages)
  socket.on("message:add", addMessage)
  socket.on("message:remove", removeMessage)
}
```

rooms.handler.js:

```
// Регистрируем обработку соответствующих событий:
// rooms:get — получение комнат
// rooms:add — добавление комнаты
// rooms:remove — удаление комнаты

// _id (string) — идентификатор комнаты
// roomName (object) — название комнаты
// messages (string) — массив сообщеный

import { LowSync, JSONFileSync } from "lowdb"
// БД хранится в директории "db" под названием "messages.json"
const adapter = new JSONFileSync("messages.json")
const db = new LowSync(adapter)

// Чтение данных из файла JSON, это установит содержимое db.data
await db.read()
// Установить данные по умолчанию
if (!db.data) {
  db.data = { rooms: [] }
  await db.write()
}

export default function roomsHandlers(io, socket) {
  // обрабатываем запрос на получение комнат
```

```
const getRooms = async () => {
  // Чтение данных из файла JSON, это установит содержимое db.data
  await db.read()
  // получаем комнаты из БД
  const rooms = db.data

  // передаем комнаты пользователю
  io.emit("rooms", rooms)
}

// обрабатываем добавление комнаты
// функция принимает идентификатор комнаты, название комнаты
const addRoom = async ({ roomId, roomName }) => {
  db.data.rooms.push({
    _id: roomId,
    roomName,
    messages: [],
  })

  await db.write()
  // выполняем запрос на получение комнат
  getRooms()
}

// обрабатываем удаление комнаты
// функция принимает id комнаты
const removeRoom = (_id) => {
  db.data.rooms.remove({ _id })
  db.write()

  // выполняем запрос на получение комнат
  getRooms()
}

// регистрируем обработчики
socket.on("rooms:get", getRooms)
```

```
  socket.on("room:add", addRoom)
  socket.on("room:remove", removeRoom)
}
```

users.handler.js:

```
// Регистрируем обработку соответствующих событий:
// user:get — получение пользователей
// user:add — добавление пользователя
// user:leave — удаление пользователя
// user:typing — пользователь печатает сообщения
// user:stopTyping — пользователь перестал печатать сообщения

// userId — индентификатор пользователя
// online — индикатор нахождения пользователя в сети
// userName — имя пользователя
// avatar — аватар пользователя

import { LowSync, JSONFileSync } from "lowdb"
// БД хранится в директории "db" под названием "users.json"
const adapter = new JSONFileSync("users.json")
const db = new LowSync(adapter)

// Чтение данных из файла JSON, это установит содержимое db.data
await db.read()
// Установить данные по умолчанию
if (!db.data) {
  db.data = { users: [] }
  await db.write()
}

export default function userHandlers(io, socket) {
  // обрабатываем запрос на получение пользователей
  const getUsers = async () => {
    // Чтение данных из файла JSON, это установит содержимое db.data
    await db.read()

    // получаем сообщения из БД
```

```
  const users = db.data

  // передаем пользователей
  io.emit("users", users)
}

// обрабатываем добавление пользователя
// функция принимает объект с именем пользователя и его id и avatar
const addUser = async ({ userName, userId, avatar }) => {
  // проверяем, имеется ли пользователь в БД
  if (
    db.data &&
    db.data.users.find((el) => {
      return el.userId === userId
    })
  ) {
    // если имеется, меняем его статус на онлайн
    db.data.users.find((el) => {
      return el.userId === userId
    }).online = true
  } else {
    // если не имеется, добавляем его в БД
    db.data.users.push({ userId, online: true, userName, avatar })
  }
  await db.write()
  getUsers()
}

// обрабатываем удаление пользователя
const removeUser = async (userId) => {
  // меняем его статус на офлайн
  if (db.data.users) {
    db.data.users.find((el) => el.userId === userId).online = false
    await db.write()
  }
  getUsers()
```

```
  }

  // обрабатываем событие когда пользователь печатает
  const UserTyping = async ({ userId, userName, roomId }) => {
    // отправляем событие только в ту комнату где находится пользователь
    socket.to(roomId).emit("usersTyping", { userId, userName })
  }

  // обрабатываем событие когда пользователь перестал печатает
  const UserStopTyping = async ({ userId, userName, roomId }) => {
    socket.to(roomId).emit("usersStopTyping", { userId, userName })
  }

  // регистрируем обработчики
  socket.on("user:get", getUsers)
  socket.on("user:add", addUser)
  socket.on("user:leave", removeUser)
  socket.on("user:typing", UserTyping)
  socket.on("user:stopTyping", UserStopTyping)
}
```

App.tsx:

