Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“____” _____ 20__ р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за першим (бакалаврським) рівнем вищої освіти
на тему
“Програмне забезпечення мобільного додатку для обміну миттєвими повідомленнями”

Виконав здобувач вищої освіти
IV курсу, групи КІ-19
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп’ютерна інженерія»
_____ Шевчук В. О.
«____» _____ 20 р.

Керівник проекту
доктор технічних наук, професор
_____ Мелешко Є. В.
«____» _____ 20 р.
Рецензент _____
_____

м. Кропивницький

Міністерство освіти і науки України
Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*
Кафедра *Кібербезпеки та програмного забезпечення*
Освітній ступінь *бакалавр*
Спеціальність *123 Комп'ютерна інженерія*

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____О.А. Смірнов
« 21 » грудня 2023 року

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

Шевчуку Владиславу Олександровичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Програмне забезпечення мобільного додатку для обміну миттєвими повідомленнями*

керівник роботи *Мелешко Єлизавета Владиславівна, д-р техн. наук, професор*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу №7-02 від 05.01.2023 року

2. Строк подання студентом роботи до захисту *22.05.2023 р.*

3. Мета та завдання кваліфікаційної бакалаврської роботи*: Метою розробки є програмне забезпечення для мобільного додатку для обімну миттєвими повідомленнями*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Призначення та область використання.
2.Перегляд аналогічних існуючих систем.
3. Опис і обґрунтування проектних рішень.
4. Етапи програмування системи.
5. Впровадження системи в промислову експлуатацію.
6. Висновки

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

Міжсторінкова структурна схема додатку	*1 аркуш*
Структурна схема колекції користувачів у базі даних Firebase Firestore	*1 аркуш*
Структурна схема колекції чат-кімнат у базі даних Firebase Firestore	*1 аркуш*
Функціональна схема архітектури представлення BLoC роботи системи	*1 аркуш*
Діаграма процесів додатку для обміну миттєвими повідомленнями	*1 аркуш*
Блок-схема основної програми	*1 аркуш*
Блок-схема Push-повідомлень	*1 аркуш*

6. Дата видачі завдання « 21 » грудня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної бакалаврської роботи	Строк виконання етапів кваліфікаційної бакалаврської роботи	Примітка
1.	Аналіз існуючих систем	10.03.2023 р.	
2.	Постановка задачі, оформлення ТЗ	15.03.2023 р.	
3.	Розробка моделі компонента	20.03.2023 р.	
4.	Розробка структур даних	25.03.2023 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.03.2023 р.	
6.	Програмування алгоритмів	10.04.2023 р.	
7.	Оформлення ПЗ	17.04.2023 р.	
8.	Попередній захист роботи	22.05.2023 р.	

Студент _____ _____
(підпис) (прізвище та ініціали)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Шевчук В.О. Програмне забезпечення мобільного додатку для обміну миттєвими повідомленнями. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2023.

У даній кваліфікаційній бакалаврській роботі розроблено програмне забезпечення мобільного додатку, яке призначено для обміну миттєвими повідомленнями через мережу Інтернет.

Метою роботи є створення програмного забезпечення мобільного додатку, яке призначено для обміну миттєвими повідомленнями через мережу Інтернет.

Результат роботи – програмна реалізація мобільного додатку, яке призначено для обміну миттєвими повідомленнями через мережу Інтернет.

В процесі роботи над реалізацією системи виконано дослідження існуючих методів, алгоритмів та програмних засобів. Розроблено та реалізовано власне програмне забезпечення, здійснено опис всіх його компонентів.

Розроблено зручний інтерфейс користувача. Наведені інструкції по роботі з програмними засобами.

Програма може використовуватися на мобільних операційних системах IOS та Android.

Програму розроблено на мові програмування Dart та Фреймворку Flutter.

Ключові слова: комп'ютерна інженерія, мобільна розробка, обмін миттєвими повідомленнями

ABSTRACT

Shevchuk V.O. Mobile instant messaging application software. 123 Computer Engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2023

In this qualifying bachelor's thesis, the software of a mobile application designed for the exchange of instant messages over the Internet was developed.

The purpose of the work is to create a mobile application software, which is intended for the exchange of instant messages over the Internet.

The result of the work is the software implementation of a mobile application designed for instant messaging over the Internet.

In the process of working on the implementation of the system, a study of existing methods, algorithms and software tools was carried out. Our own software was developed and implemented, and all its components were described.

A convenient user interface has been developed. Instructions for working with software tools are provided.

The program can be used on mobile operating systems IOS and Android.

The program was developed using the Dart programming language and the Flutter framework.

Keywords: computer engineering, mobile development, instant messaging

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

Flutter	–	Фреймворк для розробки мобільних додатків.
API	–	(application programming interface) інтерфейс програмування застосунків, набір чітко визначених методів для взаємодії різних компонентів.
Dart	–	Однопоточна мова програмування. Яку розробляє компанія Google, позиціонуючи як мову структурованого програмування
Firebase	–	Це платформи розробки мобільних та веб за стосунків.
ПЗ	–	Програмне забезпечення.

ВСТУП

Актуальність теми. Тема розробки мобільного застосунку для миттєвої комунікації є дуже актуальною у наш час, оскільки зростає попит на швидкий та надійний обмін інформацією серед користувачів мобільних пристроїв. Такий застосунок може бути корисним для різних категорій користувачів, таких як студенти, бізнесмени та інші люди, які потребують швидкої комунікації зі своїми колегами та близькими.

Крім того, розробка мобільного застосунку для передачі миттєвих повідомлень може знайти застосування в надзвичайних ситуаціях та сприяти покращенню ефективності комунікації в компаніях. Все це робить тему дуже перспективною і важливою для дослідження та розвитку.

Однією з головних переваг мобільного застосунку для миттєвої комунікації є його можливість забезпечувати швидку та безпечну передачу повідомлень в режимі реального часу. Це особливо важливо у сферах, де від часу залежить результат – таких як фінанси, медицина, транспорт тощо. Крім того, застосунок може бути інтегрований з іншими додатками та сервісами, що дозволяє забезпечувати ще більші можливості для користувачів.

Мобільний застосунок може забезпечувати користувачам необмежений доступ до важливої інформації з будь-якого місця та в будь-який час, що може значно підвищити продуктивність та допомогти швидко реагувати на поточні завдання.

Мета й завдання дослідження. Метою роботи є розробка програмного забезпечення мобільного додатку для обміну миттєвими повідомленнями.

Для досягнення поставленої мети визначена програма дослідження, що складається з наступних завдань:

1. Огляд наявних мобільних додатків для обміну миттєвими повідомленнями та аналіз їх функціональності, інтерфейсу та ефективності.

2. Визначення основних вимог до мобільного додатку для обміну миттєвими повідомленнями, таких як швидкість доставки повідомлень, безпека, тощо.

3. Розробка концепції та дизайну мобільного додатку, включаючи інтерфейс користувача та функціональність.

4. Розробка технічної архітектури мобільного додатку, включаючи вибір технологій та платформи.

Практична цінність отриманих результатів полягає в тому, Що цей додаток може бути використаний для забезпечення більш швидкого та ефективного спілкування між користувачами. Крім того, такий додаток може підвищити рівень безпеки та конфіденційності при передачі повідомлень.

Практична цінність отриманих результатів моєї дипломної роботи може також полягати в тому, що результати дослідження можуть бути використані для подальшого розвитку та удосконалення додатків, які дозволяють передавати миттєві повідомлення.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення мобільного додатку

Мобільний додаток для обміну миттєвими повідомленнями – це програмне забезпечення, що дозволяє користувачам обмінюватися текстовими повідомленнями в режимі реального часу на мобільних пристроях. Основною метою такого додатку є забезпечення швидкого та зручного обміну повідомленнями між користувачами, що дозволяє їм спілкуватися в режимі реального часу. Такий додаток може бути корисним для комунікації між друзями, колегами, родичами та іншими людьми.

Функціональні вимоги для такого додатку будуть включати реєстрацію та авторизацію користувачів за електронною поштою, надсилання та отримання повідомлень в режимі реального часу, а також систему передачі повідомлень через Інтернет та зберігання повідомлень в середовищі Firestore.

Про надходження нового повідомлення користувачів буде сповіщено повідомленням на їхній мобільний пристрій.

Крім того, в мобільному додатку для обміну миттєвими повідомленнями є функція локалізації, що дозволяє користувачам вибирати мову інтерфейсу. Додаток підтримує англійську та українську мови, що робить його більш зручним та доступним для користувачів з різних країн та регіонів. При виборі мови інтерфейсу користувачі зможуть зрозуміти всі функції та можливості додатку, що покращить їхній досвід користування.

Крім того, в разі надходження нового повідомлення користувачі будуть сповіщені повідомленням на свій мобільний пристрій, що дозволить їм оперативно реагувати на повідомлення та спілкуватися з іншими користувачами в режимі реального часу.

1.2 Область застосування

Мобільні додатки для обміну миттєвими повідомленнями широко використовуються в сучасному світі, як основний інструмент для комунікації між людьми. Ці додатки є важливим елементом в сфері комунікації, бізнесу та розваг.

У сфері бізнесу, мобільні додатки для обміну миттєвими повідомленнями дозволяють користувачам зберігати зв'язок з колегами та партнерами, спілкуватися з клієнтами та вести бізнес-переписку. Вони також допомагають вирішувати проблеми та розглядати питання в режимі реального часу, що є важливою складовою ефективної роботи.

У сфері розваг, мобільні додатки для обміну миттєвими повідомленнями дозволяють користувачам спілкуватися з друзями, родичами та знайомими з будь-якого місця та в будь-який час. Вони також можуть включати функції голосового та відеозв'язку, які дозволяють користувачам бути ближче один до одного, навіть якщо вони знаходяться на різних кінцях світу.

Крім того, мобільні додатки для обміну миттєвими повідомленнями можуть бути використані для особистих цілей, наприклад, для спілкування з рідними та близькими, або для знайомства з новими людьми через мережу.

Мобільні додатки для обміну миттєвими повідомленнями стали необхідним інструментом для багатьох компаній, організацій та індивідуальних користувачів. Вони можуть бути використані для організації віддаленої комунікації між працівниками, проведення відеоконференцій та обміну важливими документами. Крім того, мобільні додатки для обміну миттєвими повідомленнями можуть покращити взаємодію зі споживачами та збільшити задоволеність клієнтів шляхом швидкої та ефективної відповіді на запити та запитання

Усі ці функції дозволяють мобільним додаткам для обміну миттєвими повідомленнями бути важливим інструментом комунікації в сучасному світі.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ МОБІЛЬНИХ ДОДАТКІВ

2.1 Огляд існуючих мобільних додатків, технологій, архітектур, програмних рішень за профілем теми кваліфікаційної бакалаврської роботи

На Фреймворку Flutter були написані наступні додатки для миттєвої передачі повідомлень.

Google Allo – це додаток від Google, який використовує машинне навчання для пропозицій відповідей на повідомлення. Allo був запущений у 2016 році, але був закритий в 2019 році.

Reflectly – це додаток для щоденного журналювання, який використовується для записування щоденних подій та емоцій. Reflectly був написаний на Flutter та має понад 10 мільйонів завантажень з Google Play та App Store.

Hamilton – це додаток для обміну миттєвими повідомленнями, який був розроблений за допомогою Flutter. Цей додаток був випущений у 2019 році та був заснований на концепції "довіри" – користувачі можуть обмінюватися повідомленнями з іншими користувачами, яким вони довіряють.

Hookle – це соціальна мережа, яка дозволяє користувачам об'єднуватися в групи та ділитися цікавими статтями та повідомленнями. Hookle був розроблений на Flutter та був випущений у 2019 році.

Оскільки додаток був написаний на обєктно-орієнотованії мові програмування є сенс розглянути абревіатуру з принципів ООП – SOLID.

Загальною метою всіх принципів SOLID є поліпшення масштабованості, підтримки та розуміння коду. Ось детальніше про кожен з них:

1. Принцип єдиного обов'язку (SRP) – цей принцип вимагає, щоб кожен клас мав лише один функціональний обов'язок. Це означає, що кожен клас має відповідати лише за одну частину функціональності програми, і ця частина

повинна бути закріплена відповідною областю відповідальності. Такий підхід дозволяє підтримувати код більш зрозумілим, зменшити залежності між класами, зручніше розробляти і тестувати код.

2. Принцип відкритості/закритості (OCP) – цей принцип вимагає, щоб класи були відкриті для розширення, але закриті для змін. Це означає, що при додаванні нових функціональних можливостей до програми необхідно розширювати, а не змінювати вже існуючий код. Такий підхід дозволяє зберегти старий функціонал без змін, зменшити кількість помилок та зробити код більш зручним для підтримки.

3. Принцип підстановки Лісков (LSP) – цей принцип вимагає, щоб підкласи могли бути використані замість своїх батьківських класів, не змінюючи коректність роботи програми. Це означає, що класи, що наслідуються від інших класів, не повинні змінювати їх поведінку. Це дозволяє збільшити перевикористання коду, розширювати програму без зміни основного коду та зменшувати залежності між класами.

4. Принцип розділення інтерфейсу та реалізації (ISP) – цей принцип вимагає, щоб клієнти залежали від інтерфейсів, а не від конкретних реалізацій. Це означає, що класи мають реалізовувати тільки ті методи, які є необхідними для їхньої роботи, і не повинні мати зайвих методів. Це дозволяє зберегти код більш зрозумілим, зменшити залежності між класами, спростити тестування та зробити код більш гнучким.

5. Принцип інверсії залежностей (DIP) – цей принцип вимагає, щоб високорівневі модулі не залежали від низькорівневих модулів, а обидві групи модулів залежали від абстракцій. Це означає, що класи не повинні залежати від конкретних реалізацій інших класів, а лише від їх інтерфейсів. Це дозволяє зберегти код більш зручним для перевикористання та тестування, зменшити залежності між класами та зробити код більш гнучким.

Тож, застосування принципів SOLID дозволяє зменшити залежності між класами, зробити код більш зрозумілим та гнучким, забезпечити

перевикористання коду та полегшити тестування. Ці принципи є ключовими для розробки великих проектів та для забезпечення стабільності та підтримки коду в подальшому.

Також є сенс розглянути KISS (Keep It Simple, Stupid) – це принцип проектування та програмування, який рекомендує зберігати код якомога простішим та зрозумілішим. Простота коду повинна бути головним критерієм прийняття рішень щодо архітектури, дизайну та імплементації.

Основна ідея KISS полягає в тому, що прості рішення зазвичай є кращими за складні. Більш складний код може мати більше можливостей та функцій, але це також означає більше можливостей для помилок та складнішу розробку та підтримку.

Застосування принципу KISS означає, що код повинен бути зрозумілим, простим та легким для читання та зрозуміння. Це допомагає збільшити продуктивність розробника, зменшити кількість помилок та покращити читабельність та підтримку коду в майбутньому.

Отже, KISS є важливим принципом проектування та програмування, який допомагає зберегти код простим та зрозумілим.

DRY (Don't Repeat Yourself) – це принцип програмування, який підкреслює важливість того, щоб кожен елемент в програмному коді був представлений тільки в одному місці.

Основна ідея DRY полягає в тому, щоб уникати дублювання коду та зберігати кожен елемент програмного коду в одному місці. Це допомагає зменшити кількість помилок, полегшити підтримку коду, забезпечити більшу перевикористовуваність та спростити розробку нових функцій.

DRY є важливим принципом проектування та програмування, який сприяє збереженню чистоти та ефективності коду, а також полегшує його підтримку та розробку нових функцій.

YAGNI – це скорочення від англійської фрази "You Ain't Gonna Need It". Це принцип, який використовується в програмуванні та означає, що не слід

писати код або функціонал, який не потрібен в даний момент або не буде потрібний у майбутньому.

Цей принцип покликаний зменшити кількість зайвого коду в проекті та допомогти зосередитися на необхідних функціях. Зайвий код може призвести до складнощів у підтримці, ускладнити розуміння проекту та знизити продуктивність розробки.

YAGNI є однією з основних принципів методології програмування "екстремальне програмування" (Extreme Programming, XP). Ця методологія акцентує увагу на зменшенні складнощів та підвищенні продуктивності розробки шляхом спрощення процесу розробки та використання лише необхідних функцій.

BDUF (Big Design Up Front) – це підхід до розробки програмного забезпечення, де багато часу витрачається на детальне проектування системи та розробку документації перед початком реалізації.

BDUF базується на ідеї того, що під час планування проекту можна передбачити всі можливі вимоги та проблеми, що виникнуть під час розробки, Тому у BDUF надається велика увага у проектуванні та документуванні, з метою забезпечити ретельну підготовку перед фактичною реалізацією проекту.

Проте, такий підхід має декілька недоліків. По-перше, часто вимоги до системи можуть змінюватися під час розробки, тому всі деталі планування можуть стати неактуальними. По-друге, BDUF забирає багато часу на проектування та документацію, що може затримати час початку реалізації та сповільнити процес розробки.

У зв'язку з цим, деякі команди розробників використовують Agile-методології, де акцент робиться на постійному вдосконаленні та ітераційному процесі розробки, замість докладного планування перед початком роботи.

APO (Agile Product Ownership) – це підхід до управління розробкою програмного продукту, що базується на Agile методології та акцентує увагу на розробці та збереженні цінності продукту.

Основні принципи APO що можна застосувати у даному проекті:

1. Робота над короткими циклами: продукт розробляється у коротких ітераціях (Sprints), що дозволяє швидко відгукатися на зміни та швидко вносити необхідні зміни до продукту.

2. Постійна зміна та адаптація: розробка продукту повинна бути гнучкою та дозволяти змінювати його відповідно до потреб користувачів.

2.2 Обґрунтування вибору засобів для побудови мобільного додатку та мови програмування

Фреймворк Flutter та мова Dart були обранні для написання мобільного додатку. Ось кілька причин, чому Flutter і Dart це гарне рішення для написання додатків:

1. Швидкість та продуктивність: Flutter використовує власну графічну бібліотеку та компілює код напряму в нативний код мобільних платформ. Це дозволяє досягати високої продуктивності та швидкості в роботі додатків.

2. Гнучкість: Flutter дозволяє розробникам створювати віджети, які можуть бути використані на різних платформах. Це забезпечує гнучкість та зручність в розробці кросплатформних додатків.

3. Зручність у розробці: Dart – це мова програмування з високим рівнем абстракції, що дозволяє розробникам легко писати код з гарною структурою та швидкістю розробки.

4. Гарний інтерфейс: Flutter має вбудовану колекцію матеріалів та купу віджетів для створення гарного та зручного інтерфейсу додатків.

5. Велика спільнота розробників: Flutter та Dart мають велику спільноту розробників, яка постійно працює над розширенням функціональності та вдосконаленням технологій. Це забезпечує підтримку та зручність в розробці додатків.

6. Відкритий код: Flutter та Dart є відкритими проектами з відкритим кодом, що дозволяє розробникам доповнювати функціональність, вирішувати проблеми та долучатися до спільноти розробників.

Як базу даних для зберігання користувачів та діалогів було використано Firestore, на це є ряд причин:

1. Швидкість та масштабованість: Firestore забезпечує швидкий доступ до даних та легку масштабованість. Це дозволяє зберігати та отримувати повідомлення в режимі реального часу, що дуже важливо для месенджерів.

2. Легкість використання: Firestore має дуже простий та легкий інтерфейс, що дозволяє розробникам легко працювати з базою даних та забезпечує швидкий розвиток додатків.

3. Реальний час: Firestore підтримує режим реального часу, що дозволяє надсилати та отримувати повідомлення без затримок та оновлювати стан додатку в режимі реального часу.

4. Автономна робота: Firestore підтримує автономну роботу додатка, що дозволяє користувачам отримувати та надсилати повідомлення, навіть якщо вони не мають з'єднання з Інтернетом.

5. Захист даних: Firestore забезпечує захист даних та пропонує різні рівні доступу до даних, що дозволяє забезпечити конфіденційність даних користувачів та уникнути витоку даних.

Firestore також має можливість роботи з даними в офлайн режимі, що дозволяє зберігати та синхронізувати дані, коли користувач не має з'єднання з Інтернетом. Крім того, Firestore має підтримку для багатьох платформ, таких як Android, iOS, веб-браузери та сервери, що дозволяє розробникам створювати додатки для різних платформ.

Firestore також пропонує можливість реалізації транзакцій та оптимізації запитів до бази даних, що дозволяє збільшити продуктивність та ефективність роботи додатку. Крім того, Firestore є частиною екосистеми Firebase, яка пропонує широкий набір інструментів для розробки додатків, таких як аутентифікація користувачів, зберігання файлів, аналітика та багато іншого.

Всі ці можливості дозволяють розробникам швидко створювати потужні та надійні додатки для різних платформ з максимальним рівнем безпеки та

ефективності.

Узагалі, Firestore є дуже потужним інструментом для розробки месенджерів та інших додатків, які вимагають швидкого та надійного доступу до даних в режимі реального часу.

Як архітектуру представлення було вирішено обрати Bloc.

Flutter Bloc – це бібліотека для керування станом додатку в Flutter, яка забезпечує чітке розділення бізнес-логіки та представлення в інтерфейсі. Використання Flutter Bloc дозволяє досить просто та ефективно управляти станом додатку та виконувати асинхронні операції.

Flutter Bloc рекомендується для написання додатків з великою кількістю станів, де присутній багатокроковий процес асинхронних операцій, таких як отримання даних з бази даних чи мережі. Бібліотека дозволяє створювати потужні та масштабовані додатки, які можуть легко розвиватись в майбутньому.

За допомогою Flutter Bloc можна ефективно організувати код, зменшити залежності між класами та модулями, спростити тестування, а також забезпечити високу швидкість та продуктивність додатку.

Отже, Flutter Bloc – це чудовий вибір для написання додатку месенджера, де важливо правильно керувати станом додатку та виконувати багато асинхронних операцій.

Редактором вихідного коду був обраний VsCode.

VsCode є дуже популярним редактором коду в середовищі розробки програмного забезпечення. Ось кілька причин, чому VsCode можна вважати гарним редактором коду:

1. Він є безкоштовним та відкритим: VsCode можна завантажити та використовувати безкоштовно, і він розповсюджується під ліцензією MIT.

2. Має вбудовану підтримку багатьох мов програмування: VsCode має вбудовану підтримку більшості популярних мов програмування, що дозволяє працювати з різними мовами в одному і тому ж редакторі.

3. Має велику кількість розширень: VsCode надає безліч розширень, які дозволяють розширювати можливості редактора, надавати додаткові функції та підтримку для конкретних технологій.

4. Має інтегровану систему керування версіями: VsCode має вбудовану систему керування версіями Git, яка дозволяє зберігати та керувати історією змін коду.

