Галицький фаховий коледж імені В’ячеслава Чорновола
відділення комп’ютерних технологій
циклова комісія інформатики та комп’ютерних дисциплін

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач відділення
комп’ютерних технологій
Наталія СТЕФУРАК /_____/
(підпис)
«___» _____ 2024 р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи
освітньо-професійного ступеня «фаховий молодший бакалавр»
зі спеціальності 122 «Комп’ютерні науки»
на тему: «Соціальна мережа для локального бізнесу в сфері SMM»

Студентка групи КН-41	Яна КОВАЛЬЧУК	_____ (підпис)
Керівник роботи	Ольга ПОСВ’ЯТОВСЬКА	_____ (підпис)
Консультанти:		
з техніко-економічного обґрунтування	Любов МЕЛЕНЧУК	_____ (підпис)
нормоконтролер	Надія ГАВРИШКІВ	_____ (підпис)

Тернопіль – 2024

Галицький фаховий коледж імені В’ячеслава Чорновола

відділення комп’ютерних технологій

циклова комісія інформатики та комп’ютерних дисциплін

ДОПУСТИТИ ДО ЗАХИСТУ

Завідувач відділення

комп’ютерних технологій

Наталія СТЕФУРАК /_____/

(підпис)

«___» _____ 2023 р.

ЗАВДАННЯ

на кваліфікаційну роботу на здобуття освітньо-професійного ступеня «фаховий молодший бакалавр»

студентці _____

(прізвище, ім’я, по-батькові студентки)

1.Тема проєкту _____

_____

затверджена наказом по коледжу від “___”_____202_ р., №_____

2. Термін здачі студентом завершеної роботи “___” _____202_ р.

3. Вихідні дані до роботи _____

_____

_____

4. Перелік питань, які повинні бути розроблені:

а) основна частина_____

_____

б) техніко-економічне обґрунтування_____

_____

5. Перелік графічного матеріалу _____

_____

_____

6. Консультанти роботи : _____

Розділ	Консультанти	Підпис, дата	
		Завдання видано	Завдання прийнято
з техніко-економічного обґрунтування	_____ (вчена ступень, звання П.І.Б. _____ консультанта)		

КАЛЕНДАНРИЙ ПЛАН

Виконання кваліфікаційної роботи

№ п/п	Найменування етапу	Терміни	
		початку	завершення

7. Дата видачі завдання “___” _____ 2023 р. Керівник _____/

Завдання прийняла до виконання _____/ /

Реферат

Кваліфікаційна робота. Соціальна мережа для локального бізнесу в сфері SMM. Ковальчук Яна Михайлівна. Галицький фаховий коледж імені В’ячеслава Чорновола, відділення комп’ютерних технологій. 134 сторінки, рисунків – 30, додатків – 11.

Об’єкт дослідження – соціальні мережі та їх функціонал.

Метою роботи є створення соціальної мережі для локального бізнесу в сфері SMM такими засобами: кросплатформна розробка на фреймворку Flutter з використанням мови програмування Dart, середовищем розробки Visual Studio Code та нереляційною базою даних Firebase.

Соціальна мережа для локального бізнесу в сфері SMM повинна забезпечити створення акаунтів користувачів системи, редагування профілю, додавання контенту, який притаманний соціальним мережам, перегляд контенту інших користувачів, знаходження користувачів, які знаходяться в системі, ведення комунікації між користувачами. Крім того, необхідно спроєктувати та розробити зручний інтерфейс із врахуванням вимог для мобільних додатків на базі операційних систем Android та iOS.

Результатом розробки стала соціальна мережа для локального бізнесу в сфері SMM, яка готова до використання.

СОЦІАЛЬНА МЕРЕЖА, SMM, API, BLoC, Android, iOS, BaaS, Firebase, JSON.

Abstract

Qualification work. Social network for local business in the field of SMM. Kovalchuk Yana Mykhailivna. Galician Vocational College named after Vyacheslav Chornovil, Department of Computer Technologies. 136 pages, 30 figures, 11 appendices.

The object of research is social networks and their functionality.

The aim of the study is to create a social network for local business in the field of SMM by the following means: cross-platform development on the Flutter framework using the Dart programming language, Visual Studio Code development environment and Firebase non-relational database.

A social network for local businesses in the SMM sphere should provide for the creation of user accounts, profile editing, adding content that is typical for social networks, viewing content from other users, finding users who are in the system, and communicating between users. In addition, a user-friendly interface had to be designed and developed, taking into account the requirements for mobile applications based on Android and iOS operating systems.

The result is a social network for local SMM businesses that is ready for use.