```
import { StatusBar } from "expo-status-bar"
import { Text, View } from "react-native"
import { registerRootComponent } from "expo"

function App() {
  return (
    <View style={{ flex: 1 }}>
      <Text>Open up App.tsx to start working on your app</Text>
      <StatusBar style="auto" />
    </View>
  )
}

export default registerRootComponent(App)
```

babel.config.js:

```
module.exports = function (api) {
  api.cache(true)
  return {
    presets: ["babel-preset-expo"],
    plugins: [
      [
        "module:react-native-dotenv",
        {
          envName: "APP_ENV",
          moduleName: "@env",
          path: ".env",
        },
      ],
    ],
  }
}
```

firebase-config.ts:

```
import { initializeApp } from 'firebase/app';
import { getAuth } from 'firebase/auth';
import { getFirestore } from 'firebase/firestore';

// получаем наши значения конфига
import {
  REACT_APP_FIREBASE_API_KEY,
  REACT_APP_FIREBASE_AUTH_DOMAIN,
  REACT_APP_FIREBASE_PROJECT_ID,
  REACT_APP_FIREBASE_STORAGE_BUCKET,
  REACT_APP_FIREBASE_MESSAGING_SENDER_ID,
  REACT_APP_FIREBASE_APP_ID,
  REACT_APP_FIREBASE_MEASUREMENT_ID,
} from '@env'



// Конфиг для подключения firebase
const firebaseConfig = {
    apiKey: REACT_APP_FIREBASE_API_KEY ,
```

```
    authDomain: REACT_APP_FIREBASE_AUTH_DOMAIN ,
    projectId: REACT_APP_FIREBASE_PROJECT_ID ,
    storageBucket: REACT_APP_FIREBASE_STORAGE_BUCKET,
    messagingSenderId: REACT_APP_FIREBASE_MESSAGING_SENDER_ID,
    appId: REACT_APP_FIREBASE_APP_ID,
    measurementId: REACT_APP_FIREBASE_MEASUREMENT_ID
  };


  const app = initializeApp(firebaseConfig)


 export const db = getFirestore() // Для работы с Firebase Firestore


 export const auth = getAuth(app)
```

auth-context.tsx:

```
import React from "react"
import {
  createUserWithEmailAndPassword,
  signInWithEmailAndPassword,
  updatePassword,
  UserCredential,
  signInWithCredential,
  sendPasswordResetEmail,
  GoogleAuthProvider,
  updateProfile,
  User as FirebaseUser,
  User,
  OAuthCredential,
} from "firebase/auth"
import { auth as Auth } from "../../Firebase/firebase-config"
import { navigationRef } from "../../utils/rootNavigation"
import { ROUTES } from "../../constants/constants"
import { AsyncStore } from "../../services/async-store"
import * as Google from "expo-auth-session/providers/google"
import * as WebBrowser from "expo-web-browser"
```

```
import { AuthRequestPromptOptions, AuthSessionResult } from "expo-auth-session"

// чтобы закрыть всплывающее окно. Если вы забудете добавить это, всплывающее окно не закроется.
WebBrowser.maybeCompleteAuthSession()

export const AuthProvider = ({ children }: Props) => {
  // наш пользователь
  const [currentUser, setCurrentUser] = React.useState<FirebaseUser | null>(
    null
  )
  // флаг для крутилки
  const [loadingUser, setLoadingUser] = React.useState<boolean>(false)
  // флаг для крутилки нашего приложения
  const [loadingApp, setLoadingApp] = React.useState<boolean>(false)
  // сообщения об ошибки
  const [errorUserMessage, setErrorUserMessage] = React.useState<string | null>(
    null
  )
  // сообщения об успешности
  const [successUserMessage, setSuccessUserMessage] = React.useState<
    string | null
  >(null)

  // идентификаторы
  const [requst, response, promtAsunc] = Google.useAuthRequest({
    expoClientId:
      "Your_expoClientId",
  })

  React.useEffect(() => {
    ;(async () => {
      // если ответ успешный
      if (response?.type === "success") {
        // получаем полномочия на вход пользователя в наш firebase
        const credential = GoogleAuthProvider.credential(
          response.authentication?.idToken,
```