5. Має можливості для роботи в команді: VsCode дозволяє розширювати можливості для роботи в команді, надаючи можливість спільно редагувати код, обговорювати зміни та використовувати інші інструменти для покращення роботи команди.

6. Має високу продуктивність: VsCode працює дуже швидко та ефективно, що дозволяє зосередитися на розробці програмного забезпечення, а не на очікуванні завантаження та обробки файлів.

Ці переваги дозволяють збільшити продуктивність та зручність розробки програмного забезпечення з використанням VsCode.

Для реєстрації користувачів було використано Firebase Auth. Firebase Auth є сервісом аутентифікації, який надається Firebase, що дозволяє розробникам легко і швидко додавати функцію аутентифікації в свої додатки. Ось деякі причини, чому Firebase Auth є гарним рішенням для аутентифікації користувачів:

1. Простота використання: Firebase Auth пропонує простий API для інтеграції з будь-якими мобільними або веб-додатками. Це дозволяє розробникам легко додавати аутентифікацію користувача до своїх додатків з мінімальними зусиллями.

2. Різноманітність провайдерів: Firebase Auth підтримує різні провайдери аутентифікації, такі як Google, Facebook, Twitter, GitHub, Email та Password, телефонний номер і т.д. Розробники можуть додавати один або кілька провайдерів в залежності від своїх потреб.

3. Безпека: Firebase Auth пропонує вбудовану безпеку, таку як захист від перехоплення мережі, захист від атак брут-форсу та захист від вразливостей у

вхідних даних. Крім того, Firebase Auth надає можливість налаштування двофакторної аутентифікації для підвищення безпеки.

4. Розширені функції: Firebase Auth дозволяє налаштовувати різноманітні функції, такі як керування сесіями користувачів, відновлення пароля та налаштування вхідних і вихідних рівнів. Це дозволяє розробникам забезпечувати більш гнучкий та зручний досвід користувача.

Для локалізації додатку було використано пакет easy_localization. Пакет Flutter easy_localization є зручним і простим рішенням для локалізації мобільних додатків на Flutter. Ось деякі причини, чому easy_localization є гарним рішенням для розробки мобільних додатків:

1. Простота використання: easy_localization пропонує простий та інтуїтивний API для додавання локалізації до мобільних додатків. Розробники можуть легко налаштувати всі необхідні параметри та почати використовувати його в своїх проектах.

2. Підтримка багатьох мов: easy_localization підтримує багатомовність та дозволяє легко додавати нові мови до додатків. Розробники можуть створювати переклади для своїх додатків у різних мовах та легко переключатися між ними.

3. Гнучкість: easy_localization дозволяє розробникам легко змінювати мови в реальному часі без перезавантаження додатку. Крім того, він дозволяє розробникам легко перевіряти та відлагоджувати свої переклади.

4. Підтримка різних форматів: easy_localization підтримує різні формати перекладу, такі як ARB, CSV, JSON та YML. Розробники можуть вибрати формат, який їм найбільше підходить.

5. Автоматична генерація коду: easy_localization може автоматично генерувати код для перекладів на основі файлів перекладу, що значно спрощує процес локалізації та збільшує продуктивність розробки.

6. Проект повинен бути написаний з дотриманням загальноприйнятої стилістики коду що включає в себе: створення змінних та методів виключно з маленької літери подальші слова з великою за необхідністю позначать

приватність методів та змінних через символ «_», класи повинні починатись з великої літери усі подальші слова що містить клас також з великої, над кожним класом повинен бути короткий опис його роботи для кращого орієнтування у проекті, назви директорій та файлів повинні описувати їх вміст, файли та директорії що містять більше одного слова розділяються символом «_», усі шляхи навігації починаються із символу «/». Програма бере початок у файлі main що містить метод main.

Для перетворення об'єктів Dart у JSON-строку та навпаки було використано JsonSerialization.

JsonSerialization – це процес перетворення об'єктів Dart у JSON-строку та навпаки. У Flutter JsonSerialization є корисним інструментом для збереження та передачі даних в мобільних додатках.

JSON (JavaScript Object Notation) є форматом обміну даними, який можна легко читати та записувати. У багатьох випадках, додатки можуть потребувати обміну даними між клієнтом та сервером у форматі JSON.

Також JsonSerialization дозволяє зберігати дані на мобільному пристрої в форматі JSON, що може бути корисно для зберігання налаштувань додатків, кешування даних та іншого.

Для створення сповіщень що надходять на телефон було використано Firebase Cloud Messaging (FCM) – це рішення для повідомлень, що дозволяє розробникам легко надсилати повідомлення до своїх користувачів на платформах Android, iOS та веб-сайтах.

Ось декілька причин, чому Firebase Cloud Messaging може бути гарним рішенням для додатку:

1. Швидкість та надійність: FCM дозволяє надсилати повідомлення в режимі реального часу і забезпечує швидку доставку до мільйонів користувачів. Також FCM пропонує надійну доставку повідомлень, що дозволяє бути впевненими, що повідомлення досягнуть призначення.

2. Можливості налаштування: FCM дозволяє налаштовувати повідомлення, щоб вони відповідали потребам користувачів. Можна налаштувати вигляд повідомлення, змінити звук, додати відображення повідомлень у системній лозі, та багато іншого.

3. Підтримка мультиплатформеності: FCM підтримує надсилання повідомлень на платформи Android, iOS та веб-сайти. Це дозволяє забезпечити користувачів повідомленнями на будь-якій платформі, що вони використовують.

4. Інтеграція з Firebase: FCM інтегровано з Firebase, що дозволяє використовувати його разом з іншими сервісами Firebase, такими як Firebase Analytics, Firebase Crashlytics, Firebase Authentication та іншими.

Узагалі, Firebase Cloud Messaging є потужним інструментом для надсилання повідомлень на мобільні платформи. Він дозволяє ефективно спілкуватися з іншими користувачами та забезпечувати надійну та швидку доставку повідомлень.

Для роботи з асинхронністю було використано Stream та Future. У мові програмування Dart, яка використовується в Flutter, Future та Stream – це два різні механізми асинхронної обробки даних.

Future – це об'єкт, який представляє обчислювальну операцію, що ще не завершилась. Можна запустити операцію та отримати обіцянку результату у вигляді Future. Коли операція завершиться, результат буде встановлено в Future, та ви зможете отримати результат, використовуючи різні методи, такі як then(), catchError(), whenComplete(). Використання Future дозволяє не блокувати потік виконання під час виконання довгих операцій, а замість цього зарезервувати ресурси, та продовжувати роботу з програмою.

Stream – це послідовність даних, які можуть бути випущені в будь-який момент часу. Можна отримувати дані з Stream за допомогою підписки на нього. Кожен раз, коли з'являється нове значення, він буде викликати підписників. Stream дозволяє реагувати на змінюючіся дані та асинхронно їх обробляти.

Отже, Future та Stream дозволяють працювати з асинхронними

операціями. Future дозволяє отримати результат після завершення операції, тоді як Stream дозволяє працювати з даними, які можуть змінюватись з часом. Використання Future та Stream дозволяє зробити програму більш ефективною, тому що можна продовжувати роботу з програмою під час виконання довгих асинхронних операцій.

Для керування та зберігання версій проекту було використано Github – це платформа для розміщення та спільної роботи над проектами з використанням системи контролю версій Git. Це гарне рішення з кількох причин:

1. Безкоштовна реєстрація та зберігання коду: Github надає безкоштовні репозиторії для зберігання коду, що дозволяє легко зберігати та оновлювати свій проект на централізованому сервері.

2. Спільна робота над проектами: Github дозволяє багатокористувацьку роботу над проектами, що дозволяє команді розробників легко співпрацювати та ділитись своїми досягненнями.

3. Контроль версій: Github дозволяє використовувати систему контролю версій Git, що дозволяє зберігати різні версії свого проекту та легко відстежувати зміни, які вносилися до проекту в різні моменти часу.

4. Наявність інструментів для спільної роботи: Github надає різні інструменти для спільної роботи над проектами, такі як pull requests, issues, code reviews, що дозволяє команді розробників легко взаємодіяти між собою та вирішувати проблеми в процесі розробки.

5. Широкі можливості для відкритого програмного забезпечення: Github підтримує відкрите програмне забезпечення, що дозволяє ділитись своїм кодом з іншими користувачами та залучати їх до спільної роботи над проектом.

Узагалі, Github є потужним та дуже корисним інструментом для розробників, що дозволяє зберігати та спільно працювати над проектами, використовуючи систему контролю версій Git.

Крім того, Github є важливим інструментом для спільноти відкритого програмного забезпечення, оскільки він дозволяє розробникам легко

співпрацювати та вносити свій внесок до проектів, що є відкритими для участі. Користувачі можуть використовувати Github для пошуку відкритих проектів, які відповідають їх інтересам та долучитись до них, надаючи свій внесок та допомагаючи розвивати ці проекти.

У якості маршрутизації між сторінками була використана іменна маршрутизація в Flutter – це підхід до організації маршрутизації, де кожен екран має свій унікальний ідентифікатор (назву) і звернення до екранів відбувається за допомогою цієї назви.

Це гарне рішення з кількох причин:

1. Простота: Іменна маршрутизація дозволяє легко і швидко змінювати маршрути без необхідності вручну оновлювати посилання на ці маршрути в усьому коді. Крім того, вона дозволяє з легкістю додавати нові екрани до додатку без необхідності змінювати весь код маршрутизації.

2. Флексібільність – іменна маршрутизація дозволяє передавати параметри з одного екрану на інший, що дозволяє додатку бути більш гнучким і пристосовуватися до потреб користувачів.

3. Зручність – іменна маршрутизація дозволяє розробникам швидко знаходити та відлагоджувати проблеми з маршрутизацією, а також зробити його більш зрозумілим і легким для розуміння.

4. Розділення відповідальності – іменна маршрутизація дозволяє розділити відповідальність за маршрутизацію між різними модулями та класами, що сприяє кращій організації коду і забезпечує більшу чистоту та зрозумілість.

Отже, іменна маршрутизація Flutter є гарним рішенням, яке дозволяє розробникам ефективно та гнучко керувати маршрутизацією в додатку.

2.3 Розгорнута постановка завдання

У роботі було поставлено задачу розробити програмне забезпечення для мобільного додатку, яке дозволить користувачам обмінюватися миттєвими

повідомленнями.

Опис функціональності:

1. Реєстрація користувача: користувачі зможуть створювати свій профіль, вказуючи ім'я користувача, електрону пошту пароль та інші дані, які можуть знадобитись для забезпечення ідентифікації користувачів.

2. Авторизація користувача: користувачі зможуть увійти в систему, використовуючи свою електронну пошту та пароль.

3. Обмін повідомленнями: користувачі зможуть обмінюватись текстовими повідомленнями один з одним в режимі реального часу.

4. Можливість обрати мову додатку Локалізація дозволяє додатку бути більш доступним та зрозумілим для користувачів з різних мовних середовищ. Процес локалізації може включати переклад і адаптацію текстових рядків.

5. Зрозумілий та приємний для користувача інтерфейс. Сторінки реєстрації користувача що міститиме поля для імені, електронною пошти, паролю та вибору мови. Сторінки авторизації. Сторінки зі списком користувачів та їх останніми повідомленнями. Сторінки чату з користувачем яка відображає діалог з полем для вводу повідомлень та короткою інформацією про співбесідника.

6. Середовище для збереження інформації про користувачів, чатів, повідомлень та можливість швидко та зручно маніпулювати цими даними .

7. Доступність додатку як для операційних систем для смартфонів Android так і для IOS. Можливість відтворення додатку на різних емуляторах.

Проект повинен дотримуватися принципів об'єктно-орієнтованого програмування – методології програмування, що заснована на представленні програми у вигляді сукупності об'єктів, кожен з яких є екземпляром певного класу, а класи утворюють ієрархію спадкування. А саме інкапсуляція, наслідування, абстракція та поліморфізм.

Наслідування (Inheritance): Даний принцип полягає у тому, що клас може успадковувати властивості та методи від батьківського класу. Це дозволяє

підвищити повторне використання коду, знизити його складність і спростити його підтримку.

Поліморфізм (Polymorphism): Поліморфізм дозволяє використовувати один і той же метод для обробки різних типів даних. Це забезпечує більш високу гнучкість програми і дозволяє забезпечити більш ефективну роботу з класами.

Інкапсуляція (Encapsulation): Цей принцип вимагає розмежування внутрішньої і зовнішньої інформації класу, тобто приховування внутрішньої реалізації від зовнішнього світу. Інкапсуляція дозволяє забезпечити безпеку даних і забезпечити більш гнучке керування.

Абстракція (Abstraction): Абстракція вимагає, щоб класи були створені настільки абстрактними, щоб їх можна було легко змінювати без втрати функціональності. Це забезпечує більш гнучке керування і дозволяє забезпечити більш ефективну роботу з класами.

Ці принципи взаємодіють між собою і доповнюють один одного, щоб забезпечити більш високу ефективність та гнучкість програмного забезпечення. Використання цих принципів дозволяє створювати програмне забезпечення, яке є більш гнучким, ефективним та легко змінюваним.

Крім виконання функціональності, що описана вище, важливим аспектом розробки програмного забезпечення є тестування. Для забезпечення якості програмного забезпечення рекомендується проводити різні види тестування, такі як модульне тестування та тестування на реальних користувачах. Тестування дозволить виявити та виправити помилки та недоліки, які можуть виникнути в процесі розробки та забезпечить високу якість продукту.

3.ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування мобільного додатку

Все функціонування системи даної роботи базується на декількох аспектах. Основним аспектом є архітектура представлення BLoC на потоках.

BLoC (Business Logic Component) – це патерн архітектури, що використовується в Flutter для розробки масштабованих та повторно використовуваних мобільних додатків. Його основною ідеєю є розділення бізнес-логіки від представлення даних та взаємодії з користувачем.

Архітектура BLoC містить три головні складові:

1. Потік подій (Events) – відображає дії користувача або зовнішніх подій, які мають вплив на стан додатку.

2. BLoC – це клас, що приймає вхідний потік подій та відображає стан додатку, що залежить від цих подій. Він може виконувати бізнес-логіку та взаємодіяти з різними джерелами даних (наприклад, базою даних або мережею) для отримання та збереження даних.

3. Потік стану (State) – відображає поточний стан додатку, що залежить від вхідного потоку подій та бізнес-логіки BLoC. Цей потік використовується для відображення даних у відповідних елементах інтерфейсу користувача.

Така архітектура дозволяє розділити бізнес-логіку та представлення даних, що забезпечує більшу масштабованість та повторне використання коду. Крім того, вона дозволяє легко тестувати окремі складові додатку та використовувати їх у різних проектах. Крім того, вона дозволяє підтримувати однакову структуру додатку незалежно від його розміру та складності, що забезпечує зручне та ефективне керування додатком у майбутньому.

У даному проекті архітектура представлення BLoC розділяється на:

1. RegisterBloc – що відповідає за логіку реєстрації користувача в додатку

та запису даних в Firestore та FirebaseAuth, містить наступні Стани(State) RegisterEmptyState, RegisterErrorState, UserRegisteredState (що містить в собі обєкт з детальною інформацією про користувача – User) .

2. LoginBloc – що відповідає за логіку авторизації користувача в додатку, що містить наступні Стани (State) LoginEmptyState, LoginErrorState, UserLoginedState (що містить в собі обєкт User).

3. UsersBloc – який відповідає за стягування актуальної інформації про користувачів та їх останніх повідомлень з Firestore. Містить наступні стани UserLoadingState, UserEmptyState, ListenLastMessage, UserLogOutState, UserLoadedState (де loadedUsers – це List з користувачами).

4. ChatRoomBloc – відовідає за перевірку на наявність чату між двома користувачами. Має наступні стани: ChatRoomEmptyState, ChatRoomErrorState, ChatRoomNewState, ChatRoomCreatedState, ChatRoomIdState (який має String з айді бесіди).

5. ChatBloc – відповідає за постійне відслідковування повідомленнь між двома користувачами та їх запис в Firestore. Має наступні стани ChatEmptyState, ChatErrorState, ChatListState (що містить List з повідомленнями, які в свою чергу складаються з самого повідомлення та айді відправника).

Також одним із основних аспектів є використання Single subscription Stream – це потік, на який можна підписатися багато разів, і він не закривається після отримання першого значення. Такий потік може використовуватися, наприклад, для передачі повідомлень між різними компонентами додатку. У додатку він використовується для прослуховування нових повідомлень які потім відображаються користувачеві.

Міжсторінкова навігація є одним з основних функціоналів проекту, яка представлена у вигляді іменної навігації, в основі якої у даному випадку є наступні шляхи:

1. ‘/login’ – що є початковим шляхом на сторінці авторизації (LoginPage), з якої є навігація через методи pushName на сторінку реєстрації (RegisterPage), та

на головну сторінку (MainPage), куди передаються дані користувача.

2. ‘/register’ – що є шляхом на сторінці реєстрації (RegisterPage), з якої є навігація через методи pop (повернення назад) до сторінки навігації, push до головної сторінки

3. ‘/main’ – що є шляхом до основної сторінки (MainPage) з переліком користувачів та їх останніх повідомлень, з якої є навігація до сторінки авторизації, до сторінки чату (ChatRoomPage), куди передаються дані про чат.

4. ‘/chatroom’ – що є шляхом до ChatRoomPage.

Важливим фактором також являється перелік основних віджетів (контент-модулів, що вбудовуються в додаток), які застосовуються в проекті для побудови інтерфейсу користувача.

Для початку зазначимо, що у Flutter є два основні типи віджетів: Stateless та Stateful.

StatelessWidget – це віджет, який не має стану та не може змінюватись під час рендерингу. Однак, коли стан змінюється ззовні, StatelessWidget перерендерюється.

StatefulWidget – це віджет, який може мати внутрішній стан та змінюватись під час рендерингу. Якщо стан змінюється, StatefulWidget перерендерюється, але при цьому його стан залишається незмінним.

initState() – це метод, який викликається один раз при створенні віджета Stateful і призначений для ініціалізації початкового стану віджета. В цьому методі можна ініціалізувати змінні, встановити початкові значення, завантажити дані з мережі та т.д.

setState() – це метод, який викликається після зміни стану віджета Stateful. Використовуючи цей метод, можна оновити стан віджета та перерендерити його, щоб відобразити зміни на екрані. Важливо пам'ятати, що setState() не змінює стан безпосередньо, а тільки позначає віджет для перерендерингу на наступному кадрі.

Далі йде перелік віджетів, які складають основу UI/UX частини проекту.

1. Container – використовується для обгортання інших віджетів, дозволяє налаштовувати їхню відстань, колір тла, форму та інші параметри. Наприклад, якщо потрібно вставити кнопку, використовуючи Container, можна налаштувати параметри контейнера, щоб створити потрібний розмір та відстань.

2. Row – використовується для організації віджетів в одному рядку, причому віджети займають рівномірну частину доступного місця. Він дозволяє розташовувати віджети горизонтально і змінювати їх порядок відображення. Наприклад, якщо потрібно вставити дві кнопки поруч, можна використовувати Row.

3. Column – використовується для організації віджетів в стовпчик, причому віджети займають рівномірну частину доступного місця. Він дозволяє розташовувати віджети вертикально і змінювати їх порядок відображення. Наприклад, якщо потрібно вставити два текстові блоки один під одним, можна використовувати Column.

4. Text – використовується для відображення тексту. Можна використовувати його для відображення різних шрифтів, розмірів та стилів тексту. Він дозволяє вставляти текст змінної довжини та форматувати його за потребою.

5. TextField – використовується для введення тексту користувачем. Він дозволяє користувачу вводити текст в поле введення, яке можна налаштувати для різних типів даних, таких як числа, адреси електронної пошти, паролі та інші. TextField також має вбудовану можливість валідації введеного тексту та автозаповнення. За допомогою TextField можна збирати дані від користувача, наприклад, при реєстрації, вході до облікового запису, пошуку та інших сценаріях.

6. Button – використовується для створення кнопок, на які можна натискати, щоб виконувати дії в додатку. Він може мати текст або іконку та може мати різні стилі та налаштування. Наприклад, можна створити кнопку "Відправити" для надсилання повідомлень або "Додати" для створення нового

елементу в додатку. Кнопки можуть бути використані для взаємодії користувача з додатком та виконання різноманітних дій, таких як збереження даних, переходи на інші сторінки та інші.

7. ListView – використовується для відображення списку елементів, який може бути скроллінгом. ListView може бути горизонтальним або вертикальним, в залежності від налаштувань. Є кілька способів створення ListView в Flutter, один з яких полягає в використанні конструктора ListView та передачі списку даних, які необхідно відобразити. ListView буде автоматично генерувати віджети для кожного елемента в списку та розміщувати їх на екрані.

8. BlocProvider – це віджет в Flutter, який дозволяє надавати доступ до об'єкту BLoC для дочірніх віджетів у дереві віджетів без необхідності явно передавати його через конструктор.

BlocProvider використовується з пакетом flutter_bloc. BlocProvider працює на основі взаємодії з двома класами: BlocProvider і Bloc. BlocProvider є посередником між Bloc та дочірніми віджетами. Він зберігає об'єкт Bloc та надає доступ до нього через дочірні віджети, які використовують BlocProvider.of (BuildContext context) для отримання доступу до Bloc.

9. Scaffold – це віджет в Flutter, який дозволяє створювати базовий макет сторінки, який містить заголовок, тулбар (AppBar), бічне меню (Drawer), навігаційні кнопки та інші важливі компоненти, які часто використовуються в мобільних додатках.

10. Expanded – це віджет, який використовується для розширення дочірнього віджета на розмір, що залишився від батьківського віджета. В інших словах, Expanded займає всю доступну вільну простору, який залишився після розміщення всіх інших віджетів у батьківському віджеті. Expanded використовуються, як правило, в комбінації з Row або Column, для того, щоб дочірні віджети розміщувалися по горизонталі або вертикалі відповідно. При цьому, якщо потрібно розмістити кілька віджетів в одному рядку або стовпці

11. BlocListener є віджетом, який слухає стан івентів, що видає бізнес-

логіка від Bloc або Cubit, і запускає відповідні дії при зміні стану.

BlocListener має два обов'язкові параметри: bloc та listener. Параметр bloc вказує на об'єкт Bloc або Cubit, який слухатиметься, а listener – це функція, яка буде виконуватися при зміні стану.

3.2 Розробка структурної схеми

На рисунку 3.1 зображено структурну міжсторінкову схему додатку. Структура сторінок додатку включає в себе 5 сторінок, послідовність їх появлення при певних діях у додатку та короткий опис вмісту сторінок.