SOCIAL NETWORK, SMM, API, BLoC, Android, iOS, BaaS, Firebase, JSON.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

SMM – Social Media Marketing

UML – Unified Modeling Language

API – Application Programming Interface

BLoC – Business Logic Component

Android – Операційна система Android

iOS – Операційна система iOS

BaaS – Backend as a Service

Firebase – Платформа для розробки мобільних і вебдодатків

JSON – JavaScript Object Notation

AVD – Android Virtual Device

ЄСВ – Єдиний соціальний внесок

ПДФО – Податок на доходи фізичних осіб

ВСТУП

У сучасному висококонкурентному бізнес-середовищі впровадження інноваційних технологій є вирішальним стратегічним кроком, який гарантує успіх. У цьому контексті соціальні мережі відіграють вирішальну роль у розвитку та успіху бізнесу.

Важливість цієї теми полягає в стрімкому розвитку інтернет-технологій і постійному зростанні популярності соціальних мереж. У сфері локального бізнесу, що спеціалізується на Social Media Marketing, створення спеціальної соціальної мережі має величезний потенціал для просування зростання та маркетингу послуг.

Мета кваліфікаційної роботи – створити соціальну мережу, яку можна використовувати на різних пристроях, спеціально розроблену для допомоги малому та середньому бізнесу у сфері маркетингу та реклами. Ця мережа запропонує легкі можливості спілкування та реклами.

До основних завдань цієї кваліфікаційної роботи відновиться наступне:

– вивчення поточних моделей використання платформ соціальних мереж в комерційних цілях;

– розробка архітектури та функціоналу соціальної мережі, що враховує особливості локального бізнесу;

– впровадження та тестування кросплатформенної програми;

– оцінка ефективності та придатності розробленої платформи для місцевого бізнесу.

Об’єктом дослідження є комунікаційні процеси локального бізнесу через соціальну мережу. Предметом дослідження є створення та впровадження кросплатформної соціальної мережі для локального бізнесу у сфері SMM.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАНЬ

1.1 Дослідження предметної області

Соціальні мережі за останні роки стали невіддільною частиною життя людей. Їхня популярність постійно зростає, що робить їх потужним інструментом для маркетингу. SMM (Social Media Marketing) – це форма інтернет-маркетингу, яка використовує соціальні мережі як маркетинговий інструмент [1].

Послуги із SMM можу бути використані для наступних цілей:

– підвищення впізнаваності особистого бренду. Це охоплює регулярні публікації цікавого та якісного контенту, щоб охопити ширшу аудиторію користувачів соціальних мереж;

– знаходження потенційних клієнтів завдяки наданню можливості створення спеціальних сторінок для збору контактної інформації потенційних клієнтів за допомогою лід-форм;

– підвищення продажів. Можливість публікування фото та відео продуктів, для показу їх у дії та наголошені на тому, як вони зможуть розв'язувати проблеми клієнтів;

– підтримка зв'язків з клієнтами. Соціальні мережі мають зручні інструменти для цілодобової комунікації із клієнтами. Це дозволяє дізнаватися думку аудиторії про бізнес-продукти чи послуги.

Завдяки цьому, SMM послуги мають високий попит та ведення бізнесу в цій сфері є досить прибутковим.

1.2 Обґрунтування доцільності створення соціальної мережі для локального бізнесу в сфері SMM

SMM є одним із найперспективніших напрямів маркетингу у XXI столітті. За даними Statista, станом на січень 2024 року в світі налічувалося 5,35 мільярда користувачів Інтернету, що становило

66,2 відсотка світового населення. З них 5,04 мільярда, або 62,3 відсотка населення світу, були користувачами соціальних мереж [2]. Цей показник візуально можна побачити на рисунку 1.1, що свідчить про значне поширення та доступність соціальних мереж, роблячи їх потужним інструментом для SMM.