```
        response.authentication?.accessToken
      )
      try {
        // записуем в наше локальное хранилище данные пользователя
        await AsyncStore.setValue(
          "google",
          JSON.stringify({
            idToken: response.authentication?.idToken,
            accessToken: response.authentication?.accessToken,
          })
        )
      } catch (error) {}
      handleSignInWithCredential(credential)
    }
  })()
}, [response])

const [isAuth, setIsAuth] = React.useState<boolean>(false)

// функция сброса пароля
const resetPassword = (password: string) => {
  setLoadingUser(true)
  return updatePassword(Auth.currentUser as User, password)
    .then(() => {
      setErrorUserMessage("")
      setSuccessUserMessage("Password reset successful")
      navigationRef.current.navigate(ROUTES.profile)
    })
    .catch(() => {
      setErrorUserMessage("Failed to reset password")
    })
    .finally(() => {
      setLoadingUser(false)
      setTimeout(() => {
        setSuccessUserMessage(null)
      }, 5000)
```

```
    })
}

// функция обновления профиля пользователя
const handleUpdateProfile = (displayName: string) => {
  setLoadingUser(true)
  console.log("displayName", displayName)
  return updateProfile(Auth.currentUser as User, {
    displayName,
  })
    .then(() => {
      console.log("YES")
      setErrorUserMessage("")
      setSuccessUserMessage("Profile update successful")
      navigationRef.current.navigate(ROUTES.profile)
    })
    .catch(() => {
      setErrorUserMessage("Failed to profile update")
    })
    .finally(() => {
      setLoadingUser(false)
      setTimeout(() => {
        setSuccessUserMessage(null)
      }, 5000)
    })
}

// функция сброса почты
const resetPasswordOnEmail = (email: string) => {
  setLoadingUser(true)
  return sendPasswordResetEmail(Auth, email)
    .then(() => {
      setErrorUserMessage(null)
      setSuccessUserMessage("Check your email address")
      navigationRef.current.navigate(ROUTES.login)
    })
```

```
    .catch((error) => {
      setErrorUserMessage("Failed to reset password on email")
      console.log("error", error)
    })
    .finally(() => {
      setLoadingUser(false)
      setTimeout(() => {
        setSuccessUserMessage(null)
      }, 5000)
    })
}

// функция выхода
const logout = async () => {
  setLoadingUser(true)
  try {
    // удаляем состояния нашего локального хранилищя
    await AsyncStore.deleteValue("user")
    await AsyncStore.deleteValue("google")
  } catch (error) {
    setErrorUserMessage("AsyncStore error")
  }
  return Auth.signOut()
    .then(() => {
      setErrorUserMessage(null)
      setIsAuth(false)
    })
    .catch(() => {
      setErrorUserMessage("Failed to log out")
    })
    .finally(() => {
      setLoadingUser(false)
    })
}

// функция входа в firebase через google
```

```
const handleSignInWithCredential = (credential: OAuthCredential) => {
  signInWithCredential(Auth, credential)
    .then(() => {
      Auth?.onAuthStateChanged((user) => {
        setCurrentUser(user)
      })
      setIsAuth(true)
      setErrorUserMessage(null)
      setSuccessUserMessage("Success")
    })
    .catch(() => {
      setErrorUserMessage("Failed to log in with Google")
    })
    .finally(() => {
      setTimeout(() => {
        setSuccessUserMessage(null)
      }, 5000)
    })
}

// функция входа в firebase через email и password
const login = async (email: string, password: string) => {
  setLoadingUser(true)
  try {
    // записуем в наше локальное хранилище данные пользователя
    await AsyncStore.setValue(
      "user",
      JSON.stringify({
        email,
        password,
      })
    )
  } catch (error) {
    setErrorUserMessage("AsyncStore error")
  }
  // делаем вход в наш firebase
```

```
  return signInWithEmailAndPassword(Auth, email, password)
    .then(() => {
      setErrorUserMessage(null)
      setIsAuth(true)
    })
    .catch((error) => {
      setErrorUserMessage("Incorrect login or password")
      console.log("error", error)
    })
    .finally(() => {
      setLoadingUser(false)
    })
}

// функция регистраци польвателя firebase
const singUp = (email: string, password: string) => {
  setLoadingUser(true)
  return createUserWithEmailAndPassword(Auth, email, password)
    .catch(() => {
      setErrorUserMessage("Failed to sing up")
    })
    .then(() => {
      setErrorUserMessage("")
      setIsAuth(true)
    })
    .finally(() => {
      setLoadingUser(false)
    })
}