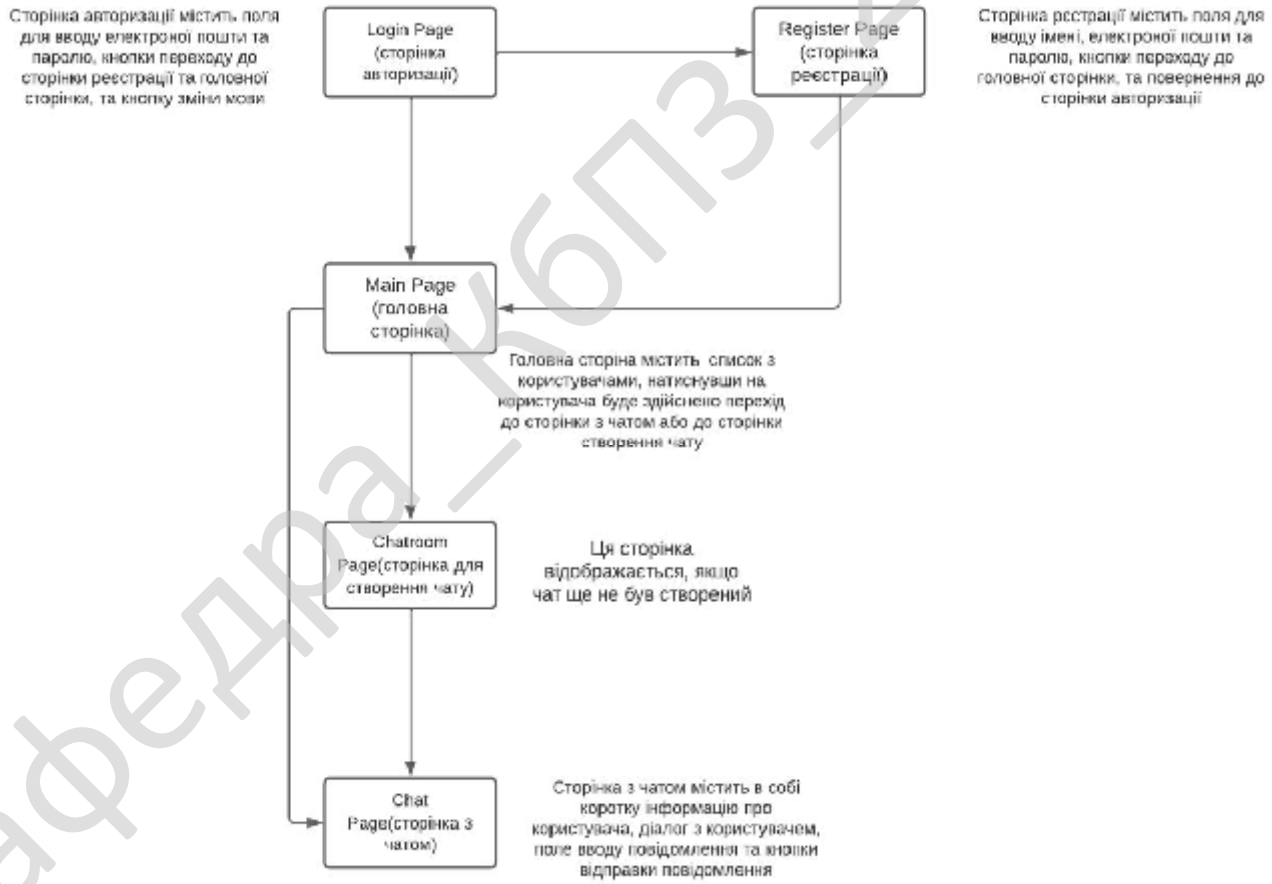

Рисунок 3.1 – Міжсторінкова структурна схема додатку

Кожна сторінка включає в себе окремий BLoC та ідентифікатор типу String, за яким відбувається міжсторінкова навігація, де початковою сторінкою є Login Page.

На рисунку 3.2 зображено структурну схему колекції користувачів у базі даних Firebase Firestore, яка вкючає в себе чітку ієрархію елементів колекції, їх тип та короткий опис елементу його призначення при потребі додаткового пояснення.

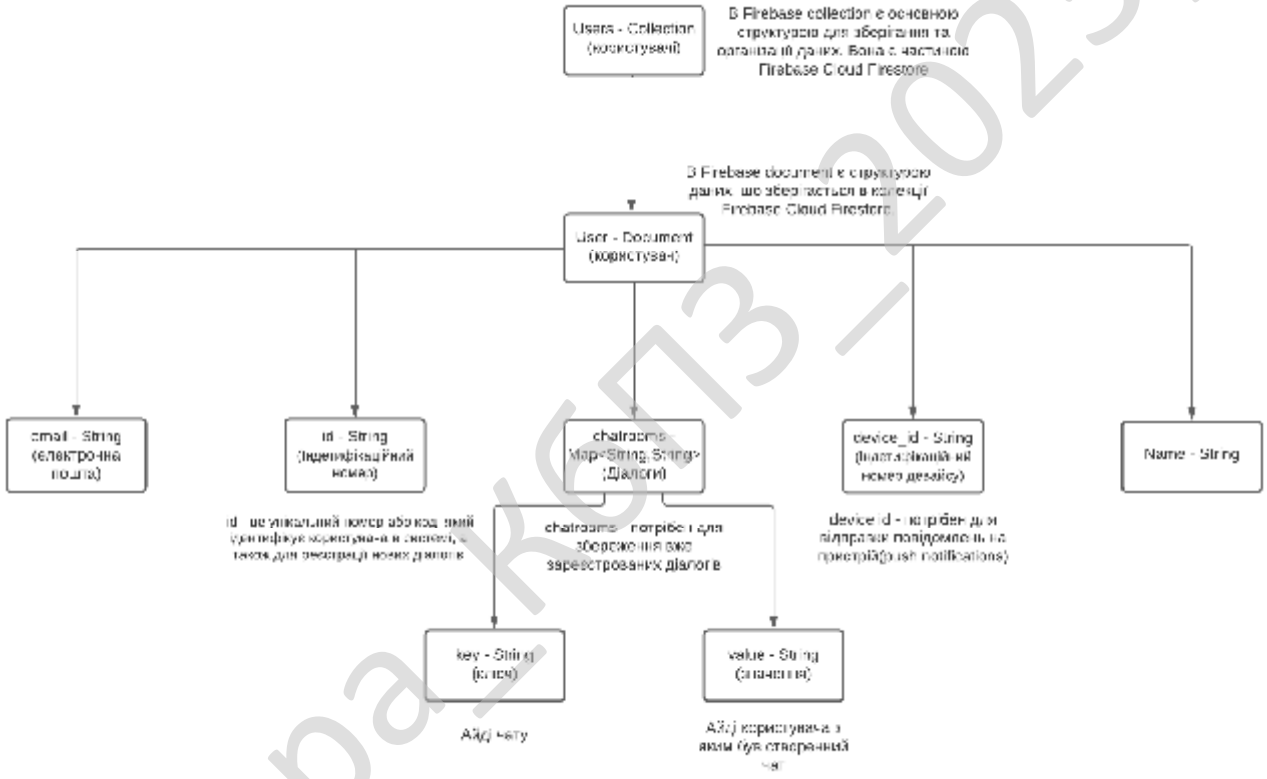

Рисунок 3.2 – Структурна схема колекції користувачів у базі даних Firebase Firestore

Важливо зазначити наступні моменти:

1. Map<String, String> – це тип даних в мові програмування Dart, що використовується для зберігання набору пар "ключ-значення" (key-value pairs). У цьому випадку, "String" вказує на тип даних, який буде використовуватись для

зберігання як ключа, так і значення.

2. Колекції є групою документів, які можуть містити різні типи даних, наприклад рядки, числа, об'єкти, масиви та інші. Кожен документ має унікальний ідентифікатор, що дозволяє легко звертатися до нього та змінювати його дані.

3. Колекції можуть містити будь-яку кількість документів, індексовані та доступні за допомогою запитів. Firebase Cloud Firestore забезпечує автоматичне синхронізування даних між всіма пристроями, що використовують Firebase SDK, що дозволяє розробникам створювати потужні додатки в реальному часі.

На рисунку 3.3 зображено структурну схему колекції чат-кімнат у базі даних Firebase Firestore, яка включає в себе чітку ієрархію елементів колекції, їх тип та короткий опис елементу його призначення при потребі додаткового пояснення.

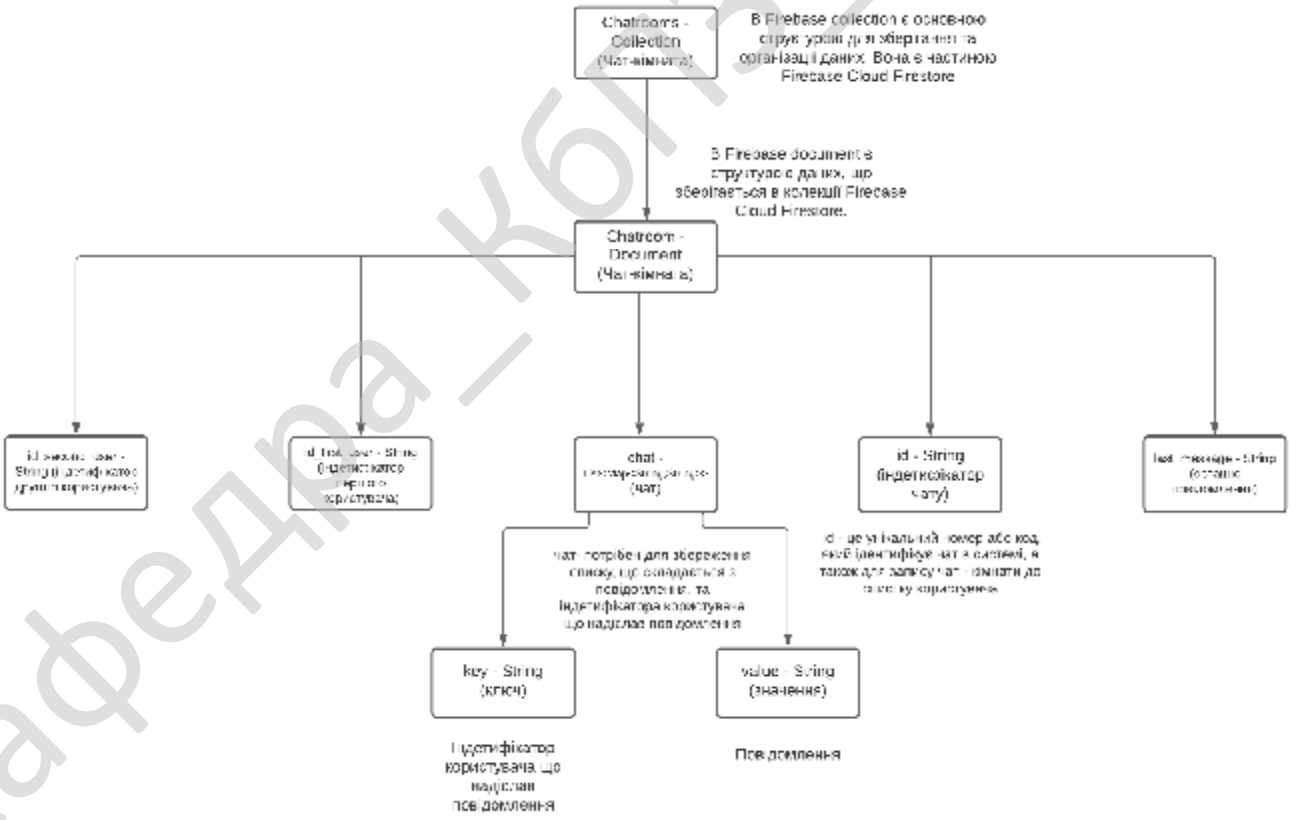

Рисунок 3.3 – Структурна схема колекції чат-кімнат у базі даних Firebase Firestore

List в мові Dart є об'єктом, який використовується для зберігання та управління даними у вигляді послідовності елементів. Він є одним з найбільш базових та потужних типів даних у мові Dart, що дозволяє зберігати дані будь-якого типу, такі як цілі числа, рядки, об'єкти та інші.

Він є динамічним та дозволяє додавати, видаляти та модифікувати елементи у списку під час його виконання. Він також підтримує ітерацію через елементи списку, що дозволяє здійснювати різноманітні операції з даними, що зберігаються у списку.

Основні методи, які можна використовувати з об'єктом List, включають add() (додати елемент), remove() (видалити елемент), insert() (вставити елемент), contains() (перевірити, чи містить список заданий елемент), sort() (відсортувати список) та багато інших.

Для повного розуміння важливо зазначити що дженеріком цього списку є Map про який було загадано раніше,

А Дженерік (Generics), в свою чергу, це механізм, який дозволяє створювати класи, функції та інші типи даних, які можуть працювати з будь-яким типом даних. Використання дженеріків дозволяє збільшити універсальність коду та зменшити кількість дублюючого коду, що спрощує розробку та підтримку програмного забезпечення.

Виходячи зі структурних схем, що зображенні на рисунках 3.2 та 3.3, можна затвердити, що основа структури бази даних цього додатку складається з двох основних елементів якими являються чат-кімнати та користувачі, що містять документи та поля необхідні для створення та подальшого удосконалення бізнес-логіки додатку.

3.3 Розробка функціональної схеми

На рисунку 3.4 зображено функціональну схему роботи архітектури представлення BLoC та усі процеси, які пов’язані з ним.

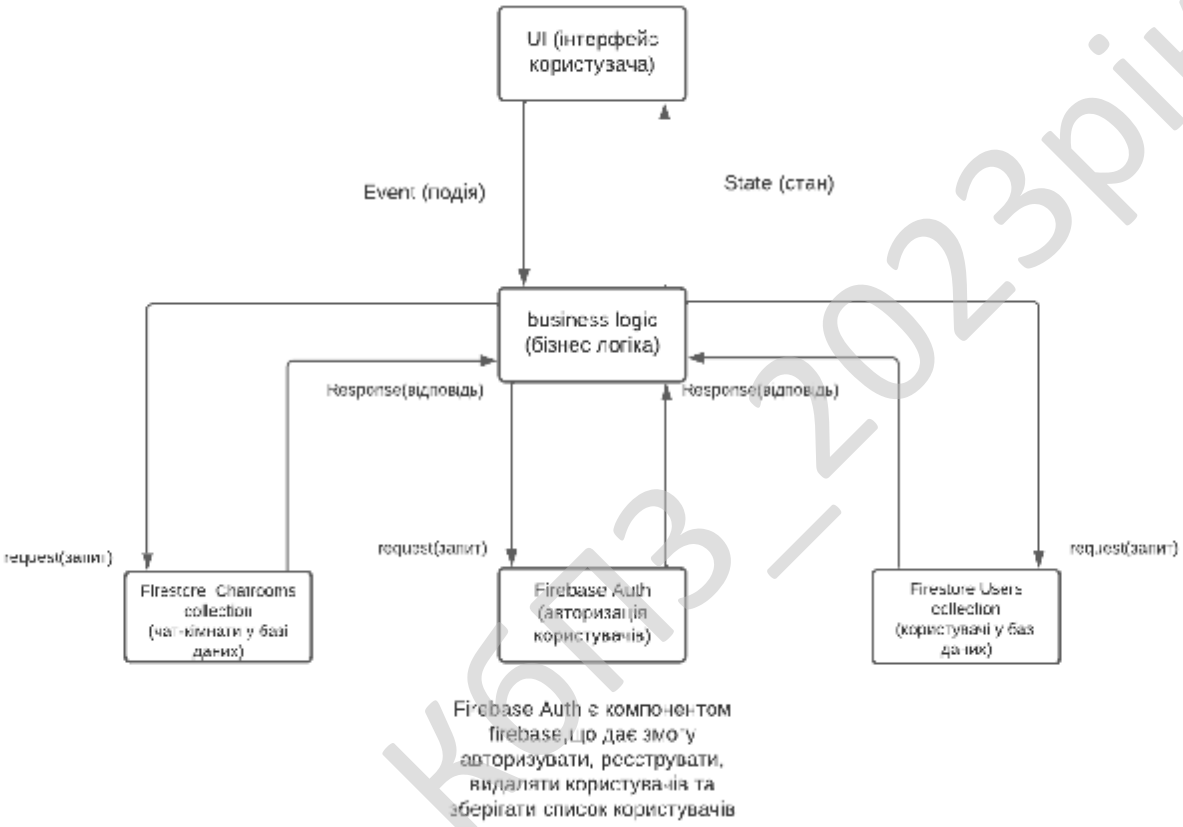

Рисунок 3.4 – Функціональна схема архітектури представлення BLoC (Business Logic of Component)

Важливо відзначити що, бізнес-логіка визначає, як програма обробляє дані, як вона приймає рішення та як вона взаємодіє з іншими системами або користувачами.

У даній схемі користувач надсилає так званий event (подію) до BLoC, який в свою чергу обробляє подію та надсилає request (запит) до бази даних Firestore або до сервісу авторизації Firebase Auth.

Firebase Auth приймає та обробляє інформацію, після цього відправляє response (відповідь) до BloC, який в свою чергу обробляє response та надсилає state (стан) до UI (користувацького інтерфейсу), на якому відбуваються певні зміни.

3.4 Розробка діаграми процесів

На рисунку 3.5 зображено діаграму процесів додатку для обміну миттєвими повідомленнями. Дана діаграма включає в себе усі процеси в додатку та їх взаємодію, де основні взаємодії відбуваються з автентифікацією користувача, головною сторінкою та сторінкою з чатом.

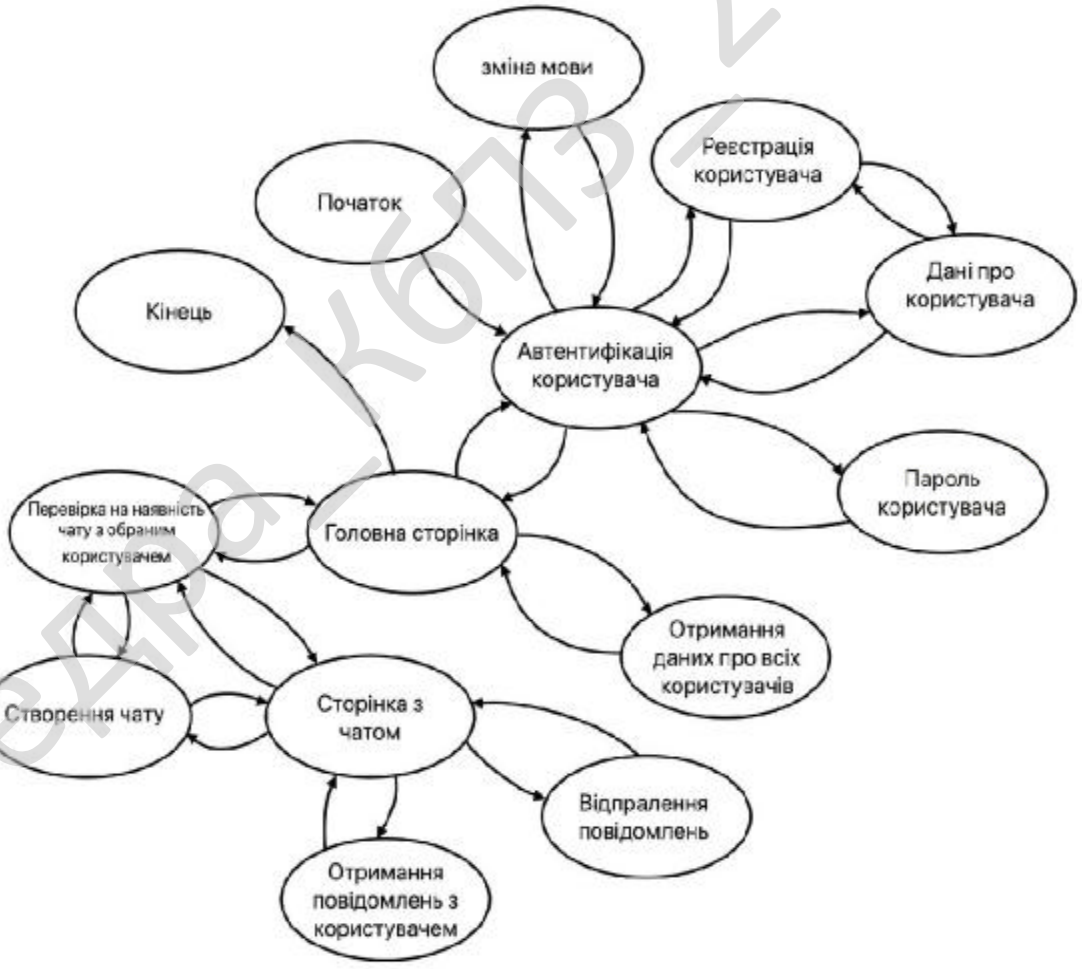

Рисунок 3.5 – Діаграма процесів додатку для обміну миттєвими повідомленнями

Початком діаграми являється автентифікація користувача, а кінцем кнопка виходу на головній сторінці, із додаткових процесів можна відмітити зміну мови користувача.

Більшість процесів можна поділити на http запити, де GET – це отримання даних про всіх користувачів, отримання повідомлень з користувачем, та POST – реєстрація користувача, автентифікація користувача, перевірка на наявність чату, створення чату, відправлення повідомлень.

4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ ТА ПРОГРАМНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування мобільного додатку

На рисунку 4.1 було зображено частину блок-схеми основної програми, у якій наведено процес авторизації та реєстрації користувачів додатку.