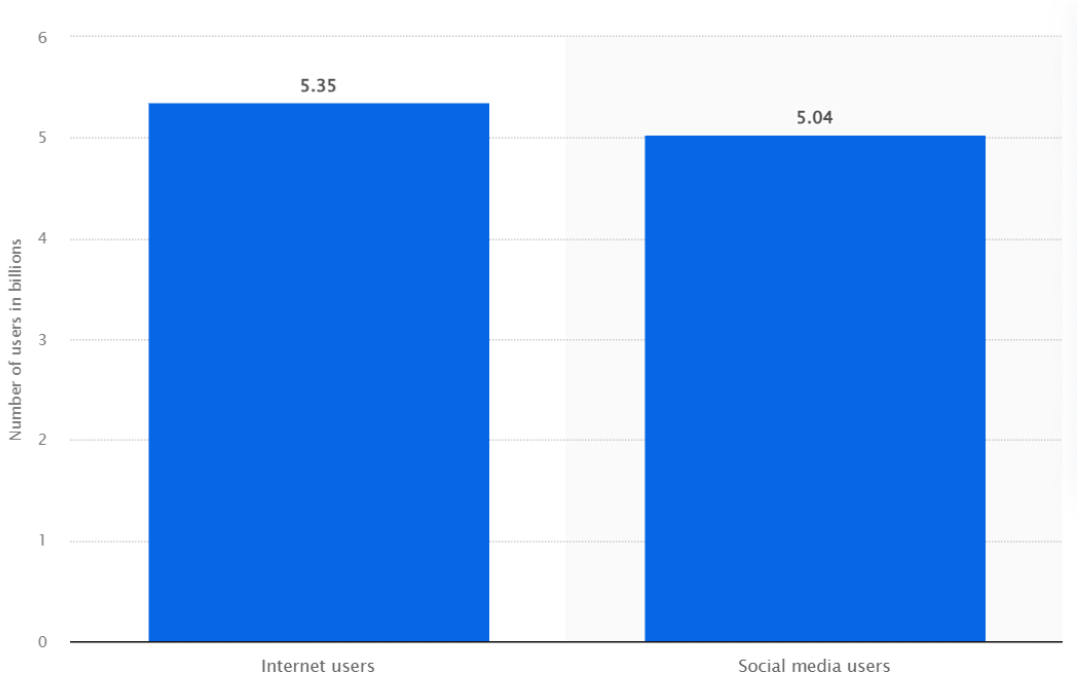

Рисунок 1.1 – Кількість користувачів Інтернету та соціальних мереж у всьому світі станом на січень 2024 року

Кажучи про бізнес, очікується, що глобальні продажі електронної комерції у 2024 році становитимуть 6,3 трильйона доларів США [3]. Прогнозується, що ця цифра продовжить зростати протягом наступних кількох років, доводячи, що ведення бізнесу у соціальних мереж стає все більш прибутковим варіантом для компаній, рисунок 1.2.

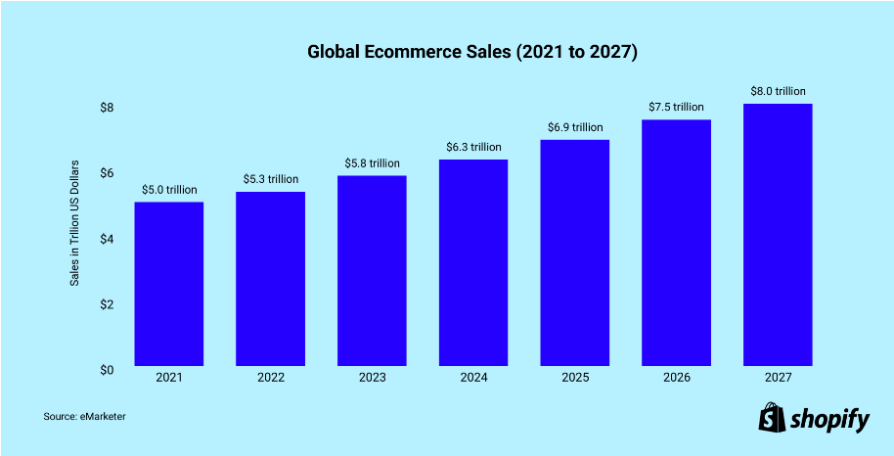

Рисунок 1.2 – Прогноз глобальних продажів електронної комерції (2021–2027)

Загалом, можна стверджувати, що збільшення кількості користувачів Інтернету і соціальних мереж, а також зростання онлайн-покупок, роблять послуги із SMM важливим інструментом маркетингу для бізнесу та особистих брендів. Створення власної соціальної мережі для локального бізнесу в сфері SMM дозволяє бізнесу створити унікальний імідж, висвітлити свої цінності та виділити свою ексклюзивність, що сприяє позитивному сприйняттю бренду та успішному його позиціювання в конкурентному середовищі.

1.3 Аналіз наявних рішень

Для розробки застосунку для локального бізнесу в сфері SMM важливо проаналізувати декілька різних популярних платформ, оскільки їх аналіз дозволить чітко сформувати технічне завдання.

Для аналізу були підібрані найпопулярніші соціальні мережі, такі як:

– Instagram.
– TikTok.
– UkrOpen.

Однією із найрейтинговіших соціальних мереж є Instagram, інтерфейс якої можна побачити на рисунку 1.3.