React.useEffect(() => {
  // когда меняется состояние AuthState записуем нашего пользователя
  const unSubscribe = Auth?.onAuthStateChanged((user) => {
    setCurrentUser(user)
  })
  return unSubscribe
```

```
}, [])

// при входе в приложения проверяем есть ли пользователь в локальном хранилище
React.useEffect(() => {
  ;(async () => {
    setLoadingApp(true)
    try {
      let user: { email: string; password: string } | null | string =
        await AsyncStore.getValue("user")
      user = user != null ? JSON.parse(user) : null
      if (user) {
        await login(user.email, user.password)
      } else {
        let google: { idToken: string; accessToken: string } | null | string =
          await AsyncStore.getValue("google")
        google = google != null ? JSON.parse(google) : null
        if (google) {
          const credential = GoogleAuthProvider.credential(
            google.idToken,
            google.accessToken
          )
          handleSignInWithCredential(credential)
        }
      }
    } catch (error) {
      setErrorUserMessage("AsyncStore error")
    } finally {
      setLoadingApp(false)
    }
  })()
}, [])

return (
  <AuthContext.Provider
    value={{
      currentUser,
```

```
        singUp,
        login,
        logout,
        resetPassword,
        resetPasswordOnEmail,
        handleUpdateProfile,
        loadingUser,
        loadingApp,
        errorUserMessage,
        successUserMessage,
        isAuth,
        promtAsunc,
      }}
    >
      {children}
    </AuthContext.Provider>
  )
}

type Props = {
  children: React.ReactNode
}

// прописуем типы для наших value
type ContextProps = {
  currentUser: FirebaseUser | null
  singUp: (email: string, password: string) => Promise<void> | void
  login: (
    email: string,
    password: string
  ) => Promise<UserCredential | void> | void
  logout: () => Promise<UserCredential | void> | void
  resetPassword: (password: string) => Promise<void> | void
  resetPasswordOnEmail: (email: string) => Promise<void> | void
  handleUpdateProfile: (displayName: string) => Promise<void> | void
  promtAsunc: (
```

```
    options?: AuthRequestPromptOptions | undefined
  ) => Promise<AuthSessionResult> | void
  loadingUser: boolean
  loadingApp: boolean
  errorUserMessage: string | null
  successUserMessage: string | null
  isAuth: boolean
}

export const AuthContext = React.createContext<ContextProps>({
  currentUser: null,
  promtAsunc: () => {},
  singUp: () => {},
  login: () => {},
  logout: () => {},
  resetPassword: () => {},
  resetPasswordOnEmail: () => {},
  handleUpdateProfile: () => {},
  loadingUser: false,
  loadingApp: false,
  errorUserMessage: null,
  successUserMessage: null,
  isAuth: false,
})

export const useAuthContext = () => {
  const authContext = React.useContext(AuthContext)
  if (!authContext) {
    throw new Error("useAuthContext must be used within a AuthProvider")
  }
  return authContext
}
```

WebSocketContext.tsx:

```
import React from 'react';
import { io, Socket } from 'socket.io-client';
import {
```

```
  ClientToServerEvents,
  ServerToClientEvents,
} from '../../constants/types';

type SocketType = Socket<ServerToClientEvents, ClientToServerEvents> | null;

export const WebSocketProvider = ({ children }: Props) => {
  const socketRef = React.useRef<SocketType>(null);

  // создаем экземпляр сокета, передаем ему адрес сервера
  // и записываем объект с названием комнаты в строку запроса "рукопожатия"
  // socket.handshake.query.roomId
  const connectToServer = () => {
    socketRef.current = io('http://your_ip:4000');

  const joinInRoom = ({ roomId }: { roomId: string }) => {
    socketRef?.current?.emit('user:connectToRoom', {
      roomId,
    });
  };

  const leaveInRoom = ({ roomId }: { roomId: string }) => {
    socketRef?.current?.emit('user:leaveToRoom', {
      roomId,
    });
  };