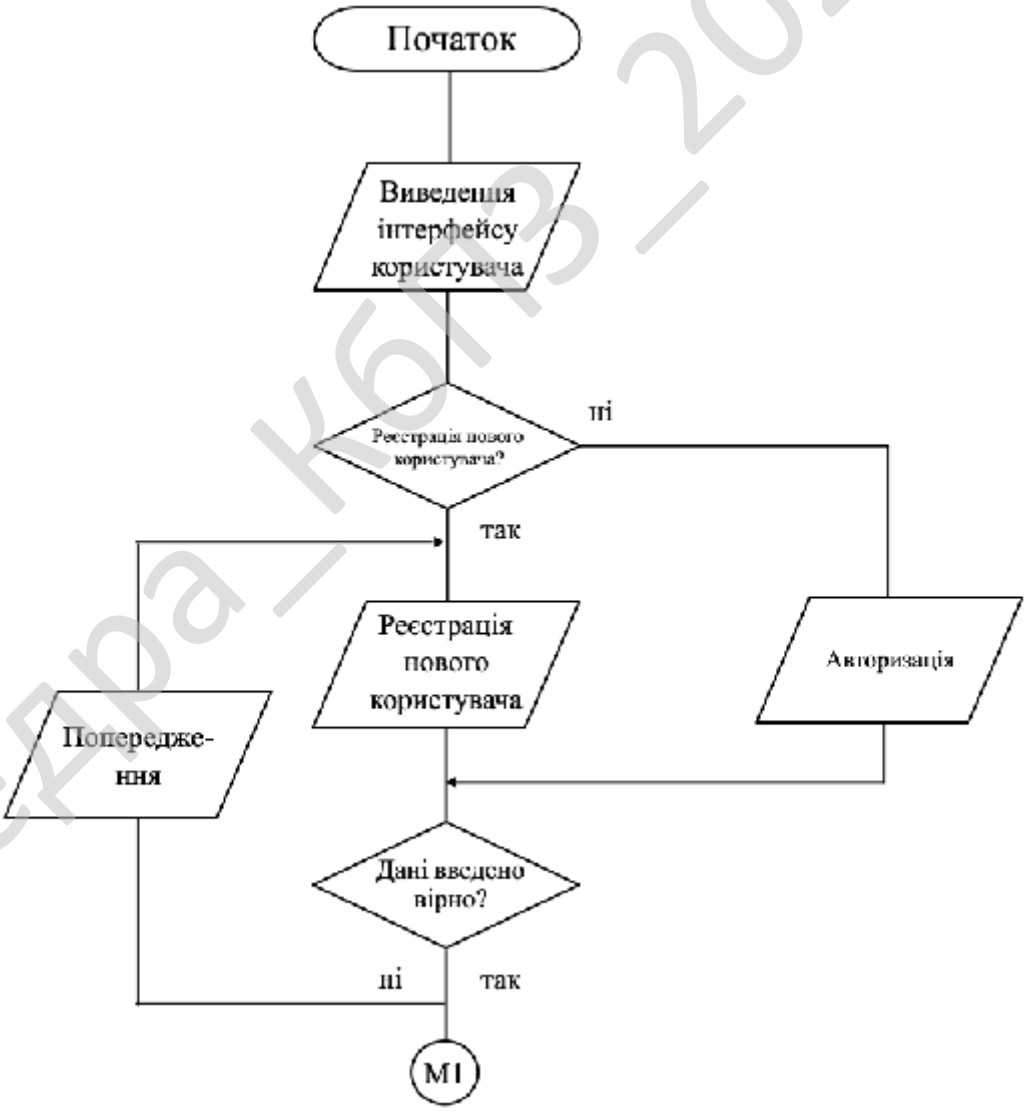

Рисунок 4.1 – Частина блок-схеми основної програми – авторизація та реєстрації користувачів

Важливо зазначити роботу окремих методів та функцій, а саме: реєстрація користувача, авторизація користувача.

Розберемо процес реєстрації користувача. Після того, як користувач ввів відповідні дані для реєстрації – вони передаються в BLoC реєстрації як подія, що іменується GetUserDataEvent, після чого дані передаються в окремий сервіс Firebase Auth, далі в BLoC іде перевірка статусу реєстрації, відбувається це наступним чином: метод signUp повертає дані,які повинні бути об'єктом класу User, інакше викликається метод для перевірки помилок під час введення даних _checkRegisterError.

Метод signUp сервісу Firebase Auth виглядає наступним чином:

```
Future<dynamic>signUp(String email,String password,String name) async {
    try {
      String? newToken = await FirebaseMessaging.instance.getToken();
      User? user = FirebaseAuth.instance.currentUser;
      await FirebaseAuth.instance.createUserWithEmailAndPassword(
          email: email, password: password);
      user = FirebaseAuth.instance.currentUser;
      if(user!=null)
        {
          FirebaseFirestore.instance.collection('users').add({
            'email': user.email,
            'name': name,
            'id': user.uid,
            'chatrooms': <String, String>{},
            'device_id': newToken
          });
        }
      return (user);
    }catch (e) {
      if (e is FirebaseAuthException) {
        return (e);
      }
  }

  }
```

newToken потрібен нам для стягування ідентифікатору девайсу, з якого відбувається реєстрація користувача. Методом сервісу Firebase Auth createUserWithEmailAndPassword здійснюємо реєстрацію. Після чого перевіряємо успішність реєстрації, якщо реєстрація пройшла успішно – вносимо користувача до бази даних.

Авторизація користувача відбувається майже ідентичним алгоритмом за виключенням внесення нового користувача. Замість цього відбувається оновлення даних про пристрій, з якого відбувалась авторизація.

Метод signIn у FirebaseAuth виглядає наступним чином:

```
Future<dynamic>SignIn(String email,String password)
  async {
    String? newToken = await FirebaseMessaging.instance.getToken();
    User? user = FirebaseAuth.instance.currentUser;
    try {
      await FirebaseAuth.instance.signInWithEmailAndPassword(
          email: email, password: password);
      user = FirebaseAuth.instance.currentUser;
      var getIdDoc = await FirebaseFirestore.instance
          .collection('users')
          .where('id', isEqualTo: user?.uid)
          .get();
      FirebaseFirestore.instance
          .collection('users')
          .doc(getIdDoc.docs.first.id)
          .set({'device_id': newToken}, SetOptions(merge: true));
      return (user);
    } catch (e) {
      if (e is FirebaseAuthException) {
        return e;
      }
    }
  }
```

Додатково відмітимо, що змінна ‘e’ є компонентом структури try/catch, що містить в собі повідомлення про помилку, яку ми повертаємо у разі невдалої реєстрації.

Зі сторони користувацького інтерфейсу для змоги написання такої інформації, як: електронна адреса, пароль та ім’я користувача Framework Flutter надає нам віджет TextField. Цей віджет має широкий спектр для редагування як візуальних, так і функціональних компонентів поля. Поля для авторизації були винесені в окремий віджет TextFieldsWidget.

```
Column(
      children: [
        SizedBox(height: screenHeight / 5.3),
        Text(LocaleKeys.Login.tr(), style: TextStyles.loginTextStyle),
        const SizedBox(height: 26),
        TextField(
          controller: emailController,
          decoration: InputDecoration(
              hintText: LocaleKeys.Email.tr(),
              suffixIcon: const Icon(Icons.alternate_email)),
```

```
        ),
        const SizedBox(height: 20),
        TextField(
            obscureText: true,
            controller: passwordController,
            decoration: InputDecoration(
                hintText: LocaleKeys.Password.tr(),
                suffixIcon: const Icon(Icons.password))),
        const SizedBox(height: 8.0),
      ],
    );
```

Цей віджет містить 3 поля з різними параметрами. Наприклад для поля, в якому користувач вводить свій пароль було застосовано параметр obscureText, який допомагає приховати зміст паролю під час його вводу.

Для надання більшої зрозумілості користувацькому інтерфейсу усі поля були підписані спеціальним текстом – hintText.

HintText в Flutter використовується для встановлення тексту-підказки (placeholder) у віджетах TextField або TextFormField. Цей текст з'являється у полі вводу та дає користувачеві контекст або інструкції щодо очікуваного формату або типу даних, які мають бути введені.

Наприклад, якщо у вас є поле для введення імені користувача, можна встановити hintText на значення "Введіть ваше ім'я". Цей текст-підказка відображатиметься у полі вводу до того моменту, поки користувач не почне вводити текст, після чого вона автоматично зникне або буде замінена введеним користувачем текстом.

Такий підказковий текст є важливим елементом користувацького інтерфейсу, оскільки він допомагає користувачам зрозуміти очікуваний формат вводу та спрощує навігацію в додатку.

На рисунку 4.2 зображено частину блок-схеми основної програми, що включає в себе основні процеси додатку на етапі відображення головної сторінки та створення чат-кімнати.

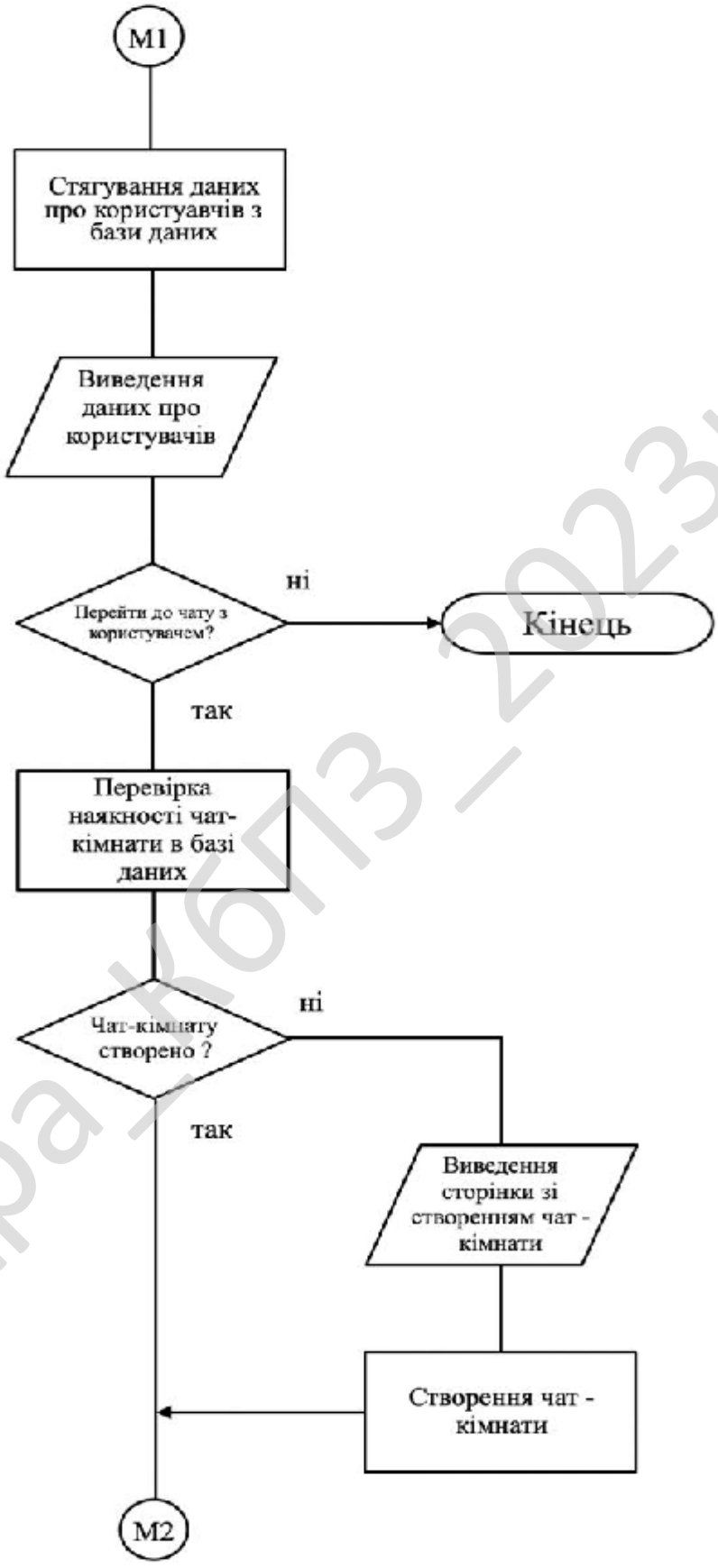

Рисунок 4.2 – Частина блок-схеми основної програми – бізнес-логіка на головній сторінці та сторінці створення чату

Опишемо роботу основних процесів:

1. Стягування та виведення даних про користувачів з бази даних.

Окремою подією UserLoadingEvent ми починаємо слухати потік (stream) з користувачами, яких, в свою чергу, отримуємо з сервісу Firestore. Після чого відбувається нова подія UserLoadedEvent, в якій відбувається додавання до динамічного масиву usersEndLastMessage окремим елементом користувач та його останнє повідомлення. Якщо діалог з користувачем ще не було розпочато, то в змінну останнього повідомлення відображаємо відповідне повідомлення “ви не розпочали діалог”.

```
if (event is UserLoadingEvent) {
      firestore.checkLastMessageAndGetListUsers().listen((snapshot) async {
        add(UserLoadedEvent((await firestore.getUsers()).docs));});
      id = event.userID;
    }
    if (event is UserLoadedEvent) {
      List<dynamic> usersAndLastMessage = <dynamic>[];
      try {
        for (int i = 0; i < event.docs.length; i++) {
          String lastMessage = LocaleKeys.You_did_not_start_dialog.tr();
          UserModel userInList = UserModel.fromJson(event.docs[i].data());
          var getLastMessage = await firestore.getChatroomWithId('$id-
${userInList.id}');
          if (getLastMessage.docs.isNotEmpty) {
            lastMessage = getLastMessage.docs.first.get('lastMessage');
          } else {
            getLastMessage = await firestore.getChatroomWithId('${userInList.id}-
$id');
            if (getLastMessage.docs.isNotEmpty) {
              lastMessage = getLastMessage.docs.first.get('lastMessage');
            }
          }
          usersAndLastMessage.add([userInList, lastMessage]);
        }
        yield UserLoadedState(usersAndLastMessage);
      } catch (_) {}
    }
  }
}
```

2. Перевірка наявності та створення чат-кімнат в базі даних.

Перевірка створення чат-кімнат відбувається за допомогою ChatRoomBloc. Спочатку з користувацького інтерфейсу надходе подія FindChatRoomEvent, який викликає метод _findUserChatRooms, що шукає за ідентифікаторами користувачів чат-кімнату та відправляє подію GetChatRoomEvent, яка, в свою чергу, відправляє

стан ChatRoomIdState, що містить ідентифікатор чат-кімнати. Для створення чат-кімнати надходить подія CreateChatRoomEvent, що містить в собі ідентифікатори користувачів. Викликається метод _addChat, що додає до бази даних чат-кімнату. Після цього викликається метод _addChatUser, що додає до списку чат-кімнат користувача ідентифікатор чат-кімнати та ідентифікатор співрозмовника.

```
if (event is CreateChatRoomEvent) {
      id = await _addChat(event.OurId, event.IdSecondUser);
      firestore.listenUsers().listen((snapshot) {
        for (int i = snapshot.docs.length - 1; i >= 0; i--) {
          if (snapshot.docs[i].get('id') == event.OurId) {
            _addChatToUser(id, event.IdSecondUser, snapshot, i);
          }
          if (snapshot.docs[i].get('id') == event.IdSecondUser) {
            _addChatToUser(id, event.OurId, snapshot, i);
          }
        }
      });
      yield ChatRoomIdState(id);
    } else if (event is FindChatRoomEvent) {
      _findUserChatRooms(event.ourUser.uid, event.secondUser.id);
    } else if (event is GetChatRoomEvent && event.chatroom.isEmpty) {
      yield ChatRoomNewState();
    } else if (event is GetChatRoomEvent) {
      for (var item in event.chatroom.entries) {
        if (item.value == event.secondUserId) {
          newRoom = false;
          yield ChatRoomIdState(item.key);
        }
      }
      if (newRoom) {
        yield ChatRoomNewState();
      }
    }
```

Зі сторони користувацького інтерфейсу кожен користувач відображається в окремо оформленому віджеті ListTile

ListTile в Flutter є віджетом, який використовується для відображення одного рядка інформації в списку або на панелі. Він зазвичай використовується в комбінації з ListView або ListView.builder для створення списку елементів.

ListTile надає стандартний вигляд рядка, який містить заголовок, додатковий підзаголовок, значок або зображення, іконку зліва або справа, а також можливість обробки натискання на елемент.

Цей віджет дозволяє швидко створювати списки зручного формату, такі як списки контактів, пункти меню, варіанти налаштувань тощо. Він також має

вбудовану підтримку для відображення спливаючих контекстних меню або інших дій, пов'язаних з елементом списку.

Завдяки гнучким параметрам і налаштуванням, ListTile може бути легко призначений для відображення різних типів даних та взаємодії з користувачем в мобільних додатках, розроблених з використанням Flutter.

```
ListTile(
                    tileColor: index.isEven
                        ? ColorStyle.EvenUserColor
                        : ColorStyle.NotEvenUserColor,
                    onTap: () {
                      Navigator.of(context).pushNamed('/chatroom', arguments: [
                        state.loadedUsers[index].first as UserModel,
                        dataUser
                      ]);
                    },
                    leading: Container(
                        height: 50,
                        width: 50,
                        decoration: BoxDecoration(
                          borderRadius: BorderRadius.circular(90),
                          color: index.isEven
                              ? ColorStyle.EvenAvatarColor
                              : ColorStyle.NotEvenAvatarColor,
                        ),
                        child: Center(
                            child: Text(
                          state.loadedUsers[index].first.name[0],
                          style: TextStyles.firstLetterOfNameTextStyle,
                        ))),
                    title: UserInfoWidget(
                      name: state.loadedUsers[index].first.name,
                    ),
                    subtitle: Text(
                      state.loadedUsers[index][1],
                      maxLines: 1,
                    ),
                  ),
                ),
              ),
            );
          },
        );
```

На рисунку 4.3 наведена частина блок-схеми основної програми, на якій були відображені основні процеси, які пов’язані з чатом додатку.

Рисунок 4.3 – Частина блок-схеми основної програми – чату додатку

Стягнення інформації про чат з бази даних введення та виведення повідомлень, оновлення чату в базі даних відбувається за допомогою ChatBloc. Спочатку з користувацького інтерфейсу надходить подія GetChatId, яка вказує на прослуховування потоку (stream) listenChatRoomById, яка в свою чергу являється методом сервісу firestore, відстежує зміни в динамічному масиві з повідомленнями та надсилає оновлений список повідомлень, що надходить в події GetReversedChat, яка в свою чергу містить в собі динамічний масив з повідомленнями та відправляє їх окремим станом ChatListState до користувацького інтерфейсу.

Відправка повідомлень відбувається за допомогою події SendMessage, що містить в собі метод _changeChatList, який приймає ідентифікатор чат-кімнати та нове повідомлення користувача. Оскільки змінився динамічний масив з повідомленнями – відправляється стан до користувацького інтерфейсу. Стан ChatListState, що містить в собі оновлений список повідомлень.

```
    if (event is GetChatId) {
      try {
        firestore.listenChatRoomById(event.chatId).listen((snapshot) async {
          add(GetReversedChat(snapshot.data()!['chat']));
        });
      } catch (e) {
        print(e);
      }
    }
    try {
      if (event is SendMessage) {
        _changeChatList(event.idChatRoom, event.messageAndAuth);
        yield ChatListState(_getNewReversedChatList(event.idChatRoom));
      }
    } catch (_) {}
    if (event is GetReversedChat) {
      var reversed = event.chat.reversed.toList();
      yield ChatListState(reversed);
    }
  }

  ///flip the chat for a better view of the message

  _getNewReversedChatList(String id) async {
    var documentUpdate = await firestore.getChatroomById(id);
    var chatUpdate = (documentUpdate.data()?['chat'] as List<dynamic>);
    return chatUpdate.reversed.toList();
  }

  ///open the message array and add a new message at the end
  ///consisting of the sender and the message itself
```

```
  void _changeChatList(String id, dynamic message) async {
    String to = '';
    var document = await firestore.getChatroomById(id);
    List<dynamic> chat = document.data()?['chat'];
    chat.add(message);
    var key;
    var messageAuth = message as Map<String, dynamic>;
    for (var item in messageAuth.entries) {
      message = item.value;
      key = item.key;
    }
    await _pushNotification(key, document, to, message);
    firestore.addNewMassageToFirestore(id, chat, message);
  }
```

Додамо також те, що процес віправки повідомлень тісно пов'язаний з процесом відправки повідомлень на пристрій (Push-notifications).

Зі сторони користувацького інтерфейсу віджетом ListView.builder відображається список повідомлень, який можна прокручувати (scroll) вгору.

```
ListView.builder(
                  reverse: true,
                  scrollDirection: Axis.vertical,
                  shrinkWrap: true,
                  itemCount: state.chat.length,
                  controller: _scrollController,
                  itemBuilder: (context, index) {
                    Map<String, dynamic> messageAuth =
                        state.chat[index] as Map<String, dynamic>;
                    return MessageWidget(
                        userId: userId,
                        auth: messageAuth.entries.first.key,
                        message: messageAuth.entries.first.value);
                  }),
                   )
```

Зі сторони користувацького інтерфейсу кожне повідомлення відображається в окремому контнейнері. Колір та позиція цього контейнера залежить від того, хто надіслав повідомлення. Наприклад, якщо повідомлення надіслав користувач, то воно буде відображатись з правої сторони екрану та буде блакитного кольору, а якщо повідомлення співрозмовника, то воно буде відображатись з лівої сторони екрану та буде фіолетового кольору.

Нагадаємо, що Container в Flutter – це віджет, що служить для створення і налаштування прямокутного контейнера, в якому можуть розміщуватись інші віджети. Він надає можливість контролювати розмір, вигляд, відступи та інші

властивості контейнера.

За допомогою Container можна вирівнювати, стилізувати та розміщувати вміст всередині нього. Цей віджет дозволяє задати фоновий колір, границі, тіні, радіуси кутів та інші стилістичні ефекти. Також Container надає можливість встановлювати розташування та розмір вмісту за допомогою параметрів, таких як padding, margin, width, height тощо.

Container є важливим будівельним блоком в Flutter і використовується для створення складних макетів та розміщення елементів на екрані. Його часто поєднують з іншими віджетами, такими як Text, Image, ListView і т.д., для створення власних інтерфейсів з багатошаровим дизайном.

Container – це потужний і гнучкий віджет, який дозволяє контролювати вміст додатку залежно від потреб і вимог дизайну.

У нашому випадку такий контейнер з усіма необхідними параметрами винесений в окремий віджет – MessageWidget.

```
Padding(
      padding: const EdgeInsets.all(10.0),
      child: Align(
        alignment: auth == userId ? Alignment.topRight : Alignment.topLeft,
        child: FittedBox(
          clipBehavior: Clip.hardEdge,
          child: Container(
              alignment:
                  auth == userId ? Alignment.topRight : Alignment.topLeft,
              decoration: BoxDecoration(
                  borderRadius: BorderRadius.circular(10),
                  color: auth == userId
                      ? ColorStyle.ourMessageColor
                      : ColorStyle.messageColor),
              child: Padding(
                padding: const EdgeInsets.all(8.0),
                child: Container(
                    constraints: const BoxConstraints(maxWidth: 250),
                    child: Text(
                        softWrap: true,
                        message,
                        style: TextStyles.messageTextStyle)),
              )),
        ),
      ),
    );
```

На рисунку 4.4 зображено блок-схему, яка відображає всі процеси, що пов’язані з push-повідомленнями.