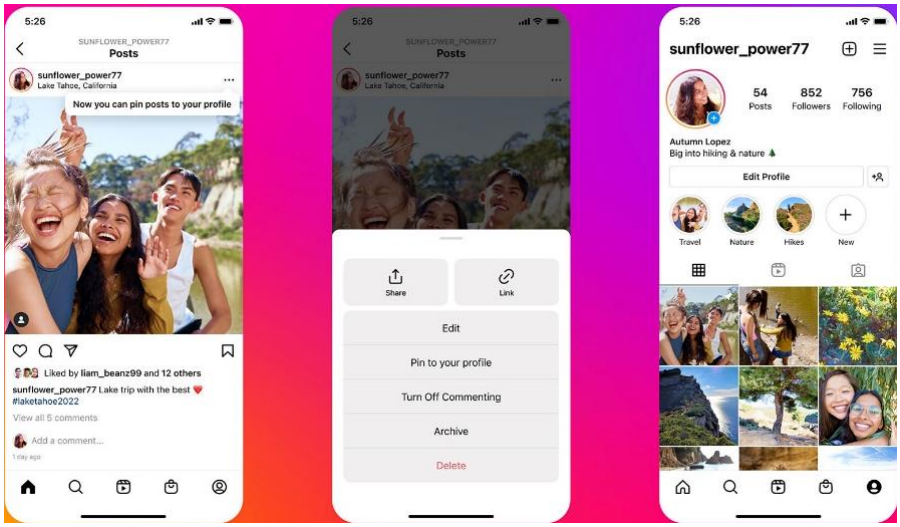

Рисунок 1.3 – Інтерфейс соціальної мережі Instagram

Це соціальна мережа привертає увагу доволі активної аудиторії серед молоді, та підходить для брендів з сильною візуальною ідентичністю, але в свою чергу, має обмежену аналітику та інструменти звітування для бізнесу.

Детальнішу характеристику наведено в таблиці 1.1.

Таблиця 1.1 – Переваги та недоліки Instagram для використання в бізнесі

Переваги	Недоліки
Наявність веб та мобільної версії.	Обмежена функціональність для підприємств.
Візуальна платформа, де основним контентом виступають фото та відео.	Обмежена функціональність чатів.

Продовження таблиці 1.1

Великий потенціал для взаємодії з аудиторією через коментарі та приватні повідомлення.	Обмежені інструменти аналітики.
Зручний інтерфейс.	Алгоритмічні зміни та обмеження розповсюдження контенту.

Наступною варто проаналізувати соціальну мережу TikTok, інтерфейс якої можна побачити на рисунку 1.4.

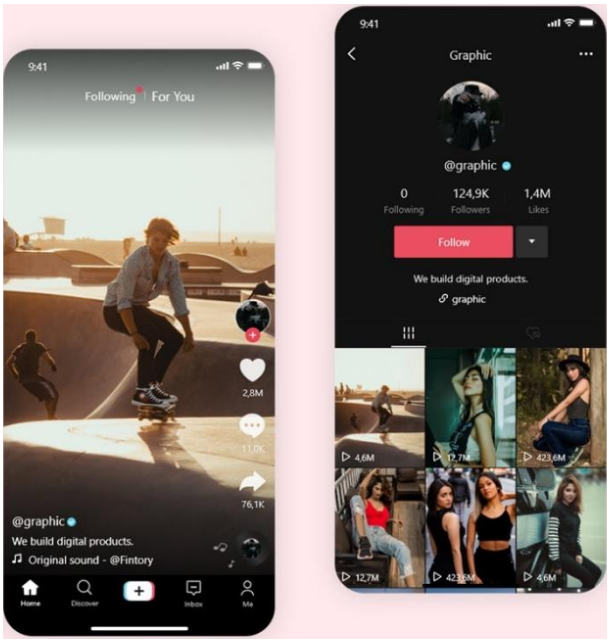

Рисунок 1.4 – Інтерфейс соціальної мережі TikTok

Це соціальна мережа досягла значного розквіту у період пандемії Covid19 та зазнає швидкого зростання аудиторії та має великий потенціал вірусної реклами, але найбільшим недоліком є обмежена аналітика. Повну характеристику можна розглянути в таблиці 1.2.

Таблиця 1.2 – Переваги та недоліки TikTok для використання в бізнесі

Переваги	Недоліки
Швидке зростання аудиторії серед молоді.	Молодша аудиторія порівняно з Instagram.
Наявність веб, десктопної та мобільної версії.	Обмежена аналітика та інструменти для бізнесу.
Велика кількість творчих можливостей для контенту.	Обмежений функціонал чатів.
Широкі можливості взаємодії з аудиторією через коментарі.	Відсутність можливості створення постів текстового контенту.
Простий та зручний інтерфейс.	Ризик залежності від алгоритмів TikTok, які можуть змінюватися.