  // при размонтировании компонента выполняем отключение сокета
  const disconnectToServer = () => {
    socketRef.current?.disconnect();
  };

  return (
    <WebSocketContext.Provider
      value={{
        socketRef,
```

```
        joinInRoom,
        leaveInRoom,
        connectToServer,
        disconnectToServer,
      }}
    >
      {children}
    </WebSocketContext.Provider>
  );
};

type Props = {
  children: React.ReactNode;
};

type ContextProps = {
  socketRef: React.MutableRefObject<SocketType> | null;
  joinInRoom: ({ roomId }: { roomId: string }) => void;
  leaveInRoom: ({ roomId }: { roomId: string }) => void;
  connectToServer: () => void;
  disconnectToServer: () => void;
};

export const WebSocketContext = React.createContext<ContextProps>({
  socketRef: null,
  connectToServer: () => {},
  joinInRoom: () => {},
  leaveInRoom: () => {},
  disconnectToServer: () => {},
});

export const useWebSocketContext = () => {
  const webSocketContext = React.useContext(WebSocketContext);
  if (!webSocketContext) {
    throw new Error(
      'useWebSocketContext must be used within a WebSocketProvider',
```

```
    );
  }
  return webSocketContext;
};
```

useAppActive.ts:

```
import React from "react";
import { AppState, AppStateStatus } from "react-native";

export const useAppActive = () => {
  // получаем Ref на состояние нашего приложения
  const appState = React.useRef(AppState.currentState);
  const [appStateVisible, setAppStateVisible] = React.useState(appState.current);


  const listener = (nextAppState : AppStateStatus) => {
       if (
        appState.current.match(/inactive|background/) &&
        nextAppState === "active"
      ) {
        console.log("App has come to the foreground!");
      }

      appState.current = nextAppState;
      setAppStateVisible(appState.current);
      console.log("AppState", appState.current);
  }

  React.useEffect(() => {

    // добавляем слушатель
    AppState.addEventListener("change", listener)

    // отписуемся от него при демонтировании компоненты
    return () => {
      AppState.removeEventListener('change', listener);
    };
```

```
  }, []);

  // возвращаем наше состояние
  return { appStateVisible }

};
```

useChat.ts:

```
 import React from 'react';
import { IMessage } from 'react-native-gifted-chat';
import { messagesType, MessageUser, usersType, usersOBJ, usersTypingType } from '../constants/types'
import { useAuth, useWebSocket } from '../hooks/index';

// проверяем на параметры наш хук
type Props = {
    roomId : string,
    userId :string,
    userName : string,
}

export const useChat = ({ roomId , userId, userName } : Props)  => {
  // локальное состояние для пользователей
  const [users, setUsers] = React.useState<usersType | []>([])
   // локальное состояние для пользователей
  const [usersTyping, setUsersTyping] = React.useState<usersTypingType[] | []>([])
  // локальное состояние для сообщений
  const [messages, setMessages] = React.useState<IMessage[] | []>([])

  // получаем наш сокет и функции
  const { socketRef, leaveInRoom, joinInRoom } = useWebSocket()

  // получаем нашего пользователя
  const {currentUser } = useAuth()

  // создаем timeout для идентификатора печатания пользователя
```

```
const timeout = React.useRef<ReturnType<typeof setTimeout> | null>(null);
// создаем Ref состаяния печатает ли пользователь
const isTypingRef = React.useRef<boolean>(false);



React.useEffect(() => {

  // подключаемся к комнате
  joinInRoom({ roomId });

  return () => {
      // при демонтировании компонента выполняем отключение от комнаты
      leaveInRoom({roomId})
  }

}, [roomId, userId, userName])

React.useEffect(() => {

  // отправляем запрос на получение сообщений
  socketRef?.current?.emit('message:get', { roomId })

  // обрабатываем получение сообщений
  socketRef?.current?.on("messages" , (messages : messagesType) => {

    // обновляем массив сообщений
    setMessages(messages.reverse() as IMessage[] | [])
  })

  return () => {
    // удаляет указанный прослушиватель из массива прослушивателей для события с именем messages
.
    socketRef?.current?.off("messages")
  }

}, [roomId, userId, userName])
```

```
  React.useEffect(() => {

    // отправляем событие добавления пользователя,
    socketRef?.current?.emit('user:add', { userName, userId, avatar: currentUser?.photoURL ||
'https://i.pravatar.cc/300' })

      // обрабатываем получених пользователей
    socketRef?.current?.on("users", (users : usersOBJ) => {

      // обновляем массив пользователей
      setUsers(users.users)
    })

    return () => {
        // удаляет указанный прослушиватель из массива прослушивателей для события с именем users
.
        socketRef?.current?.off("users")
    }

  }, [roomId , userId, userName])

  React.useEffect(() => {

    // получаем пользотеля который начал печатать
    socketRef?.current?.on("usersTyping", (userTyping : usersTypingType) => {
        // обновляем массив пользотелей которые печатают
        setUsersTyping((prev) => {
          const temp = prev.slice()
          // если пользователь уже есть не добавляем эго
          !temp.some(item => item.userId === userTyping.userId) && temp.push(userTyping)
          // if (temp.some(item => item.userId === userTyping.userId)) {