Рисунок 4.4 – Блок-схема Push-повідомлень

Під час реєстрації додається ідентифікатор пристрою до бази даних, а при кожній авторизації він оновлюється. Цей ідентифікатор потрібен нам для того,

щоб надсилати повідомлення на пристрій. Відбувається це наступним чином: якщо користувач надіслав повідомлення, а його співрозмовник не був у цей час в додатку, то на його пристрій надходить повідомлення з іменем співрозмовника та текстом повідомлення.

Процес відправки повідомлення на пристрій відбувається за допомогою метода _pushNotification, який знаходиться в ChatBloc.

```
Future<void> _pushNotification(
      String key, var document, String to, String message) async {
    var user = await firestore.getUserWithId(key);
    if (user.docs.first.get('id') == document.data()?['id_first_user']) {
      to = document.data()?['id_second_user'];
    } else {
      to = document.data()?['id_first_user'];
    }
    var userTo = await firestore.getUserWithId(to);
    await PushNotification.push(
        to: userTo.docs.first.get('device_id'),
        title: "Message from ${user.docs.first.get('name')}",
        body: message);
  }
}
```

В свою чергу метод викликає інший метод push, що є частиною окремого класу PushNotification. Цей клас містить ключ серверу, який також потрібен для надсилання push-повідомлень. У метод push передаються 3 змінні: to – ідентифікатор пристрою користувача, якому надійде дане повідомлення, title – ім’я відправника повідомлення, body – зміст повідомлення. Метод містить http запит POST, в який передаються дані в форматі JSON, потім йде перевірка статусу відповіді. Якщо статус має код 200 – запит успішно надіслано, в іншому випадку – відображаємо помилку.

```
class PushNotification{
  static const String pushNotificationServerKey ="AAAAzmp6sfg:APA91bHYb6nFgJApW"
      "3H8iDiwzEfsciudHS-KGwraAqqle5E0547oUCfZAPWm2rFCnGbd-Eo1TTy3lRWYpQAEe3fgZ"
      "27xTgMNlzpWmpbm4vaTODSoMY1NAbPGVrkxQM19YTDkcrV_bTy-";
  static Future<void> push({required String to, required String title, required
String body}) async
  {
    try{
      http.Response response = await http.post(
          Uri.parse('https://fcm.googleapis.com/fcm/send'),
          headers: <String, String>{
            'Content-Type': 'application/json',
            'Authorization': 'key=$pushNotificationServerKey',
          },
```

```
          body: jsonEncode(PushNotificationModel(
              notification: Notification(title: title, body: body), to: to)));
      if (response.statusCode == 200) {
      } else {
        throw Exception();
      }
    }catch(e){
      throw Exception(e);
    }
  }
}
```

Отже, вище було наведено блок-схему основної програми, що детально описує алгоритми роботи додатку, а також блок-схему підпрограми, яка відповідає за відправку повідомлень на пристрій користувача.

Ці блок-схеми розкривають початковий алгоритм авторизації та реєстрації користувача, а також вказують на кінцевий етап – вихід з додатку.

Рисунок 4.2 відображає алгоритм, який зосереджений на виведенні даних про користувачів, навігації та виконанні додаткових операцій з чат-кімнатами. Рисунок 4.3 відображає алгоритм, який стосується виведення даних та надсилання повідомлень користувачу, а також взаємодії з базою даних.

Блок-схема підпрограми на рисунку 4.4 детально описує процес внесення даних про пристрій до бази даних, необхідних для успішної надсилання повідомлень на сам пристрій. В рамках цієї операції активно використовується компонент Firebase, а саме Cloud Message. Крім того, блок-схема включає умови, які регулюють надсилання повідомлень.

Варто зауважити декілька важливих моментів. Усі операції, пов'язані з роботою з базою даних та бізнес-логікою, виконуються виключно через архітектурний підхід, відомий як представлення BLoC.

Дані про пристрій містять у собі рядок (String), який функціонує як унікальний ідентифікатор пристрою. Саме повідомлення включає в себе текстовий контент повідомлення, а також електронну адресу відправника. Всі ці компоненти взаємодіють, щоб забезпечити ефективне та безперебійне функціонування системи надсилання повідомлень на пристрій користувача.

4.2 Захист розробленого програмного забезпечення

Додаток розповсюджується за ліцензією GNU General Public License (GPL), ця ліцензія передбачає можливість вільного використання програмного забезпечення, зміну та поширення відкритого коду, а також зобов'язання зберігати авторські права та джерело походження додатку.

Детальніше про GNU General Public License (GPL) – це вільна ліцензія на програмне забезпечення, створена Фондом вільного програмного забезпечення (FSF) для забезпечення вільного доступу до програмного забезпечення та захисту прав користувачів. Ліцензія передбачає, що додаток, що розповсюджується під GPL, повинен бути вільним та відкритим для користувачів.

Основні принципи GPL полягають у тому, що програмне забезпечення, розповсюджуване під GPL, повинно мати відкритий вихідний код та можливість вільної зміни та поширення цього коду користувачами. Крім того, вільна ліцензія також передбачає, що користувачі можуть використовувати програмне забезпечення безкоштовно, та зобов'язує розробників зберігати авторські права та джерело походження додатку.

Одна з головних особливостей GPL полягає в тому, що вона передбачає взаємність ліцензій. Це означає, що якщо програмне забезпечення, що розповсюджується під GPL, використовується у проекті з відкритим кодом, то цей проект також повинен бути розповсюджуваний під GPL.

Таким чином, GPL забезпечує збереження відкритості та вільності відкритого ПЗ, навіть якщо воно використовується у комерційних проектах.

GPL також забезпечує захист прав користувачів, забороняючи зміну ліцензії після того, як програмне забезпечення було розповсюджено під GPL. Це означає, що якщо додаток був розповсюджений під GPL, то користувачі можуть використовувати та модифікувати його, розповсюджувати у вигляді оригінального додатку або зміненої версії, а також використовувати його у комерційних проектах, за умови дотримання вимог ліцензії.

Однак, GPL також має свої обмеження. Наприклад, згідно з GPL, весь код, що поєднується з додатком, також повинен бути вільним та розповсюджуватись під GPL. Це може бути проблемою у випадку, якщо ви використовуєте код, що розповсюджується під іншими ліцензіями, наприклад, пропрієтарними.

У будь-якому випадку, GPL є однією з найпопулярніших ліцензій для вільного програмного забезпечення, що забезпечує відкритість та вільність програмного забезпечення, а також захист прав користувачів та розробників. Якщо ви плануєте розповсюджувати свій додаток під GPL, слід ретельно вивчити цю ліцензію та її вимоги. Імплементація GPL полягає в додаванні до програмного забезпечення файлу ліцензії GPL та забезпеченні вільного доступу до вихідного коду програми. Крім того, потрібно дотримуватися вимог ліцензії, зокрема, забезпечувати вільний доступ до вихідного коду та дозволяти його зміну та розповсюдження під GPL.

Одним зі способів імплементації GPL є додавання файлу ліцензії до кореневої папки програмного забезпечення, а також до кожного файлу з вихідним кодом, які включені до проекту. У файлі ліцензії повинні бути вказані умови ліцензії та права користувачів.

Крім того, слід забезпечити доступ до вихідного коду програми через Інтернет або на диску, який постачається разом із програмним забезпеченням. Вихідний код повинен бути доступний в тому ж форматі, в якому було розроблено ПЗ, тобто якщо програма була написана на мові програмування Dart, то вихідний код також має бути наданий у форматі коду на мові Dart.

Імплементація GPL також вимагає, щоб весь код, який поєднується з програмним забезпеченням, був також доступний під GPL або іншою вільною ліцензією. Це може включати бібліотеки, фреймворки або будь-який інший код, який використовується в проекті.

У даному випадку усі вихідні дані лежать у веб-сервісі GitHub, та є змога переглянути та експортувати усі файли. Усі бібліотеки, що використовуються у додатку є відкритими для імплементування.

5 МЕТОДИКА ВПРОВАДЖЕННЯ МОБІЛЬНОГО ДОДАТКУ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Розроблений додаток має інтуїтивно зрозумілий інтерфейс та функціонал, що задовольняє потреби користувачів.

На рисунку 5.1 зображено сторінку чат-кімнати, яка має градієнтний фон. Верхня частина сторінки складається з кнопки назад, при натисканні якої ми повертаємось до головної сторінки, ім’я та електронну пошту співрозмовника. Кожен чат розпочинається з повідомлення ‘Start the dialog’.

Нижня частина сторінки має поле для вводу повідомлення та кнопки відправлення повідомлення.

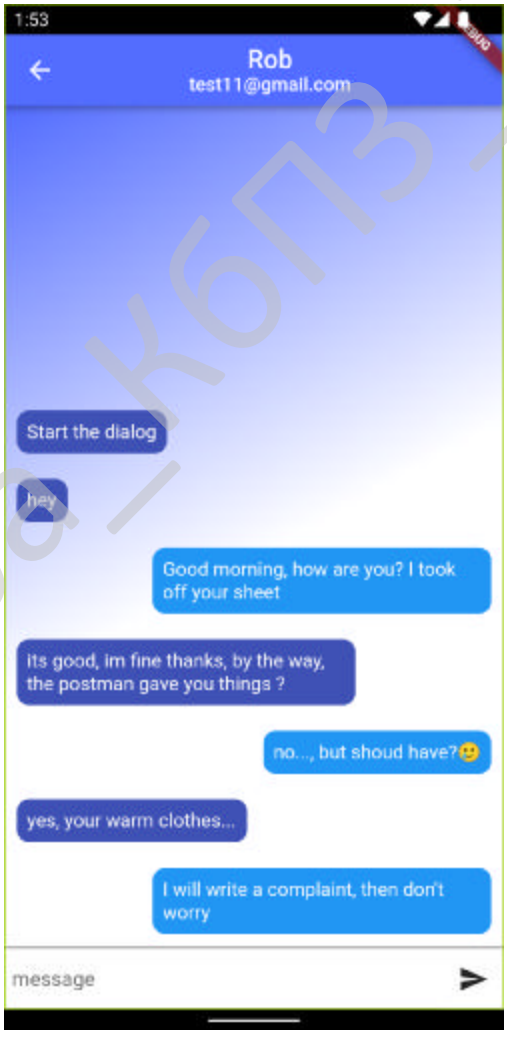

Рисунок 5.1 – Чат-кімната додатку

На рисунку 5.2 зображено головну сторінку з контактами користувача. Верхня частина головної сторінки складається з електронної пошти користувача та кнопки виходу з облікового запису.

Основна частина сторінки складається зі списку контактів, де окремий елемент складається з білого контейнеру, що містить ім’я користувача та останнє повідомлення, якщо воно є. Якщо діалог не було розпочато – додаток повідомляє користувача про це.

Рисунок 5.2 – Головна сторінка додатку

На рисунку 5.3 зображено Push-повідомлення розробленого додатку, що складається з імені користувача, що його надіслав та вмісту повідомлення.

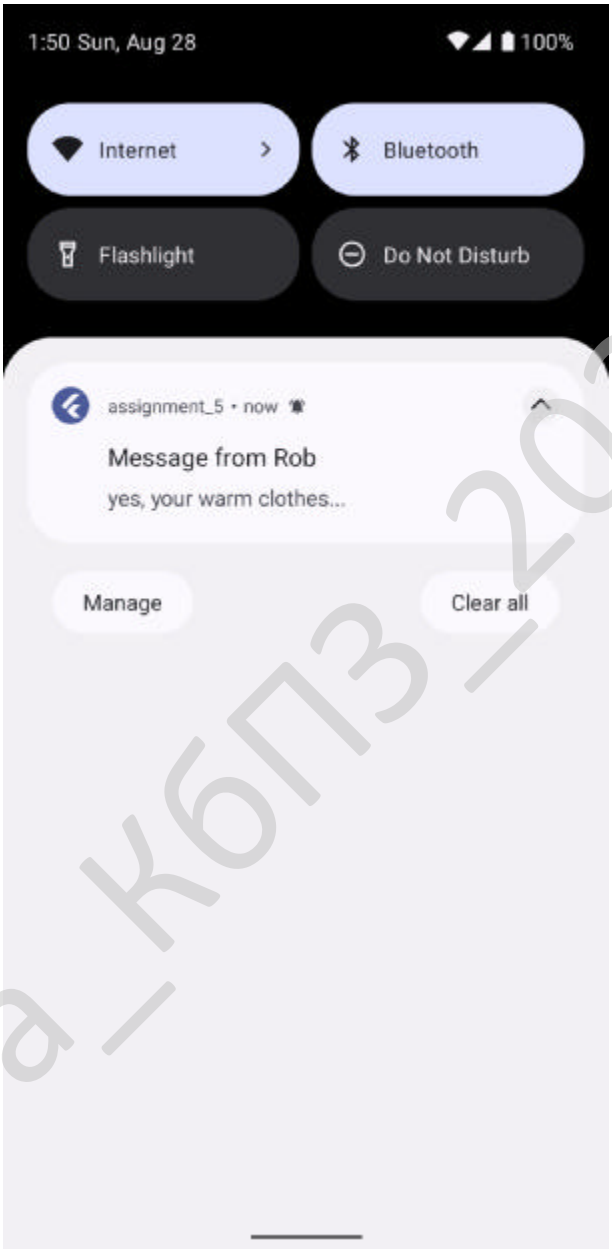

Рисунок 5.3 – Push-повідомлення розробленого додатку

6 ОСНОВНІ ВИСНОВКИ

У даній бакалаврській роботі було розроблено програмне забезпечення для обміну миттєвими повідомленнями, у якому вирішуються такі основні задачі:

1. Побудова інтерфейсу користувача (UI/UX).

2. Побудова бізнес-логіки додатку. Підключення бізнес-логіки додатку до інтерфейсу користувача.

3. Підключення бази даних до додатку, побудова ієрархії даних, підключення бази даних до бізнес-логіки додатку.

Було досліджено додатки побудовані за допомогою фреймворку Flutter, що наближенні до обміну миттєвими повідомленнями.

Був проведений огляд віджетів, компонентів, для роботи з Flutter. Було освоєно мову програмування Dart. Також освоєно та застосовано у проекті такі технології як Push Notifications – для відправки повідомлень на пристрій, Localization – для зміни мови додатку. Були створенні схеми структури додатку, схеми функціоналу додатку, діаграма процесів додатку, блок-схеми додатку та підпрограми.

У результаті проведених тестів було встановлено, що програмне забезпечення працює швидко та стабільно, а також забезпечує надійну передачу повідомлень між користувачами.

Крім того, варто зазначити, що розроблене програмне забезпечення для обміну миттєвими повідомленнями на Flutter є вільним програмним забезпеченням та розповсюджується за ліцензією GNU General Public License (GPL), що дозволяє користувачам отримати доступ до вихідного коду та зробити внесок. до подальшого розвитку додатку.

Також, для реалізації функціональності додатку було використано архітектуру представлення BLoC (Business Logic Component), що дозволяє розділити бізнес-логіку від інтерфейсу користувача та забезпечити більшу

масштабованість та підтримку коду.

Для подальшого розвитку додатку заплановано проведення рефакторингу з метою забезпечення дотримання принципів SOLID, DRY, KISS, що дозволить збільшити масштабованість та підтримку коду, зменшити залежність між компонентами додатку та забезпечити його більшу стабільність та зручність розробки.

Крім того, у подальшому розвитку додатку планується додавання нової функціональності, такої як реалізація відео- та голосового дзвінків, розвиток системи сповіщень та покращення інтерфейсу користувача.

Отже, розроблене програмне забезпечення є не тільки ефективним інструментом для обміну миттєвими повідомленнями, але й відкритим джерелом, що забезпечує можливість подальшого розвитку та покращення.

Для подальшого розвитку додатку заплановано проведення рефакторингу з метою забезпечення дотримання принципів SOLID, що дозволить збільшити масштабованість та підтримку коду, зменшити залежність між компонентами додатку та забезпечити його більшу стабільність та зручність розробки.

Крім того, у подальшому розвитку додатку планується додавання нової функціональності, такої як реалізація відео- та голосових дзвінків, розвиток системи сповіщень та покращення інтерфейсу користувача.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Учасники проектів Вікіпедіа. JSON – вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/JSON (дата звернення: 14.04.2022).

2. Bloc | dart package. Dart packages. URL: https://pub.dev/packages/bloc (date of access: 29.04.2022).

3. Bloc state management library. Bloc State Management Library. URL: https://bloclibrary.dev/#/ (date of access: 29.04.2022).

4. Bracha G. Dart programming language. Addison-Wesley Longman, Incorporated, 2015. 224 p.

5. Building layouts. Flutter documentation | Flutter. URL: https://docs.flutter.dev/ui/layout/tutorial (date of access: 12.04.2022).

6. Cloud_firestore | flutter package. Dart packages. URL: https://pub.dev/packages/cloud_firestore (date of access: 14.04.2022).

7. Contributors to Wikimedia projects. Don't repeat yourself – Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Don't_repeat_yourself#:~:text=The%20DRY%20principle%20is%20stated,their%20book%20The%20Pragmatic%20Programmer. (date of access: 28.04.2021).

8. Contributors to Wikimedia projects. KISS principle – Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/KISS_principle#:~:text=First%20seen%20partly%20in%20American,unnecessary%20complexity%20should%20be%20avoided. (date of access: 09.01.2022).

9. Dart D. Coin inventory log: inventory log book record sheet – inventory management control – simple inventory tracker – personal management – large 8. 5 X 11 inches. Independently Published, 2019.

10. Easy_localization | flutter package. Dart packages.

URL: https://pub.dev/packages/easy_localization (date of access: 05.04.2021).

11. Firebase | firebase cloud messaging. Firebase. URL: https://firebase.google.com/docs/cloud-messaging?hl=ru (дата звернення: 02.02.2022).

12. Firebase. Firebase. URL: https://firebase.google.com/ (date of access: 27.04.2022).

13. Flutter R. Uncomfortable australia: looking back on xavier herbert's masterwork poor fellow my country. Independently Published, 2019. 161 p.

freeCodeCamp.org. How to explain object-oriented programming concepts to a 6-year-old. freeCodeCamp.org. URL: https://www.freecodecamp.org/news/object-oriented-programming-concepts-21bb035f7260/ (date of access: 28.04.2021).

14. JAGAN DIGITECH. How to read data from local JSON file and open the next page from ListView | Json Parse in Flutter, 2021. YouTube. URL: https://www.youtube.com/watch?v=Vll8ARb05qY (date of access: 13.04.2022).

15. Json_serializable | dart package. Dart packages. URL: https://pub.dev/packages/json_serializable (date of access: 03.04.2022).

Proto Coders Point. How to read local json file in flutter & show json data in listview builder, 2021. YouTube. URL: https://www.youtube.com/watch?v=vpFxmFl5cUk (date of access: 21.04.2022).

16. The dart programming language. Boston, Massachusetts, USA : Addison-Wesley, 2016. 201 p.

The importance of SOLID design principles. BMC Blogs. URL: https://www.bmc.com/blogs/solid-design-principles/#:~:text=SOLID%20is%20an%20acronym%20that,some%20important%20benefits%20for%20developers. (date of access: 29.04.2021).

17. Yazeed AlKhalaf. How to use easy localization package? | flutter, 2021. YouTube. URL: https://www.youtube.com/watch?v=cposNqIsyAY (date of access: 08.01.2023).

18. A solid guide to SOLID principles | baeldung. Baeldung.

URL: https://www.baeldung.com/solid-principles (date of access: 29.04.2021).

19. Basic principles of object-oriented programming. Sitecore | Kentico | Umbraco | nopCommerce & .NET ontwikkeling en interim consultancy, support en training – ParTech. URL: https://www.partech.nl/en/publications/2020/10/basic-principles-of-object-oriented-programming (date of access: 29.04.2022).

20. Codemy.com. Maps in dart – learn dart programming 5, 2022. YouTube. URL: https://www.youtube.com/watch?v=pEhpWxQxLrw (date of access: 29.04.2021).

21. Create a bloc provider. Home | Flutter by Example. URL: https://flutterbyexample.com/lesson/create-a-bloc-provider (date of access: 29.04.2021).

22. Dart programming language. Dart programming language | Dart. URL: https://dart.dev/ (date of access: 26.04.2022).

23. Dart:ui library – Dart API. Flutter – Dart API docs. URL: https://api.flutter.dev/flutter/dart-ui/dart-ui-library.html (date of access: 24.04.2021).

24. Erinç Y. K. The SOLID principles of object-oriented programming explained in plain english. freeCodeCamp.org. URL: https://www.freecodecamp.org/news/solid-principles-explained-in-plain-english/ (date of access: 29.04.2021).

25. Expanded class – widgets library – Dart API. Flutter – Dart API docs. URL: https://api.flutter.dev/flutter/widgets/Expanded-class.html (date of access: 20.04.2021).

26. Firebase authentication. Firebase. URL: https://firebase.google.com/docs/auth?hl=ru (date of access: 01.10.2022).

27. Flutterly. #3 – Flutter BLoC Concepts – BlocProvider, BlocBuilder, BlocListener | BLoC – from Zero to Hero, 2020. YouTube. URL: https://www.youtube.com/watch?v=NqUx-NfTts4 (date of access: 29.04.2021).

28. Injectable | dart package. Dart packages.

URL: https://pub.dev/packages/injectable (date of access: 29.04.2022).

29. Singleton – Dart API docs. Dart packages. URL: https://pub.dev/documentation/singleton/latest/ (date of access: 29.04.2021).

30. The Net Ninja. Firebase authentication tutorial #1 – introduction, 2019. YouTube. URL: https://www.youtube.com/watch?v=aN1LnNq4z54 (date of access: 05.04.2021).

31. What is the dart programming language? A guide to its applications. Emeritus Online Courses. URL: https://emeritus.org/blog/coding-dart-programming-language/ (date of access: 28.04.2021).

32. AboutDialog class – material library – Dart API. Flutter – Dart API docs. URL: https://api.flutter.dev/flutter/material/AboutDialog-class.html (date of access: 17.04.2023).

33. AlertDialog class – material library – Dart API. Flutter – Dart API docs. URL: https://api.flutter.dev/flutter/material/AlertDialog-class.html (date of access: 29.04.2021).

34. AlignmentGeometry class – painting library – Dart API. Flutter – Dart API docs. URL: https://api.flutter.dev/flutter/painting/AlignmentGeometry-class.html (date of access: 20.04.2023).

35. Asynchronous programming: futures, async, await. Dart programming language | Dart. URL: https://dart.dev/codelabs/async-await (date of access: 03.03.2023).

36. Asynchronous programming: streams. Dart programming language | Dart. URL: https://dart.dev/tutorials/language/streams (date of access: 22.04.2022).