          // } else {
          //   temp.push(userTyping)
          // }
          return temp
        })
```

```
    })


    return () => {
        // удаляет указанный прослушиватель из массива прослушивателей для события с именем usersTyping .
        socketRef?.current?.off("usersTyping")
    }


  }, [roomId , userId, userName])


  React.useEffect(() => {


     // получаем пользотеля который перестал печатать
    socketRef?.current?.on("usersStopTyping", (userStopTyping : usersTypingType) => {
        // обновляем массив пользотелей которые печатают
        setUsersTyping((prev) => {
          const temp = prev.slice()
          // содаем index пользователя которого будем удалять
          let indexDelete = 0
          if (temp.some((item, index) => {
            if (item.userId === userStopTyping.userId) {
              indexDelete = index
              return true
            }
            return false
           })) {
            // удаляем пользователя
            temp.splice(indexDelete, 1)
          }
          return temp
        })
    })


    return () => {
        // удаляет указанный прослушиватель из массива прослушивателей для события с именем usersStopTyping .
        socketRef?.current?.off("usersStopTyping")
```

```
    }

  }, [roomId , userId, userName])


  // функция отправки сообщения
  // принимает объект с текстом сообщения и именем отправителя
  const sendMessage = ({ _id, createdAt, text, user, roomId } : { _id : string, createdAt : string,
text : string , user  : MessageUser, roomId : string}) => {
    // отправляем событие что пользователя перестал печатать
    socketRef?.current?.emit('user:stopTyping', { userName, userId, roomId })
    isTypingRef.current = false

    // отправляем событие на добавления сообщения
    socketRef?.current?.emit("message:add", {
        _id,
        createdAt,
        text,
        user,
        roomId,
    })
  }

    // функция удаления сообщения по id
    const removeMessage = (_id : string) => {
       // отправляем событие на удаления сообщения
        socketRef?.current?.emit('message:remove', _id)
    }

    // функция обработки когда пользователь пичатает
    const handleInputTextChanged = (text : string) => {
      if (text) {
        if (!isTypingRef.current) {
          socketRef?.current?.emit('user:typing', { userName, userId, roomId })
          isTypingRef.current = true
        }
        clearTimeout(timeout.current);
```

```
        timeout.current = setTimeout(() => {
          // отправляем событие что пользователя начал печатать
          socketRef?.current?.emit('user:stopTyping', { userName, userId, roomId })
          isTypingRef.current = false
        }, 1500);
      }
    }


    // возвращаем наши значения и функции
    return { users, messages, sendMessage, removeMessage, handleInputTextChanged, usersTyping }

}
```

useChats.ts:

```
import React from 'react'
import { roomsType, roomsOBJ } from '../constants/types'
import { useAppActive } from './useAppActive';
import { useAuth, useWebSocket } from '../hooks/index';

// проверяем на параметры наш хук
type Props = {
    userId :string,
    userName : string,
}

export const useChats = ({ userId, userName } : Props)  => {

  // локальное состояние для комнат
  const [rooms, setRooms] = React.useState<roomsType | []>([])

  // получаем функии connectToServer, disconnectToServer для нашего socket
  // и сам socket
  const { connectToServer, disconnectToServer, socketRef } = useWebSocket()
  const { currentUser } = useAuth()

  // получаем состояния нашего приложения
```

```
  const { appStateVisible } = useAppActive();


  React.useEffect(() => {
    // если приложения свернуто
    if (appStateVisible === 'background') {
      // вызываем функцию что пользователь вышел
      leaveUser(userId)
    }
    // если приложения стало активно
    if (appStateVisible === "active") {
      // вызываем функцию что пользователь зашел
      userConnect({userName,userId, avatar: currentUser?.photoURL || 'https://i.pravatar.cc/300'
})
    }
  }, [appStateVisible]);



  React.useEffect(() => {
    // коннектимся к нашему серверу
    connectToServer()

    return () => {
        leaveUser(userId)
        // отключаемся от нашего серверу
        disconnectToServer()
        console.log("disconnect")
    }

  }, [userId, userName])

  React.useEffect(() => {

    // отправляем запрос на получение комнат
    socketRef?.current?.emit('rooms:get')

    // обрабатываем получение комнаты
    socketRef?.current?.on("rooms" , (rooms : roomsOBJ) => {
```