37. Dalvi P. Microtasks and event loop in dart. Medium. URL: https://medium.com/@purvajg/microtasks-and-event-loops-in-dart-9f5863f031d8 (date of access: 16.04.2022).

38. Dart future. Dart Tutorial. URL: https://www.darttutorial.org/dart-tutorial/dart-future/ (date of access: 15.04.2022).

39. Dynamic links for firebase | flutterfire. FlutterFire | FlutterFire.

URL: https://firebase.flutter.dev/docs/dynamic-links/overview/ (date of access: 02.02.2023).

40. Firebase_dynamic_links | flutter package. Dart packages. URL: https://pub.dev/packages/firebase_dynamic_links (date of access: 12.04.2022).

41. Flexible class – widgets library – Dart API. Flutter – Dart API docs. URL: https://api.flutter.dev/flutter/widgets/Flexible-class.html (date of access: 29.04.2021).

42. Future class – dart:async library – Dart API. Dart – Dart API docs. URL: https://api.dart.dev/be/176281/dart-async/Future-class.html (date of access: 20.04.2022).

43. ListTile class – material library – Dart API. Flutter – Dart API docs. URL: https://api.flutter.dev/flutter/material/ListTile-class.html (date of access: 12.02.2023).

44. Microsoft. Visual studio code – code editing. redefined. Visual Studio Code – Code Editing. Redefined. URL: https://code.visualstudio.com/ (date of access: 07.03.2021).

45. Navigate to a new screen and back. Flutter documentation | Flutter. URL: https://docs.flutter.dev/cookbook/navigation/navigation-basics (date of access: 11.04.2022).

46. Navigate with named routes. Flutter documentation | Flutter. URL: https://docs.flutter.dev/cookbook/navigation/named-routes (date of access: 29.04.2021).

47. Navigation and routing. Flutter documentation | Flutter. URL: https://docs.flutter.dev/ui/navigation (date of access: 20.04.2022).

48. Navigation. Flutter documentation | Flutter. URL: https://docs.flutter.dev/cookbook/navigation (date of access: 18.04.2022).

49. Programming Point. Dart programming course in just 4 hours – dart programming tutorial 2022, 2022. YouTube. URL: https://www.youtube.com/watch?v=YUANhchdQwI (date of access:

20.04.2022).

50. Stream | dart package. Dart packages. URL: https://pub.dev/packages/stream (date of access: 01.09.2022).

51. What are streams in Dart?. Educative: Interactive Courses for Software Developers. URL: https://www.educative.io/answers/what-are-streams-in-dart (date of access: 29.04.2021).

Додаток А
(обов'язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку програмного забезпечення мобільного додатку для обміну миттєвими повідомленнями

2 Підстава для розробки

Підставою для розробки служить завдання на кваліфікаційну бакалаврську роботу, видане на кафедрі кібербезпеки та програмного забезпечення (нак. №7-02 від 05.01.2023 року).

3 Мета та призначення розробки

Метою кваліфікаційної бакалаврської роботи є розробка програмного забезпечення мобільного додатку для обміну миттєвими повідомленнями.

4 Джерела розробки

Джерелом цієї кваліфікаційної бакалаврської роботи є стосовна до теми література і існуючі аналоги.

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

– розробка програмної частин системи, а також розробка взаємодії системи з ОС та з користувачем;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– Обмін миттєвими повідомленнями;
– цілісність даних у процесі роботи та при зберіганні;
– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення не повинно мати обмежень на версію драйверів та операційної системи.

5.4 Вимоги до архітектури

Компонент, що розробляється повинен використовувати системні засоби та апаратні засоби, що на даному етапі розвитку обчислювальної техніки найбільше поширені.

5.5 Вимоги до надійності

Програмні модулі написані по всім правилам, які стосуються стандартних викликів процедур, функцій, методів і форм, визначених технічною документацією на середовище розробки.

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

– температура повітря: 19-20 град. по Цельсію;

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

Програмне забезпечення повинно бути реалізоване на ПЕОМ архітектури IBM PC, працювати на Емуляторах Android та пристроях з цією ОС.

5.8 Вимоги до інформаційної і програмної сумісності

Переносність програмного забезпечення повинна бути забезпечена за рахунок його реалізації стандартного інтерфейсу взаємодії з ОС, що працюють під управлінням ОС Windows 10.

5.8.1 Обладнання

Комп'ютер Intel® Celeron/8 Mb/1.2 Gb/SVGA 14" 1Mb або сумісні з ним.

5.8.2 Мова програмування

Програму розроблено на мові програмування Dart.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

Програмна продукція повинна бути представлена у виді опису структури даних, схем та опису алгоритму, а також текстів вихідних модулів програмного забезпечення згідно ЄСПД .

7 Перелік документів, що розробляються

– Структурна схема колекції користувачів – 1 аркуш.
у базі даних Firebase Firestore
– Структурна схема колекції чат-кімнат – 1 аркуш.
у базі даних Firebase Firestore
– Міжсторінкова структурна схема додатку – 1 аркуш.
– Функціональна схема архітектури
представлення BLoC роботи системи – 1 аркуш.
– Діаграма процесів додатку – 1 аркуш.
для обміну миттєвими повідомленнями
– Блок-схема основної програми – 1 аркуші.
– Блок-схема Push-повідомлень – 1 аркуш.
– Пояснювальна записка – 61 аркуш.

8 Етапи розробки

8.1 Збір і обробка інформації по темі кваліфікаційної бакалаврської роботи. Постановка задачі на виконання кваліфікаційної бакалаврської роботи (складання ТЗ).

8.2 Проведення досліджень або експериментальних робіт для уточнення основних положень кваліфікаційної бакалаврської роботи.

8.3 Розробка функціональних схем, блок схем алгоритмів роботи програмного забезпечення.

8.4 Побудова схем взаємодії даних.

8.5 Створення прототипу ПЗ.

8.6 Віднаходження ПЗ, аналіз отриманих результатів.

8.7 Оформлення пояснювальної записки і виконання робіт по графічній частині.

9 Порядок контролю та приймання

9.1 Подання кваліфікаційної бакалаврської роботи на попередній захист 22.05.2023 р.

9.2 Подання кваліфікаційної бакалаврської роботи на захист 2.06.2023 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи

за першим (бакалаврським) рівнем вищої освіти

_____ Мелешко Є.В.

Програмне забезпечення мобільного додатку для обміну миттєвими повідомленнями

Лістинг програми

Код документу 12

Носій: USB-флеш-накопи́чувач

Загальна кількість аркушів: 31

Літера: РП

Кропивницький – 2023 року

// main.dart – Базовий файл що містить підключення бази даних, локалізації та віджету що є основою - MatirialApp який знаходиться в StatelessWidget App

```
import 'package:easy_localization/easy_localization.dart';
import 'package:flutter/material.dart';
import 'package:firebase_core/firebase_core.dart';
import 'app.dart';
import 'generated/codegen_loader.g.dart';


void main() async{
  WidgetsFlutterBinding.ensureInitialized();
  await EasyLocalization.ensureInitialized();
  await Firebase.initializeApp();
  runApp( EasyLocalization(
      assetLoader: const CodegenLoader(),
      supportedLocales:  const [Locale('en'), Locale('ru')],
      path: 'assets/translation',
      fallbackLocale:  const Locale('en'),
      child: const App()
  ),);
}
```

// app.dart – файл що містить підключення міжсторінкову маршрутизацію та віджету що є основою - MatirialApp

```
import 'package:assignment_5/pages/chat_room/chat_room_page.dart';
import 'package:assignment_5/pages/login_page/login_page.dart';
import 'package:assignment_5/pages/main_page/main_page.dart';
import 'package:assignment_5/pages/register_page/register_page.dart';
import 'package:easy_localization/easy_localization.dart';
import 'package:flutter/material.dart';


class App extends StatelessWidget {
  const App({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      localizationsDelegates: context.localizationDelegates,
      supportedLocales: context.supportedLocales,
      locale: context.locale,
      routes: {'/main' :(context,{arguments})=>const MainPage(),
      '/register' : (context) => const RegisterPage(),
      '/login' : (context,{arguments}) => const LoginPage(),
      '/chatroom':(context,{arguments}) =>const ChatRoomPage()},
      initialRoute: '/login',
    );
  }
}
```

// login_page.dart – файл що знаходиться за відповідним шляхом pages/login_page/login_page.dart містить в собі сторіну авторизації

```
import 'package:assignment_5/generated/locale_keys.g.dart';
import
'package:assignment_5/pages/login_page/widgets/sign_in_button_widget.dart';
import 'package:assignment_5/pages/login_page/widgets/text_fields_widget.dart';
import 'package:easy_localization/easy_localization.dart';
import 'package:flutter/material.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
```

```
import '../../bloc/login_bloc/login_bloc.dart';

class LoginPage extends StatefulWidget {
  const LoginPage({Key? key}) : super(key: key);
  @override
  State<LoginPage> createState() => LoginPageState();
}

class LoginPageState extends State<LoginPage> {
  static String message = '';
  static final emailController = TextEditingController();
  static final passwordController = TextEditingController();

  @override
  void didChangeDependencies() async{
    if (FirebaseAuth.instance.currentUser != null) {
      Future.delayed(Duration.zero, () {
        Navigator.of(context).pushReplacementNamed('/main',
            arguments: FirebaseAuth.instance.currentUser);
      });
    }
    super.didChangeDependencies();
  }

  @override
  Widget build(BuildContext context) {
    var screenHeight = (MediaQuery.of(context).size.height);
    return BlocProvider<LoginBloc>(
      create: (context) => LoginBloc(),
      child: SafeArea(
        child: Scaffold(
          appBar: AppBar(
              leading: IconButton(
                onPressed: changeLanguage,
                icon: const Icon(Icons.language),
              ),
              title: Text(LocaleKeys.Authorization.tr())),
          body: SingleChildScrollView(
            child: Padding(
              padding: const EdgeInsets.all(16.0),
              child: Column(
                children: [
                  TextFieldsWidget(
                      screenHeight: screenHeight,
                      emailController: emailController,
                      passwordController: passwordController),
                  Column(
                    mainAxisAlignment: MainAxisAlignment.spaceAround,
                    children: [
                      const SizedBox(
                        height: 16,
                      ),
                      const SignInButtonWidget(),
                      const SizedBox(
                        height: 16,
                      ),
                      TextButton(
                          onPressed: () {
                            Navigator.of(context)
                                .pushReplacementNamed('/register');
                            setState(() {});
                          },
                          child: Text(LocaleKeys.Sign_Up.tr())),
                    ],
                  )
                ],
              ),
            ),
          ),
        ),
```

```
        ),
      ),
    );
  }

  void changeLanguage() {
    if (context.locale == const Locale('en')) {
      context.setLocale(const Locale.fromSubtags(languageCode: 'ru'));
    } else if (context.locale == const Locale('ru')) {
      context.setLocale(const Locale('en'));
    }
  }
}
```

// dialog_builder_widget.dart – файл що знаходиться за відповідним шляхом pages/login_page/widgets/dialog_builder_widget.dart та містить в віджет що відповідає за відображення діалогів

```
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';

import '../login_page.dart';

class ShowMessage{
  static Route<Object?> dialogBuilderWidget(BuildContext context,
      Object? arguments) {
    return DialogRoute<void>(
      context: context,
      builder: (BuildContext context) =>
          AlertDialog(
            title: Text(LoginPageState.message),
            actions: [
              ElevatedButton(
                  onPressed: () {
                    Navigator.of(context).pop();
                  },
                  child: const Text('ok'))
            ],
          ),
    );
  }
}
```

//sign_in_button_widget.dart – файл що знаходиться за відповідним шляхом pages/login_page/widgets/sign_in_button_widget.dart та містить в віджет з кнопокою авторизації

```
import 'dart:async';

import 'package:assignment_5/bloc/login_bloc/login_event.dart';
import
'package:assignment_5/pages/login_page/widgets/dialog_builder_widget.dart';
import 'package:easy_localization/easy_localization.dart';
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';

import '../../../bloc/login_bloc/login_bloc.dart';
import '../../../bloc/login_bloc/login_state.dart';
import '../../../generated/locale_keys.g.dart';
import '../../../models/user_model.dart';
import '../../../util/decorations/decorations.dart';
import '../login_page.dart';

class SignInButtonWidget extends StatelessWidget {
  static const double widthButton=300;
  const SignInButtonWidget({Key? key}) : super(key: key);

  @override
```

```
  Widget build(BuildContext context) {
    context.locale;
    final LoginBloc loginBloc = BlocProvider.of<LoginBloc>(context);
    BlocProvider.of<LoginBloc>(context).initialState;
    return BlocBuilder<LoginBloc, LoginState>(builder: (context, state) {
      if (state is UserLoginedState) {
        Future.delayed(Duration.zero, () {
          Navigator.of(context)
              .pushReplacementNamed('/main', arguments: state.user);
        });
      }
      if (state is LoginErrorState) {
        loginBloc.add(SetInitialEvent());
        LoginPageState.message = state.message;
        scheduleMicrotask(() => Navigator.of(context)
            .restorablePush(ShowMessage.dialogBuilderWidget));
      }
      return Container(
        decoration: Decorations.buttonDecoration,
        child: SizedBox(
          width: widthButton,
          child: ElevatedButton(
            style: ElevatedButton.styleFrom(primary: Colors.transparent,
              onSurface: Colors.transparent,
              shadowColor: Colors.transparent,),
            onPressed: () async {
              UserLogin userModel = UserLogin(
                  LoginPageState.emailController.text,
                  LoginPageState.passwordController.text);
              loginBloc.add(GetUserDataEvent(userModel));
              LoginPageState.emailController.text = "";
              LoginPageState.passwordController.text = "";
            },
            child: Text(LocaleKeys.Sign_In.tr()),
          ),
        ),
      );
    });
  }
}
```

// text_fields_widget.dart – файл що знаходиться за відповідним шляхом pages/login_page/widgets/text_fields_widget.dart та містить в віджет що відповідає за поля вводу інформації

```
import 'package:easy_localization/easy_localization.dart';
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';

import '../../../generated/locale_keys.g.dart';
import '../../../util/text_styles/text_styles.dart';

class TextFieldsWidget extends StatelessWidget {
  final double screenHeight;
  final emailController;
  final passwordController;

  const TextFieldsWidget(
      {Key? key,
      required this.screenHeight,
      required this.emailController,
      required this.passwordController})
      : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Column(
      children: [
        SizedBox(height: screenHeight / 5.3),
```

```
        Text(LocaleKeys.Login.tr(), style: TextStyles.loginTextStyle),
        const SizedBox(height: 26),
        TextField(
          controller: emailController,
          decoration: InputDecoration(
              hintText: LocaleKeys.Email.tr(),
              suffixIcon: const Icon(Icons.alternate_email)),
        ),
        const SizedBox(height: 20),
        TextField(
            obscureText: true,
            controller: passwordController,
            decoration: InputDecoration(
                hintText: LocaleKeys.Password.tr(),
                suffixIcon: const Icon(Icons.password))),
        const SizedBox(height: 8.0),
      ],
    );
  }
}
```

// register_page.dart – файл що знаходиться за відповідним шляхом pages/register_page/register_page.dart та містить в віджет що відповідає за відображення діалогів

```
import 'package:assignment_5/bloc/register_bloc/register_bloc.dart';
import
'package:assignment_5/pages/register_page/widgets/button_register_widget.dart';
import
'package:assignment_5/pages/register_page/widgets/text_fields_widget.dart';
import 'package:easy_localization/easy_localization.dart';
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';

import '../../generated/locale_keys.g.dart';
import '../../util/text_styles/text_styles.dart';

class RegisterPage extends StatelessWidget {
  const RegisterPage({Key? key}) : super(key: key);

  static final emailController = TextEditingController();
  static final passwordController = TextEditingController();
  static final nameController = TextEditingController();

  @override
  Widget build(BuildContext context) {
    var screenHeight = (MediaQuery.of(context).size.height);
    return BlocProvider<RegisterBloc>(
      create: (context) => RegisterBloc(),
      child: SafeArea(
        child: Scaffold(
        appBar: AppBar(
          title: Text(LocaleKeys.Authorization.tr()),
        ),
        body: SingleChildScrollView(
          child: Padding(
            padding: const EdgeInsets.all(16.0),
            child: Column(
              children: [
                SizedBox(height: screenHeight / 5.8),
                Text(LocaleKeys.Register.tr(),
                    style: TextStyles.registerTextStyle),
                TextFieldsWidget(
                    nameController: nameController,
                    emailController: emailController,
                    passwordController: passwordController),
                Column(
                  mainAxisAlignment: MainAxisAlignment.spaceAround,
                  children: [
```

```
                    const SizedBox(
                      height: 16,
                    ),
                    const ButtonRegisterWidget(),
                    const SizedBox(
                      height: 16,
                    ),
                    TextButton(
                        onPressed: () {
                          Navigator.of(context)
                              .pushReplacementNamed('/login', arguments: null);
                        },
                        child: Text(LocaleKeys.Go_Back.tr()))
                  ],
                )
              ],
            ),
          ),
        ),
      )),
    );
  }
}
```

// button_register_widget.dart – файл що знаходиться за відповідним шляхом pages/register_page/widgets/button_register_widget.dart

```
import 'dart:async';

import
'package:assignment_5/pages/login_page/widgets/dialog_builder_widget.dart';
import 'package:assignment_5/pages/register_page/register_page.dart';
import 'package:easy_localization/easy_localization.dart';
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';

import '../../../bloc/register_bloc/register_bloc.dart';
import '../../../bloc/register_bloc/register_event.dart';
import '../../../bloc/register_bloc/register_state.dart';
import '../../../generated/locale_keys.g.dart';
import '../../../models/user_model.dart';
import '../../../util/decorations/decorations.dart';
import '../../login_page/login_page.dart';

class ButtonRegisterWidget extends StatelessWidget {
  const ButtonRegisterWidget({
    Key? key,
  }) : super(key: key);

  @override
  Widget build(BuildContext context) {
    final RegisterBloc registerBloc = BlocProvider.of<RegisterBloc>(context);

    return BlocBuilder<RegisterBloc, RegisterState>(
      builder: (context, state) {
        if (state is UserRegisteredState) {
          Future.delayed(Duration.zero, () {
            Navigator.of(context)
                .pushReplacementNamed('/main', arguments: state.user);
          });
        }
        if (state is RegisterErrorState) {
          registerBloc.add(SetInitialRegisterEvent());
          LoginPageState.message = state.message;
          scheduleMicrotask(() => {
                Navigator.of(context)
                    .restorablePush(ShowMessage.dialogBuilderWidget)
              });
```

```
        }
        return Container(
            decoration: Decorations.buttonDecoration,
            child: SizedBox(
                width: 300,
                child: ElevatedButton(
                    style: ElevatedButton.styleFrom(
                      primary: Colors.transparent,
                      onSurface: Colors.transparent,
                      shadowColor: Colors.transparent,
                    ),
                    onPressed: () {
                      UserRegister userModel = UserRegister(
                          RegisterPage.nameController.text,
                          RegisterPage.emailController.text,
                          RegisterPage.passwordController.text);
                      registerBloc.add(GetUserDataEvent(userModel));
                      RegisterPage.emailController.text = "";
                      RegisterPage.passwordController.text = "";
                      RegisterPage.nameController.text = "";
                    },
                    child: Text(LocaleKeys.Sign_Up.tr()))));
      },
    );
  }
}
```

// text_fields_widget.dart – файл що знаходиться за відповідним шляхом pages/register_page/widgets/text_fields_widget.dart та містить в віджет що відповідає за поля вводу інформації

```
import 'package:easy_localization/easy_localization.dart';
import 'package:flutter/material.dart';

import '../../../generated/locale_keys.g.dart';

class TextFieldsWidget extends StatelessWidget {
  final nameController;
  final emailController;
  final passwordController;

  const TextFieldsWidget(
      {Key? key,
      required this.nameController,
      required this.emailController,
      required this.passwordController})
      : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Column(
      children: [
        const SizedBox(height: 26),
        TextField(
            controller: nameController,
            decoration: InputDecoration(
                hintText: LocaleKeys.Name.tr(),
                suffixIcon: const Icon(Icons.account_circle_rounded))),
        Padding(
          padding: const EdgeInsets.symmetric(vertical: 8.0),
          child: TextField(
              controller: emailController,
              decoration: InputDecoration(
                  hintText: LocaleKeys.Email.tr(),
                  suffixIcon: const Icon(Icons.alternate_email))),
        ),
        TextField(
            obscureText: true,
            controller: passwordController,
```

```
            decoration: InputDecoration(
                hintText: LocaleKeys.Password.tr(),
                suffixIcon: const Icon(Icons.password))),
        const SizedBox(height: 8.0),
      ],
    );
  }
}
```

// main_page.dart – файл що знаходиться за відповідним шляхом pages/main_page/main_page.dart містить в собі головну сторінку з користувачами

```
import 'package:assignment_5/pages/main_page/widgets/card_widget.dart';
import 'package:assignment_5/pages/main_page/widgets/log_out.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';

import '../../bloc/user_bloc/users_bloc.dart';
import '../../util/decorations/decorations.dart';

class MainPage extends StatefulWidget {
  const MainPage({Key? key}) : super(key: key);

  @override
  State<MainPage> createState() => _MainPageState();
}


class _MainPageState extends State<MainPage> {

  @override
  Widget build(BuildContext context) {
    final dataUser = (ModalRoute.of(context)?.settings.arguments) as User;
    return BlocProvider<UserBloc>(
      create: (context) => UserBloc(),
      child: SafeArea(
          child: Scaffold(
        appBar: AppBar(
            title: Row(
          children: [
            Expanded(child: Text('${dataUser.email}')),
            const LogOut(),
          ],
        )),
        body: Container(
          height: double.maxFinite,
          decoration: Decorations.backgroundMainDecoration,
          child: CardWidget(
            dataUser: dataUser,
          ),
        ),
      )),
    );
  }
}
```