```
        setRooms(rooms.rooms)
    })

    return () => {
        // удаляет указанный прослушиватель из массива прослушивателей для события с именем rooms
.
        socketRef?.current?.off("rooms")
    }

  }, [userId, userName])



  const createRoom = ({ roomId, roomName } : { roomId : string, roomName : string }) => {
    // отправляем запрос на добавления комнаты
    socketRef?.current?.emit("room:add" ,  {
        roomId,
        roomName,
    })
  }

  const leaveUser = (_id : string) => {
    // отправляем событие на отключение пользователя
    socketRef?.current?.emit('user:leave', _id)
  }

  const userConnect = ({ userId, userName, avatar } : {userId : string, userName : string, avatar
: string}) => {
    // отправляем событие добавления пользователя
    socketRef?.current?.emit('user:add', { userName, userId, avatar })
  }



    // хук возвращает комнаты, и функцию создания комнаты
    return { rooms, createRoom }

}
```

index.ts:

```
export { useAuthContext as useAuth } from '../hooks/context/auth-context';

export { useWebSocketContext as useWebSocket } from '../hooks/context/WebSocketContext';

export * from './useChat';

export * from './useChats';

export * from './useAppActive';
```

types.ts:

```
export type DataMessages =  {

  _id :  string,

  roomName : string,

  messages : messagesType

}


export type User = {

    Name : string

    Age : number

    id : string

}


export type usersTypingType = {

  userName : string

  userId : string

}


export type MessageUser = {

  _id : string,

  name : string,

  avatar? : string

}


export type usersOBJ = {users : Array<{userName : string, online : boolean}>  }

export type usersType = Array<{userName : string, online : boolean, avatar? : string }>

export type roomsOBJ = {rooms : Array<DataMessages> }

export type roomsType = Array<DataMessages>

export type messagesOBJ = {messages :  Array<{_id : string, user : {

    _id  : string,

    name : string,
```

```
    avatar? : string
} , senderName : string, text : string, createdAt :string, currentUser? : boolean}> }
export type messagesType = Array<{_id : string, user : {
    _id  : string,
    name : string,
    avatar? : string
} , senderName : string, text : string, createdAt :string, currentUser? : boolean}>

// Тип событий, которые мы отправляем на сервер
export interface ClientToServerEvents {
    noArg: () => void;
    basicEmit: (a: number, b: string, c: number[]) => void;
    'user:add' : ({ userName , userId } : { userName : string, userId : string, avatar? : string
| undefined | null }) =>  void
    'message:get' : ({ roomId } : {roomId : string}) => void
    "message:add" : ({ _id , createdAt, text , user, roomId} : {_id :string, createdAt : string,
text : string , user : MessageUser, roomId : string} )  => void
    "message:remove" : ( _id : string ) => void
    "user:leave" : ( _id : string ) => void
    "user:connectToRoom" : ( { roomId } : { roomId : string } ) => void
    "user:leaveToRoom" : ( { roomId } : { roomId : string } ) => void
    "user:typing" : ({ userName , userId, roomId } : { userName : string, userId : string, roomId
: string }) => void
    "user:stopTyping" : ({ userName , userId, roomId } : { userName : string, userId : string,
roomId : string }) => void
    "rooms:get" : () => void
    "room:add" : ( { roomId, roomName } : { roomId : string, roomName : string } ) => void
  }

// Тип событий, которые мы получаем из сервер
export interface ServerToClientEvents {
    withAck: (d: string, cb: (e: number) => void) => void;
    users : (users : usersOBJ) => void
    messages : (messages : messagesType) => void
    rooms : (rooms : roomsOBJ) => void
    usersTyping : (userTyping : usersTypingType) => void
    usersStopTyping : (userStopTyping : usersTypingType) => void
  }
```

Chat.tsx:

```
import {
  NativeStackNavigationProp,
  NativeStackScreenProps,
} from "@react-navigation/native-stack"
import { Box, Center, HStack, Text } from "native-base"
import React from "react"
import { UsersList } from "../components"
import { DEVICE_WIDTH, ROUTES } from "../constants/constants"
import { MainStackParamList } from "../navigation/MainNavigator"
import { AntDesign } from "@expo/vector-icons"
import { useNavigation } from "@react-navigation/native"
import { TouchableOpacity } from "react-native"
import { GiftedChat, IMessage } from "react-native-gifted-chat"
import { useAuth, useChat } from "../hooks"
import { SafeAreaView } from "react-native-safe-area-context"
import { StatusBar } from "expo-status-bar"
// для инимации typing
import { TypingAnimation } from "react-native-typing-animation"

// Для проверки route реквизитов
type RouteProps = NativeStackScreenProps<MainStackParamList, ROUTES.chat>

// создаем тип для проверки маршрутов и параметров
type NavigationProps = NativeStackNavigationProp<MainStackParamList>

export const Chat = ({ route }: RouteProps) => {
  // получаем id комнаты и название комнаты
  const roomId = route?.params?.roomId
  const roomName = route?.params?.roomName

  const [isOpen, setIsOpen] = React.useState(false)

  // получаем текущего пользователя
```

```
const { currentUser } = useAuth()

// получаем наши сообщения и список пользователей
// так же получаем callback для отправки сообщения и callback когда пользователь печатает
const { messages, sendMessage, users, handleInputTextChanged, usersTyping } =
  useChat({
    roomId,
    userId: currentUser?.email as string,
    userName: currentUser?.displayName as string | "user",
  })

// создаем массив для наших сообщений
const [messagesTemp, setMessagesTemp] = React.useState<IMessage[] | []>(
  messages
)

React.useEffect(() => {
  setMessagesTemp(messages)
}, [messages.length])

const navigation = useNavigation<NavigationProps>()

const onClose = () => {
  setIsOpen(false)
}

// функия которая отрабатывает при отправки сообщения
const onSend = React.useCallback((messages = []) => {
  setMessagesTemp((previousMessages: IMessage[]) =>
    GiftedChat.append(previousMessages, messages)
  )
  const { _id, createdAt, text, user } = messages[0]
  sendMessage({
    _id,
    createdAt,
    text,
```

```
    user,
    roomId,
  })
}, [])

// компонент которые будет отображатся когда пользователь будет печатать
const renderFooter = () => {
  if (usersTyping?.length) {
    return (
      <HStack padding="2" maxWidth="200">
        <TypingAnimation
          style={{
            marginRight: 32,
          }}
        />
        {usersTyping?.map((item, index) => {
          return (
            <Text mr="1">
              {item.userName}
              {index + 1 === usersTyping.length ? "" : ","}
            </Text>
          )
        })}
        <Text>is typing</Text>
      </HStack>
    )
  }
  return null
}

return (
  <SafeAreaView
    edges={["right", "top", "left"]}
    style={{
      flex: 1,
      backgroundColor: "#fff",
```

```
  }}
>
  <StatusBar />
  <Box
    width={DEVICE_WIDTH}
    justifyContent="center"
    flexDirection="row"
    position="relative"
  >
    <Center
      position="absolute"
      left="0"
      top="0"
      zIndex="3"
      width="16"
      height="12"
    >
      <TouchableOpacity onPress={() => navigation.goBack()}>
        <AntDesign name="arrowleft" size={24} color="black" />
      </TouchableOpacity>
    </Center>
    <Box position="relative" width="100%">
      <TouchableOpacity activeOpacity={0.5} onPress={() => setIsOpen(true)}>
        <Box alignItems="center">
          <Text fontSize="18" maxWidth="180" noOfLines={1}>
            {roomName}
          </Text>
          <Text>
            users {users?.length},{" "}
            {
              users?.filter((user) => {
                return user.online === true
              }).length
            }{" "}
            online
          </Text>
```

```
      </Box>

      <Box position="absolute" right="0" top="3">
        <Box
          background="#fff"
          height="40px"
          borderRadius="25"
          paddingX="6"
        >
          <Text fontSize="16px" mr="3" color="#000">
            Users
          </Text>
        </Box>
      </Box>
    </TouchableOpacity>
  </Box>
</Box>
<GiftedChat
  messages={messagesTemp}
  onSend={(messages) => onSend(messages)}
  user={{
    _id: currentUser?.email as string | number,
    avatar: currentUser?.photoURL || "https://i.pravatar.cc/300",
    name: currentUser?.displayName || "user",
  }}
  messagesContainerStyle={{
    backgroundColor: "#fff",
  }}
  // если true появляется иконка прокрутки чата к концу
  scrollToBottom={true}
  renderUsernameOnMessage={true}
  onInputTextChanged={handleInputTextChanged}
  renderFooter={renderFooter}
/>
<UsersList
  // модальное окно со списком пользователей
```

```
        isOpen={isOpen}
        onClose={onClose}
        users={users}
      />
    </SafeAreaView>
  )
}
```