// card_widget.dart – файл що знаходиться за відповідним шляхом pages/main_page/widgets/card_widget.dart містить в собі віджет для відображення одного користувача

```
import 'package:assignment_5/bloc/user_bloc/user_event.dart';
import 'package:assignment_5/models/user_model.dart';
import 'package:assignment_5/pages/main_page/widgets/user_info_widget.dart';
import 'package:assignment_5/util/colors/colors_style.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';

import '../../../bloc/user_bloc/user_state.dart';
```

```
import '../../../bloc/user_bloc/users_bloc.dart';
import '../../../util/text_styles/text_styles.dart';

class CardWidget extends StatelessWidget {
  final User dataUser;

  const CardWidget({Key? key, required this.dataUser}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    final UserBloc userBloc = BlocProvider.of<UserBloc>(context);
    userBloc.add(UserLoadingEvent(dataUser.uid));
    return BlocBuilder<UserBloc, UserState>(builder: (context, state) {
      if (state is UserLoadedState) {
        return ListView.builder(
          shrinkWrap: true,
          itemCount: state.loadedUsers.length,
          itemBuilder: (context, index) {
            return Padding(
              padding: const EdgeInsets.only(left: 12.0,right:
12.0,top:8,bottom: 8),
              child: Center(
                  child: Container(
                decoration: BoxDecoration(
                  borderRadius: BorderRadius.circular(20),
                  color: index.isEven
                      ? ColorStyle.EvenBorderColor
                      : ColorStyle.NotEvenBorderColor,
                ),
                child: ListTile(
                  tileColor: index.isEven
                      ? ColorStyle.EvenUserColor
                      : ColorStyle.NotEvenUserColor,
                  onTap: () {
                    Navigator.of(context).pushNamed('/chatroom', arguments: [
                      state.loadedUsers[index].first as UserModel,
                      dataUser
                    ]);
                  },
                  leading: Container(
                      height: 50,
                      width: 50,
                      decoration: BoxDecoration(
                        borderRadius: BorderRadius.circular(90),
                        color: index.isEven
                            ? ColorStyle.EvenAvatarColor
                            : ColorStyle.NotEvenAvatarColor,
                      ),
                      child: Center(
                          child: Text(
                        state.loadedUsers[index].first.name[0],
                        style: TextStyles.firstLetterOfNameTextStyle,
                      ))),
                  title: UserInfoWidget(
                    name: state.loadedUsers[index].first.name,
                  ),
                  subtitle: Text(
                    state.loadedUsers[index][1],
                    maxLines: 1,
                  ),
                ),
              )),
            );
          },
        );
      }
      return const Center(child: CircularProgressIndicator(color:
ColorStyle.indicatorColor));
    });
```

```
  }
}
```

// log_out.dart – файл що знаходиться за відповідним шляхом pages/main_page/widgets/log_out.dart містить в собі віджет кнопки виходу з поточного профілю

```
import 'package:assignment_5/bloc/user_bloc/user_event.dart';
import 'package:assignment_5/util/colors/colors_style.dart';
import 'package:easy_localization/easy_localization.dart';
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';

import '../../../bloc/user_bloc/user_state.dart';
import '../../../bloc/user_bloc/users_bloc.dart';
import '../../../generated/locale_keys.g.dart';

class LogOut extends StatelessWidget {
  const LogOut({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    final UserBloc userBloc = BlocProvider.of<UserBloc>(context);
    return BlocListener<UserBloc, UserState>(
      listener: (context, state) {
        if (state is UserLogOutState) {
          Navigator.pushReplacementNamed(context, '/login');
        }
      },
      child: TextButton(
        onPressed: () {
          userBloc.add(UserLogOutEvent());
        },
        child: Text(
          style: const TextStyle(color: ColorStyle.LogOutButtonColor),
          LocaleKeys.Log_Out.tr(),
        ),
      ),
    );
  }
}
```

// user_info_widget.dart – файл що знаходиться за відповідним шляхом pages/main_page/widgets/user_info_widget.dart містить в собі віджет з інформацією про користувача

```
import 'package:flutter/cupertino.dart';

import '../../../util/text_styles/text_styles.dart';

class UserInfoWidget extends StatelessWidget {
  final String name;

  const UserInfoWidget({Key? key, required this.name})
      : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Column(
      children: [
       Align(alignment:Alignment.centerLeft,child: Text(name, style:
TextStyles.nameTextStyle,)),
        const SizedBox(height: 6)
```

```
      ],
    );
  }
}
```

// chat_room_page.dart – файл що знаходиться за відповідним шляхом pages/chat_room/chat_room_page.dart містить в собі сторінку з чат-кімнатою

```
import
'package:assignment_5/pages/chat_room/widgets/check_chat_room_widget.dart';
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';

import '../../bloc/chat_room_bloc/chat_room_bloc.dart';
import '../../util/colors/colors_style.dart';
import '../../util/decorations/decorations.dart';

class ChatRoomPage extends StatelessWidget {
  const ChatRoomPage({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    final users = (ModalRoute.of(context)?.settings.arguments) as List<dynamic>;
    return SafeArea(
      child: Scaffold(
          appBar: AppBar(
              backgroundColor: ColorStyle.chatAppBarColor,
              title: Row(
                  mainAxisAlignment: MainAxisAlignment.center,
                  children: [
                    Column(
                      mainAxisAlignment: MainAxisAlignment.start,
                      children: [
                        Text(users.first.name),
                        Text(users.first.email, style: const TextStyle(fontSize:
15),),
                      ],
                    ),
                    const SizedBox(width: 30)
                  ],
                ),
              ),
          body: Container(
            decoration: Decorations.backgroundChatRoomDecoration,
            child: BlocProvider<ChatRoomBloc>(
              create: (context) => ChatRoomBloc(),
              child: CheckChatRoomWidget(users: users),
            ),
          )),
    );
  }
}
```

// chat_get_list_widget.dart – файл що знаходиться за відповідним шляхом pages/chat_room/chat_get_list_widget.dart містить в собі віджет що відображає список з чатом користувачів

```
import 'dart:async';

import 'package:assignment_5/util/colors/colors_style.dart';
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';

import '../../../bloc/chat_bloc/chat_bloc.dart';
import '../../../bloc/chat_bloc/chat_event.dart';
import '../../../bloc/chat_bloc/chat_state.dart';
import 'message_widget.dart';

class ChatGetListWidget extends StatelessWidget {
  final id;
  final userId;

  ChatGetListWidget({Key? key, required this.id, required this.userId})
      : super(key: key);

  final ScrollController _scrollController = ScrollController();

  @override
  Widget build(BuildContext context) {
    final ChatBloc chatBloc = ChatBloc();
    final messageController = TextEditingController();
    chatBloc.add(GetChatId(id));
    return BlocBuilder<ChatBloc, ChatState>(bloc: chatBloc,builder: (context,
state) {
      if (state is ChatListState) {
        if (_scrollController.hasClients) {
          scheduleMicrotask(() => _scrollController
              .jumpTo(_scrollController.position.minScrollExtent));
        }
        return Column(
          children: [
            Expanded(
              child: ListView.builder(
                  reverse: true,
                  scrollDirection: Axis.vertical,
                  shrinkWrap: true,
                  itemCount: state.chat.length,
                  controller: _scrollController,
                  itemBuilder: (context, index) {
                    Map<String, dynamic> messageAuth =
                        state.chat[index] as Map<String, dynamic>;
                    return MessageWidget(
                        userId: userId,
                        auth: messageAuth.entries.first.key,
                        message: messageAuth.entries.first.value);
                  }),
            ),
            Container(
              decoration: const BoxDecoration(
                  color: Colors.white,
                  border: Border(
                    top: BorderSide(
                        color: ColorStyle.sendMessageContainerBorderColor,
                        width: 1.5),
                  )),
              child: Row(
                children: [
                  Expanded(
                    child: Padding(
                      padding: const EdgeInsets.only(left: 6.0),
```

```
                    child: TextField(
                      controller: messageController,
                      decoration: const InputDecoration(hintText: 'message'),
                    ),
                  ),
                ),
                IconButton(
                    onPressed: () {
                      chatBloc.add(SendMessage(
                        id,
                        {userId: messageController.text}));
                      messageController.text = "";
                    },
                    icon: const Icon(Icons.send))
              ],
            ),
          ),
        ],
      );
    }
    return const Center(child: CircularProgressIndicator());
  });
 }
}
```

// check_chat_room_widget.dart – файл що знаходиться за відповідним шляхом pages/chat_room/widgets/check_chat_room_widget.dart містить в собі віджет що перевіряє наявність чату та відображає сторінку для його створення якщо його немає

```
import 'package:assignment_5/bloc/chat_room_bloc/chat_room_event.dart';
import 'package:easy_localization/easy_localization.dart';
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';

import '../../../bloc/chat_room_bloc/chat_room_bloc.dart';
import '../../../bloc/chat_room_bloc/chat_room_state.dart';
import '../../../generated/locale_keys.g.dart';
import '../../../util/decorations/decorations.dart';
import '../../../util/text_styles/text_styles.dart';
import 'chat_get_list_widget.dart';

class CheckChatRoomWidget extends StatelessWidget {
  final users;

  const CheckChatRoomWidget({Key? key, required this.users}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    final ChatRoomBloc chatRoomBloc = BlocProvider.of<ChatRoomBloc>(context);
    chatRoomBloc.add(FindChatRoomEvent(users[1], users.first));
    return BlocBuilder<ChatRoomBloc, ChatRoomState>(builder: (context, state) {
      if (state is ChatRoomNewState) {
        return Center(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              FittedBox(
                fit: BoxFit.fitWidth,
                alignment: Alignment.center,
                child: Padding(
                  padding: const EdgeInsets.all(8.0),
                  child: Text(
                    LocaleKeys.You_have_not_started_a_dialog_with_this_user_yet
                        .tr(),
                    style: TextStyles.messageStartDialogTextStyle,
                  ),
                ),
```

```
              ),
              const SizedBox(height: 10),
              Container(
                height: 40,
                decoration: Decorations.buttonDialogDecoration,
                child: ElevatedButton(
                    style: ElevatedButton.styleFrom(
                        primary: Colors.transparent,
                        onSurface: Colors.transparent,
                        shadowColor: Colors.transparent),
                    onPressed: () {
                      chatRoomBloc.add(CreateChatRoomEvent(
                          users[1].uid.toString(), users.first.id.toString()));
                    },
                    child: Text(LocaleKeys.Create_dialog.tr())),
              ),
            ],
          ),
        );
      }
      if (state is ChatRoomIdState) {
        return ChatGetListWidget(
            id: state.chatRoomId, userId: users[1].uid.toString());
      }
      return const Center(child: CircularProgressIndicator());
    });
  }
}
```

// message_widget.dart – файл що знаходиться за відповідним шляхом pages/chat_room/widgets/message_widget.dart містить в собі віджет повідомлення користувача

```
import 'package:assignment_5/util/colors/colors_style.dart';
import 'package:flutter/material.dart';

import '../../../util/text_styles/text_styles.dart';

class MessageWidget extends StatelessWidget {
  final String userId;
  final String auth;
  final String message;

  const MessageWidget(
      {Key? key,
      required this.userId,
      required this.auth,
      required this.message})
      : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Padding(
      padding: const EdgeInsets.all(10.0),
      child: Align(
        alignment: auth == userId ? Alignment.topRight : Alignment.topLeft,
        child: FittedBox(
          clipBehavior: Clip.hardEdge,
          child: Container(
              alignment:
                  auth == userId ? Alignment.topRight : Alignment.topLeft,
              decoration: BoxDecoration(
                  borderRadius: BorderRadius.circular(10),
                  color: auth == userId
                      ? ColorStyle.ourMessageColor
                      : ColorStyle.messageColor),
              child: Padding(
```

```
                padding: const EdgeInsets.all(8.0),
                child: Container(
                    constraints: const BoxConstraints(maxWidth: 250),
                    child: Text(
                        softWrap: true,
                        message,
                        style: TextStyles.messageTextStyle)),
              )),
        ),
      ),
    );
  }
}
```

// firestore.dart – файл що знаходиться за відповідним шляхом networking/firestore.dart містить в собі сервіс для маніпуляцій даними з бази даних firestore

```
import 'dart:async';

import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:easy_localization/easy_localization.dart';

import '../generated/locale_keys.g.dart';

class Firestore {
  final firestore = FirebaseFirestore.instance;

  Stream listenUserbyId(String userId) {
    return firestore
        .collection('users')
        .where('id', isEqualTo: userId)
        .snapshots();
  }

  Stream listenUsers() {
    return firestore.collection('users').snapshots();
  }

  Stream listenChatRoomById(String chatId) {
    return firestore.collection('chatrooms').doc(chatId).snapshots();
  }

  Stream checkLastMessageAndGetListUsers() {
    return firestore.collection('chatrooms').orderBy('lastMessage').snapshots();
  }

  Future<DocumentReference<Map<String, dynamic>>> addChat(
      String idFirstUser, idSecondUser) async {
    return await FirebaseFirestore.instance.collection('chatrooms').add({
      'id_first_user': idFirstUser,
      'id_second_user': idSecondUser,
      'id': '$idFirstUser-$idSecondUser',
      'chat': [
        {'admin': LocaleKeys.Start_the_dialog.tr()}
      ],
      'lastMessage': ''
    });
  }

  void addChatToUser(QuerySnapshot<Map<String, dynamic>> snapshot, int i,
      Map<String, dynamic> chatrooms) async {
    await FirebaseFirestore.instance
        .collection('users')
        .doc(snapshot.docs[i].id)
        .set({'chatrooms': chatrooms}, SetOptions(merge: true));
  }
```

```
  Future<QuerySnapshot<Map<String, dynamic>>> getUserWithId(String key) async {
    return await FirebaseFirestore.instance
        .collection('users')
        .where('id', isEqualTo: key)
        .get();
  }

  Future<QuerySnapshot<Map<String, dynamic>>> getChatroomWithId(
      String id) async {
    return await FirebaseFirestore.instance
        .collection('chatrooms')
        .where('id', isEqualTo: id)
        .get();
  }

  Future<DocumentSnapshot<Map<String, dynamic>>> getChatroomById(
      String id) async {
    return await FirebaseFirestore.instance
        .collection('chatrooms')
        .doc(id)
        .get();
  }

  void addNewMassageToFirestore(
      String id, List<dynamic> chat, String message) async {
    await FirebaseFirestore.instance
        .collection('chatrooms')
        .doc(id)
        .set({'chat': chat, 'lastMessage': message}, SetOptions(merge: true));
  }

  Future<QuerySnapshot<Map<String, dynamic>>> getUsers() async {
    return await FirebaseFirestore.instance.collection('users').get();
  }
}
```

// firebase_auth_client.dart – файл що знаходиться за відповідним шляхом networking/firebase_auth_client.dart містить в собі сервіс для авторизації та реєстрації користувачів

```
import 'package:firebase_auth/firebase_auth.dart';
import 'package:firebase_messaging/firebase_messaging.dart';
import 'package:cloud_firestore/cloud_firestore.dart';

class FirebaseAuthClient{
  final FirebaseAuth auth = FirebaseAuth.instance;

  Future<dynamic>SignIn(String email,String password)
  async {
    String? newToken = await FirebaseMessaging.instance.getToken();
    User? user = FirebaseAuth.instance.currentUser;
    try {
      await FirebaseAuth.instance.signInWithEmailAndPassword(
          email: email, password: password);
      user = FirebaseAuth.instance.currentUser;
      var getIdDoc = await FirebaseFirestore.instance
          .collection('users')
          .where('id', isEqualTo: user?.uid)
          .get();
      FirebaseFirestore.instance
          .collection('users')
          .doc(getIdDoc.docs.first.id)
          .set({'device_id': newToken}, SetOptions(merge: true));
      return (user);
    } catch (e) {
      if (e is FirebaseAuthException) {
        return e;
```

```
      }
    }
  }
  Future<dynamic>SignUp(String email,String password,String name) async {
    try {
      String? newToken = await FirebaseMessaging.instance.getToken();
      User? user = FirebaseAuth.instance.currentUser;
      await FirebaseAuth.instance.createUserWithEmailAndPassword(
          email: email, password: password);
      user = FirebaseAuth.instance.currentUser;
      if(user!=null)
        {
          FirebaseFirestore.instance.collection('users').add({
            'email': user.email,
            'name': name,
            'id': user.uid,
            'chatrooms': <String, String>{},
            'device_id': newToken
          });
        }
      return (user);
    }catch (e) {
      if (e is FirebaseAuthException) {
        return (e);
      }
  }

  }
}
```

// user_model.dart – файл що знаходиться за відповідним шляхом models/user_model.dart містить в собі класи загальної моделі користувача, моделі для реєстрації та моделі авторизації

```
class UserRegister{
  final String name;
  final String email;
  final String password;
  UserRegister(this.name, this.email,this.password);
}

class UserModel{
  final String name;
  final String email;
  final String id;
  UserModel({required this.name, required this.email,required this.id});
  factory UserModel.fromJson(Map<String,dynamic> json)
  {
    return UserModel(name: json['name'],email : json['email'],id: json['id']);
  }
}

class UserLogin{
  final String email;
  final String password;
  UserLogin(this.email,this.password);
}
```

// locale_keys.g.dart – файл що знаходиться за відповідним шляхом generated/locale_keys.g.dart містить в собі згенеровані ключі для локалізації

generate.dart

```
abstract class  LocaleKeys {
  static const Authorization = 'Authorization';
  static const Login = 'Login';
  static const Email = 'Email';
```

```
  static const Password = 'Password';
  static const Sign_In = 'Sign_In';
  static const Sign_Up = 'Sign_Up';
  static const Register = 'Register';
  static const Name = 'Name';
  static const Log_Out = 'Log_Out';
  static const Update = 'Update';
  static const Go_Back = 'Go_Back';
  static const You_did_not_start_dialog ="You_did_not_start_dialog";
  static const Create_dialog = "Create_dialog";
  static const Start_the_dialog = 'Start_the_dialog';
  static const You_have_not_started_a_dialog_with_this_user_yet =
"You_have_not_started_a_dialog_with_this_user_yet";

}
```

// locale_keys.g.dart – файл що знаходиться за відповідним шляхом generated/locale_keys.g.dart містить в собі згенеровану локалізацію

```
import 'dart:ui';

import 'package:easy_localization/easy_localization.dart' show AssetLoader;

class CodegenLoader extends AssetLoader{
  const CodegenLoader();

  @override
  Future<Map<String, dynamic>> load(String fullPath, Locale locale ) {
    return Future.value(mapLocales[locale.toString()]);
  }

  static const Map<String,dynamic> uk = {
  "Authorization": "Авторизація",
    "Login":"Логін",
    "Register":"Реєстрування",
    "Email" : "електрона пошта",
    "Name": "Ім'я",
    "Password": "пароль",
    "Sign_In": "Увійти",
    "Sign_Up": "Зареєструватися",
    "Update" : "Оновити",
    "Log_Out" : "Вийти",
    "Go_Back" : "Повернутись назад",
    "You_did_not_start_dialog":"Ви ще не почали діалог",
    "Create_dialog" : "Розпочати діалог",
    "Start_the_dialog":"Діалог розпочато",
    "You_have_not_started_a_dialog_with_this_user_yet": "Ви ще не розпочинали
діалог з цим користувачем"
      };
static const Map<String,dynamic> en = {
  "Authorization": "Authorization",
  "Login":"Login",
  "Email": "email",
  "Register":"Register",
  "Name": "name",
  "Password": "password",
  "Sign_In": "Sign In",
  "Sign_Up": "Sign Up",
  "Update" : "Update",
  "Log_Out" : "Log out",
  "Go_Back" : "Go back",
  "You_did_not_start_dialog" : "You did not start dialog",
  "Create_dialog" : "Create dialog",
  "Start_the_dialog":"Start the dialog",
  "You_have_not_started_a_dialog_with_this_user_yet" : "You have not started a
dialog with this user yet"
};
static const Map<String, Map<String,dynamic>> mapLocales = {"uk": uk,"en": en};
}
```

```
// chat_bloc.dart – файл що знаходиться за відповідним шляхом
bloc/chat_bloc/chat_bloc.dart містить в собі BLoC чату
import 'package:flutter_bloc/flutter_bloc.dart';
import '../../networking/firestore.dart';
import '../../networking/push_notifications/push_notification.dart';
import 'chat_event.dart';
import 'chat_state.dart';

/// ChatBloc includes operations (sending messages and receiving messages)
/// with a branch of the database to which two users belong
/// according to the ChatroomBloc

class ChatBloc extends Bloc<ChatEvent, ChatState> {
  final Firestore firestore = Firestore();

  @override
  get initialState => ChatEmptyState();

  @override
  Stream<ChatState> mapEventToState(ChatEvent event) async* {
    //receive ID chatroom for storing and sending messages

    if (event is GetChatId) {
      try {
        firestore.listenChatRoomById(event.chatId).listen((snapshot) async {
          add(GetReversedChat(snapshot.data()!['chat']));
        });
      } catch (e) {
        print(e);
      }
    }
    try {
      if (event is SendMessage) {
        _changeChatList(event.idChatRoom, event.messageAndAuth);
        yield ChatListState(_getNewReversedChatList(event.idChatRoom));
      }
    } catch (_) {}
    if (event is GetReversedChat) {
      var reversed = event.chat.reversed.toList();
      yield ChatListState(reversed);
    }
  }

  ///flip the chat for a better view of the message

  _getNewReversedChatList(String id) async {
    var documentUpdate = await firestore.getChatroomById(id);
    var chatUpdate = (documentUpdate.data()?['chat'] as List<dynamic>);
    return chatUpdate.reversed.toList();
  }

  ///open the message array and add a new message at the end
  ///consisting of the sender and the message itself

  void _changeChatList(String id, dynamic message) async {
    String to = '';
    var document = await firestore.getChatroomById(id);
    List<dynamic> chat = document.data()?['chat'];
    chat.add(message);
    var key;
    var messageAuth = message as Map<String, dynamic>;
    for (var item in messageAuth.entries) {
      message = item.value;
      key = item.key;
    }
    await _pushNotification(key, document, to, message);
    firestore.addNewMassageToFirestore(id, chat, message);
  }
```

```
  ///send to devise if the application is minimized
  Future<void> _pushNotification(
      String key, var document, String to, String message) async {
    var user = await firestore.getUserWithId(key);
    if (user.docs.first.get('id') == document.data()?['id_first_user']) {
      to = document.data()?['id_second_user'];
    } else {
      to = document.data()?['id_first_user'];
    }
    var userTo = await firestore.getUserWithId(to);
    await PushNotification.push(
        to: userTo.docs.first.get('device_id'),
        title: "Message from ${user.docs.first.get('name')}",
        body: message);
  }
}
```

// chat_event.dart – файл що знаходиться за відповідним шляхом chat_bloc/chat_event.dart містить в собі події чат кімнати

```
class  ChatEvent{}

class GetChatId extends ChatEvent{
  final String chatId;
  GetChatId(this.chatId);
}

class GetReversedChat extends ChatEvent{
  final List<dynamic> chat;
  GetReversedChat(this.chat);
}

class SendMessage extends ChatEvent {
 final String idChatRoom;
 final Map <String, dynamic> messageAndAuth;
  SendMessage(this.idChatRoom, this.messageAndAuth);
}
```

// chat_state.dart – файл що знаходиться за відповідним шляхом bloc/chat_bloc/chat_state.dart містить в собі стани чат кімнати

```
class ChatState{}

class ChatEmptyState extends ChatState{}
class ChatErrorState extends ChatState{}

class ChatListState extends ChatState{
  final List<dynamic> chat;
  ChatListState(this.chat);
}
```

// chat_room_bloc.dart – файл що знаходиться за відповідним шляхом /bloc/chat_room_bloc/chat_room_bloc.dart містить в собі логіку BLoC чат-кімнати

```
import 'package:firebase_messenger/bloc/chat_room_bloc/chat_room_event.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import '../../networking/firestore.dart';
import 'chat_room_state.dart';

///ChatRoomBloc check availability chat room and creates a new room for two
users
///if the room has not yet been created
class ChatRoomBloc extends Bloc<ChatRoomEvent, ChatRoomState> {
  final Firestore firestore = Firestore();

  @override
  get initialState => ChatRoomEmptyState();
```

```
  @override
  Stream<ChatRoomState> mapEventToState(ChatRoomEvent event) async* {
    bool newRoom = true;
    String id = '';
    if (event is CreateChatRoomEvent) {
      id = await _addChat(event.OurId, event.IdSecondUser);
      firestore.listenUsers().listen((snapshot) {
        for (int i = snapshot.docs.length - 1; i >= 0; i--) {
          if (snapshot.docs[i].get('id') == event.OurId) {
            _addChatToUser(id, event.IdSecondUser, snapshot, i);
          }
          if (snapshot.docs[i].get('id') == event.IdSecondUser) {
            _addChatToUser(id, event.OurId, snapshot, i);
          }
        }
      });
      yield ChatRoomIdState(id);
    } else if (event is FindChatRoomEvent) {
      _findUserChatRooms(event.ourUser.uid, event.secondUser.id);
    } else if (event is GetChatRoomEvent && event.chatroom.isEmpty) {
      yield ChatRoomNewState();
    } else if (event is GetChatRoomEvent) {
      for (var item in event.chatroom.entries) {
        if (item.value == event.secondUserId) {
          newRoom = false;
          yield ChatRoomIdState(item.key);
        }
      }
      if (newRoom) {
        yield ChatRoomNewState();
      }
    }
  }

  ///check the visibility of the chat room
  void _findUserChatRooms(String loginedUserId, String secondUserId) {
    firestore.listenUserbyId(loginedUserId).listen((snapshot) {
      add(GetChatRoomEvent(snapshot.docs.first.get('chatrooms'), secondUserId));
    });
  }

  ///add created chat to the user's chat list
  void _addChatToUser(String id, String secondUser,
      QuerySnapshot<Map<String, dynamic>> snapshot, int i) {
    Map<String, dynamic> chatrooms =
        snapshot.docs[i].get('chatrooms') as Map<String, dynamic>;
    chatrooms.addAll({id: secondUser});
    firestore.addChatToUser(snapshot, i, chatrooms);
  }

  ///add new chat to firestore data base
  Future<String> _addChat(String ourId, String idSecondUser) async {
    String id = '';
    await firestore
        .addChat(ourId, idSecondUser)
        .then((value) => ((id = value.id)));
    return id;
  }
}
```

// chat_room_event.dart – файл що знаходиться за відповідним шляхом bloc/chat_room_bloc/chat_room_event.dart містить в собі події чат-кімнати

```
import '../user_bloc/user_event.dart';

class ChatRoomEvent{}

class FindChatRoomEvent extends ChatRoomEvent{
  final ourUser;
```

```
  final secondUser;
  FindChatRoomEvent(this.ourUser,this.secondUser);
}

class GetChatRoomEvent extends ChatRoomEvent{
  final Map<String,dynamic> chatroom;
  final String secondUserId;
  GetChatRoomEvent(this.chatroom,this.secondUserId);
}

class CreateChatRoomEvent extends ChatRoomEvent{
  final String OurId;
  final String IdSecondUser;
  CreateChatRoomEvent(this.OurId, this.IdSecondUser);
}
```

// chat_room_state.dart – файл що знаходиться за відповідним шляхом bloc/chat_room_bloc/chat_room_state.dart містить в собі події чат-кімнати

```
class ChatRoomState{}

class ChatRoomEmptyState extends ChatRoomState{}
class ChatRoomErrorState extends ChatRoomState{}
class ChatRoomNewState extends ChatRoomState{}
class ChatRoomCreatedState extends ChatRoomState{}

class ChatRoomIdState extends ChatRoomState{
  final String chatRoomId;
  ChatRoomIdState(this.chatRoomId);
}
```

// login_bloc.dart – файл що знаходиться за відповідним шляхом bloc/login_bloc/login_bloc.dart містить в собі BLoC авторизації

```
import 'package:firebase_messenger/bloc/login_bloc/login_event.dart';
import 'package:firebase_messenger/bloc/login_bloc/login_state.dart';
import 'package:firebase_messenger/networking/firebase_auth_client.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import '../../models/user_model.dart';

/// Authorizes the user and checks the information entered in the fields
class LoginBloc extends Bloc<LoginEvent, LoginState> {

  @override
  get initialState => LoginEmptyState();

  @override
  Stream<LoginState> mapEventToState(LoginEvent event) async* {
    final FirebaseAuthClient authClient = FirebaseAuthClient();
    if (event is SetInitialEvent) {
      yield LoginEmptyState();
    }
    if (event is GetUserDataEvent) {
      dynamic authStatus =
          await authClient.SignIn(event.userLogin.email,
event.userLogin.password);
      if (authStatus is User) {
        yield UserLoginedState(authStatus);
      } else {
        yield LoginErrorState(_checkLoginError(event.userLogin, authStatus));
      }
    }
  }

  /// checks the information entered in the fields
  String _checkLoginError(UserAuth user, FirebaseAuthException e) {
    if (user.email.isEmpty || user.password.isEmpty) {
```

```
      return ('Fill in all the fields');
    }
    if (user.password.length < 6) {
      return ('password must be at least 6 characters');
    }
    if (!user.email.contains('.') || !user.email.contains('@')) {
      return ('invalid email input form , please check availability . or @');
    }
    return (e.message.toString());
  }
}

/// checks the information entered in the fields
  String _checkLoginError(UserAuth user, FirebaseAuthException e) {
    if (user.email.isEmpty || user.password.isEmpty) {
      return ('Fill in all the fields');
    }
    if (user.password.length < 6) {
      return ('password must be at least 6 characters');
    }
    if (!user.email.contains('.') || !user.email.contains('@')) {
      return ('invalid email input form , please check availability . or @');
    }
    return (e.message.toString());
  }
}
```

// login_event.dart – файл що знаходиться за відповідним шляхом bloc/login_bloc/login_event.dart містить в собі події авторизації

```
import 'package:firebase_messenger/models/user_model.dart';
import 'package:firebase_auth/firebase_auth.dart';

import '../../models/user_model.dart';

class LoginEvent{}


class SetInitialEvent extends LoginEvent{}
class GetUserDataEvent extends LoginEvent{
  final UserAuth userLogin;
  GetUserDataEvent(this.userLogin);
}
```

// login_state.dart – файл що знаходиться за відповідним шляхом bloc/login_bloc/login_state.dart містить в собі стани авторизації

```
import 'package:firebase_auth/firebase_auth.dart';

class LoginState{}

class LoginEmptyState extends LoginState{}
class LoginErrorState extends LoginState{
  final String message;
  LoginErrorState(this.message);
}

class UserLoginedState extends LoginState{
  final User user;
  UserLoginedState(this.user);
}
```

// register_bloc.dart – файл що знаходиться за відповідним шляхом bloc/register_bloc/register_bloc.dart містить в собі BLoC реєстрації

```
import 'package:firebase_messenger/bloc/register_bloc/register_event.dart';
import 'package:firebase_messenger/bloc/register_bloc/register_state.dart';
import 'package:firebase_auth/firebase_auth.dart';
```

```
import 'package:flutter_bloc/flutter_bloc.dart';

import '../../models/user_model.dart';
import '../../networking/firebase_auth_client.dart';

///Register the user and checks the information entered in the fields
class RegisterBloc extends Bloc<RegisterEvent, RegisterState> {
  @override
  get initialState => RegisterEmptyState();

  @override
  Stream<RegisterState> mapEventToState(RegisterEvent event) async* {
    if (event is SetInitialRegisterEvent) {
      yield RegisterEmptyState();
    }
    if (event is GetUserDataEvent) {
      final FirebaseAuthClient authClient = FirebaseAuthClient();
      dynamic authStatus = await authClient.SignUp(event.userRegister.email,
          event.userRegister.password, event.userRegister.name ?? "");
      if (authStatus is User) {
        yield UserRegisteredState(authStatus);
      } else {
        yield RegisterErrorState(_checkRegisterError(event.userRegister,
authStatus));
      }
    }
  }

  String _checkRegisterError(
      UserAuth user, FirebaseAuthException exception) {
    if (user.password.length < 6) {
      return ('password must be at least 6 characters');
    }
    if (user.email.isEmpty || user.password.isEmpty || user.name=="") {
      return ('Fill in all the fields');
    }
    if (!user.email.contains('.') || !user.email.contains('@')) {
      return ('invalid email input form , please check availability . or @');
    }
    return (exception.message.toString());
  }
}
```

// register_event.dart – файл що знаходиться за відповідним шляхом bloc/register_bloc/register_event.dart містить в собі події реєстрації

```
import 'package:firebase_messenger/models/user_model.dart';

class RegisterEvent{}


class SetInitialRegisterEvent extends RegisterEvent{}
class GetUserDataEvent extends RegisterEvent {
  final UserAuth userRegister;
  GetUserDataEvent(this.userRegister);
}
```

// register_state.dart – файл що знаходиться за відповідним шляхом bloc/register_bloc/register_state.dart містить в собі стани реєстрації

```
import 'package:firebase_auth/firebase_auth.dart';

class RegisterState{}

class RegisterEmptyState extends RegisterState{}
class RegisterErrorState extends RegisterState{
 final String message;
```

```
 RegisterErrorState(this.message);
}

class UserRegisteredState extends RegisterState{
  final User user;
  UserRegisteredState(this.user);
}
```

// users_bloc.dart – файл що знаходиться за відповідним шляхом bloc/user_bloc/users_bloc.dart містить в собі BLoC користувачів

```
import 'dart:async';

import 'package:firebase_messenger/bloc/user_bloc/user_event.dart';
import 'package:firebase_messenger/bloc/user_bloc/user_state.dart';
import 'package:firebase_messenger/networking/firestore.dart';
import 'package:easy_localization/easy_localization.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import '../../generated/locale_keys.g.dart';
import '../../models/user_model.dart';

///UserBloc is responsible for loading and displaying currently registered users
class UserBloc extends Bloc<UserEvent, UserState> {
  final Firestore firestore = Firestore();
  String id = '';
  bool timeUpdate = true;

  @override
  get initialState => UserLoadingState();

  @override
  Stream<UserState> mapEventToState(UserEvent event) async* {
    if (event is UserLogOutEvent) {
      await FirebaseAuth.instance.signOut();
      yield UserLogOutState();
    }
    if (event is UserLoadingEvent) {
      firestore.checkLastMessageAndGetListUsers().listen((snapshot) async {
        add(UserLoadedEvent((await firestore.getUsers()).docs));});
      id = event.userID;
    }
    if (event is UserLoadedEvent) {
      List<dynamic> usersAndLastMessage = <dynamic>[];
      try {
        for (int i = 0; i < event.docs.length; i++) {
          String lastMessage = LocaleKeys.You_did_not_start_dialog.tr();
          UserModel userInList = UserModel.fromJson(event.docs[i].data());
          var getLastMessage = await firestore.getChatroomWithId('$id-
${userInList.id}');
          if (getLastMessage.docs.isNotEmpty) {
            lastMessage = getLastMessage.docs.first.get('lastMessage');
          } else {
            getLastMessage = await
firestore.getChatroomWithId('${userInList.id}-$id');
            if (getLastMessage.docs.isNotEmpty) {
              lastMessage = getLastMessage.docs.first.get('lastMessage');
            }
          }
          usersAndLastMessage.add([userInList, lastMessage]);
        }
        yield UserLoadedState(usersAndLastMessage);
      } catch (_) {}
    }
  }
}
```

// user_event.dart – файл що знаходиться за відповідним шляхом bloc/user_bloc/user_event.dart містить в собі події користувачів

```
class  UserEvent{}

class UserLoadingEvent extends UserEvent{
  final String userID;
  UserLoadingEvent(this.userID);
}

class UserLoadedEvent extends UserEvent{
  final docs;
  UserLoadedEvent(this.docs);
}
class UserLogOutEvent extends UserEvent{}
```

// user_state.dart – файл що знаходиться за відповідним шляхом bloc/user_bloc/user_state.dart містить в собі стани користувачів

```
class  UserState{}

class UserLoadingState extends UserState{
}
class UserEmptyState extends UserState{}
class UserErrorState extends UserState{}
class ListenLastMessage extends UserState{}
class UserLogOutState extends UserState{}

class UserLoadedState extends UserState{
  List<dynamic> loadedUsers;
  UserLoadedState(this.loadedUsers);
}
```

// user_provider.dart – файл що знаходиться за відповідним шляхом bloc/user_bloc/user_provider.dart містить в собі потік користувачів

```
import 'package:cloud_firestore/cloud_firestore.dart';
class UserProvider {
  static dynamic getDataBase() {
    dynamic usersData;
    FirebaseFirestore.instance.collection('users').snapshots().listen((
        snapshot) {
          usersData=snapshot.docs;
    });
    return usersData;
  }
}
```

// colors_style.dart – файл що знаходиться за відповідним шляхом util/colors/colors_style.dart містить в собі палітру інтерфейсу користувача

```
import 'package:flutter/material.dart';

abstract class ColorStyle{
  static const sendMessageContainerBorderColor=Colors.grey;
  static const ourMessageColor=Colors.blue;
  static const messageColor=Colors.indigo;
  static const EvenBorderColor = Colors.white;
  static const NotEvenBorderColor = Colors.white;
  static final EvenUserColor = Colors.amber[50];
  static const NotEvenUserColor = Colors.white;
  static final EvenAvatarColor = Colors.amber[700];
  static final NotEvenAvatarColor = Colors.blue[300];
  static const LogOutButtonColor = Colors.white;
  static const indicatorColor = Colors.white;
  static const chatAppBarColor = Colors.indigoAccent;

}
```

// text_styles.dart.dart – файл що знаходиться за відповідним шляхом util/text_styles/text_styles.dart містить в собі стилі тексту

```
import 'package:flutter/material.dart';

abstract class TextStyles {
  static const loginTextStyle = TextStyle(fontSize: 28, color:
Colors.black,fontWeight: FontWeight.w500);
  static const firstLetterOfNameTextStyle =
      TextStyle(color: Colors.white, fontWeight: FontWeight.w500, fontSize: 18);
  static const nameTextStyle = TextStyle(fontWeight: FontWeight.w500);
  static const emailTextStyle = TextStyle(fontStyle: FontStyle.italic);
  static const messageStartDialogTextStyle =
      TextStyle(fontSize: 18, color: Colors.black54);
  static const messageTextStyle=TextStyle(
      color: Colors.white,
      fontSize: 15);
  static const registerTextStyle = TextStyle(fontSize: 29, color:
Colors.black,fontWeight: FontWeight.w500);
}
```

// decorations.dart.dart – файл що знаходиться за відповідним шляхом util/text_styles/text_styles.dart містить в собі градієнти та інші компоненти інтерфейсу

```
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';

abstract class Decorations{
  static final buttonDecoration = BoxDecoration(
  borderRadius: BorderRadius.circular(6.0),
  gradient: const LinearGradient(
  begin: Alignment(-0.95, 0.0),
  end: Alignment(1.0, 0.0),
  colors: [Color(0xff667eea), Color(0xff64b6ff)],
  stops: [0.0, 1.0],
  ));
  static final buttonDialogDecoration = BoxDecoration(
      borderRadius: BorderRadius.circular(6.0),
      gradient: const LinearGradient(
        begin: Alignment(-0.95, 0.0),
        end: Alignment(1.0, 0.0),
        colors: [Colors.indigoAccent, Colors.deepPurpleAccent],
        stops: [0.0, 1.0],
      ));
  static const backgroundChatRoomDecoration =  BoxDecoration(
      gradient: LinearGradient(
          begin: Alignment.topLeft,
          end: Alignment.bottomRight,
          colors: [Colors.indigoAccent, Colors.white,Colors.white])
  );
  static const backgroundMainDecoration =  BoxDecoration(
      gradient: LinearGradient(
          begin: Alignment.topLeft,
          end: Alignment.bottomRight,
          colors: [Colors.blue,Colors.white])
  );
}
```

// en.json – файл що знаходиться за відповідним шляхом assets/translation/en.json містить в собі Англійську локалізацію

```
{
  "Authorization" : "Authorization",
  "Login" : "Login",
  "Email": "email",
  "Password": "password",
  "Sign_In": "Sign In",
  "Sign_Up": "Sign Up",
  "Register":"Register",
  "Name": "name",
  "Log_Out": "Log out",
  "Update": "Update",
  "Go_Back" : "Go back",
  "Start_the_dialog":"Start the dialog",
  "You_did_not_start_dialog" : "You did not start dialog",
  "Create_dialog" : "Create dialog",
  "You_have_not_started_a_dialog_with_this_user_yet" : "You have not started a
dialog with this user yet"
}
```

// uk.json – файл що знаходиться за відповідним шляхом assets/translation/uk.json містить в собі Англійську локалізацію

```
{
  "Authorization": "Авторизація",
  "Login": "Логін",
  "Email": "електрона пошта",
  "Password": "пароль",
  "Sign_In": "Увійти",
  "Sign_Up": "Зареєструватися",
  "Name": "Ім'я",
  "Register": "Реєструватися",
  "Update": "Оновити",
  "Log_Out": "Вийти",
  "Go_Back": "Повернутись назад",
  "Start_the_dialog":"Діалог розпочато",
  "You_did_not_start_dialog": "Ви ще не почали діалог",
  "Create_dialog" : "Розпочати діалог",
  "You_have_not_started_a_dialog_with_this_user_yet": "Ви ще не розпочинали
діалог з цим користувачем"
}
```

// push_notification.dart – файл що знаходиться за відповідним шляхом networking/push_notification/push_notification.dart містить в собі клас для відправки повідомлень на пристрій

```
import 'dart:core';
import 'package:http/http.dart' as http;

import '../../models/push_notification_model.dart';

///Push Notification responsible for sending messages to the phone
///when the application is minimized or disabled
class PushNotification{
  static const String pushNotificationServerKey ="AAAAzmp6sfg:APA91bHYb6nFgJApW"
      "3H8iDiwzEfsciudHS-KGwraAqqle5E0547oUCfZAPWm2rFCnGbd-Eo1TTy3lRWYpQAEe3fgZ"
      "27xTgMNlzpWmpbm4vaTODSoMY1NAbPGVrkxQM19YTDkcrV_bTy-";
  static Future<void> push({required String to, required String title, required
String body}) async
  {
    try{
      http.Response response = await http.post(
          Uri.parse('https://fcm.googleapis.com/fcm/send'),
          headers: <String, String>{
            'Content-Type': 'application/json',
            'Authorization': 'key=$pushNotificationServerKey',
          },
```

```
          body: jsonEncode(PushNotificationModel(
              notification: Notification(title: title, body: body), to: to)));
      if (response.statusCode == 200) {
      } else {
        throw Exception();
      }
    }catch(e){
      throw Exception(e);
    }
  }
}
```

// push_notification_model.dart – файл що знаходиться за відповідним шляхом models/push_notification_model/push_notification_model.dart містить в собі модель повідомлення для відправки на пристрій

```
import 'dart:convert';
import 'dart:core';
import 'package:http/http.dart' as http;

import '../../models/push_notification_model.dart';

///Push Notification responsible for sending messages to the phone
///when the application is minimized or disabled
class PushNotification{
  static const String pushNotificationServerKey ="AAAAzmp6sfg:APA91bHYb6nFgJApW"
      "3H8iDiwzEfsciudHS-KGwraAqqle5E0547oUCfZAPWm2rFCnGbd-Eo1TTy3lRWYpQAEe3fgZ"
      "27xTgMNlzpWmpbm4vaTODSoMY1NAbPGVrkxQM19YTDkcrV_bTy-";
  static Future<void> push({required String to, required String title, required
String body}) async
  {
    try{
      http.Response response = await http.post(
          Uri.parse('https://fcm.googleapis.com/fcm/send'),
          headers: <String, String>{
            'Content-Type': 'application/json',
            'Authorization': 'key=$pushNotificationServerKey',
          },
          body: jsonEncode(PushNotificationModel(
              notification: Notification(title: title, body: body), to: to)));
      if (response.statusCode == 200) {
      } else {
        throw Exception();
      }
    }catch(e){
      throw Exception(e);
    }
  }
}
```

// push_notification_model.g.dart – файл що знаходиться за відповідним шляхом models/push_notification_model/push_notification_model.g.dart містить в собі згенеровану модель повідомлення для відправки на пристрій

```
import 'package:firebase_messenger/models/push_notification_model.g.dart';
import 'package:json_annotation/json_annotation.dart';

///Model notification consist of title(from user) and body(message)
class PushNotificationModel {
  Notification notification;
  String to;

  PushNotificationModel({required this.notification, required this.to});

  factory PushNotificationModel.fromJson(Map<String, dynamic> json) =>
      $PushNotificationModelFromJson(json);

  Map<String, dynamic> toJson() => $PushNotificationModelToJson(this);
```

```
}

@JsonSerializable()
class Notification {
  String title;
  String body;

  Notification({required this.title, required this.body});

  factory Notification.fromJson(Map<String, dynamic> json) =>
      $NotificationFromJson(json);

  Map<String, dynamic> toJson() => $NotificationToJson(this);
}
```

// pubspec.yaml.json – файл що містить в собі усі залежності та пакети що використовує додаток

```
name: firebase_messenger
description: A new Flutter project.

publish_to: 'none' # Remove this line if you wish to publish to pub.dev

version: 1.0.0+1

environment:
  sdk: ">=2.17.1 <3.0.0"

dependencies:
  flutter:
    sdk: flutter

  cupertino_icons: ^1.0.2
  firebase_core: ^1.19.1
  firebase_auth: ^3.4.1
  flutter_bloc: ^4.0.1
  easy_localization: ^3.0.1
  firebase_messaging: ^11.2.11
  json_annotation: ^4.5.0
  json_serializable: ^6.2.0
  http: ^0.13.4

  flutter_local_notifications:

dev_dependencies:
  cloud_firestore: 3.1.18
  flutter_test:
    sdk: flutter


  flutter_lints: ^2.0.0

flutter:

  uses-material-design: true
  assets:
  - assets/translation/
```