Серед соціальних мереж українського походження варто виділити UkrOpen, інтерфейс якої можна побачити на рисунку 1.5.

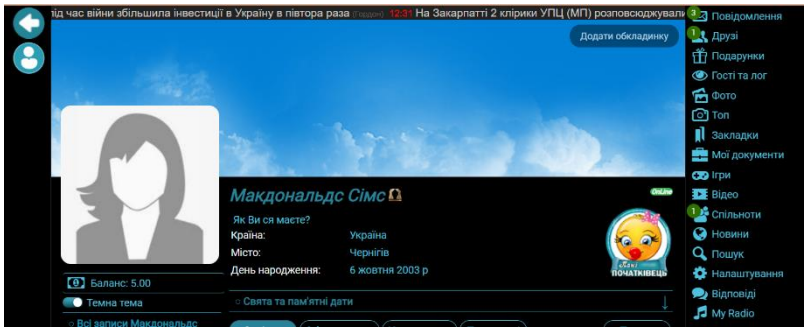

Рисунок 1.5 – Інтерфейс соціальної мережі UkrOpen

Ця соціальна мережа позиціонується як мережа, що створена з використанням сучасних технологій і розроблена з урахуванням побажань користувачів [4]. Детальніше інформацію про переваги та недоліки цієї мережі можна розглянути в таблиці 1.3.

Таблиця 1.3 – Переваги та недоліки UkrOpen для використання в бізнесі

Переваги	Недоліки
Створений як альтернатива популярним соціальним мережам.	Незручний інтерфейс та застарілий дизайн.
Наявність веб та мобільної версії для Android.	Відсутність мобільного додатку для IOS.
Можливість створення спільнот для обговорень.	Велика кількість неактуальної інформації.

Порівнюючи кожну з розглянутих соціальних мереж поставлено завдання розробити соціальну мережу для локального бізнесу в сфері SMM з урахуванням усіх вимог для зручної передачі інформації між працівниками сфери маркетингу соціальних мереж та їх клієнтами.

1.4 Постановка завдань

На основі дослідження предметної області було розроблено таблицю 1.4, яка показує формування задач для користувачів системи.

Таблиця 1.4 – Формування задач для користувачів системи

Потенційний користувач системи	Задачі, які потрібно вирішити	Проблеми, які виникають	Формування вирішення проблеми
SMM-спеціаліст	Пошук потенційних клієнтів, ведення комунікації, публікація контенту	Де знаходити потенційних клієнтів? Який контент викладати? Як налагодити комунікацію?	Знаходження потенційних клієнтів через публікацію контенту, комунікація із клієнтами через чат

Продовження таблиці 1.4

Потенційний клієнт	Знаходження SMM-спеціаліста, перегляд контенту smm-спеціаліста	Як знаходити SMM-спеціаліста? Яким чином переглядати контент smm-спеціаліста?	Знаходження SMM-спеціаліста через перегляд його контенту

Коли мова заходить про розроблювальну систему, є кілька ключових речей, які потрібно враховувати:

– можливість обміну повідомленнями між користувачами мережі зробить спілкування швидким та ефективним;

– можливість реєстрації на основі Google акаунту – вхід у систему стане легким, заощаджуючи час і зусилля користувачів;

– можливість створення та додавання фото та відео контенту, який притаманний соціальним мережам;

– інструменти для взаємодії з контентом користувачів;

– захист персональних даних користувачів;

– розробка зрозумілого інтерфейсу для легкого користування.

Отже, створюючи соціальну мережу для локального бізнесу в сфері SMM, важливо врахувати всі вимоги та особливості, які забезпечать ефективну роботу системи та задоволення потреб користувачів.

2 ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Аналіз варіантів використання системи

Діаграма варіантів використання — це форма вимог до системи або програмного забезпечення, яка показує очікувану поведінку, а не точний спосіб її реалізації. Моделювання варіантів використання допомагає проектувати систему з урахуванням точки зору кінцевого користувача. Згідно з вимогами, що визначені у розділі 1 пункт 4, була розроблена діаграма варіантів використання для системи соціальної мережі для локального бізнесу у сфері SMM. Ця діаграма дозволяє передбачити всі можливі дії користувачів системи. Стандартні елементи UML-діаграми використовуються для наочного відображення варіантів використання та їх взаємозв'язків. Зображення діаграми варіантів використання на рисунку 2.1 надає візуальне представлення сценаріїв використання системи, що допомагає розуміти їхній характер та взаємодію.

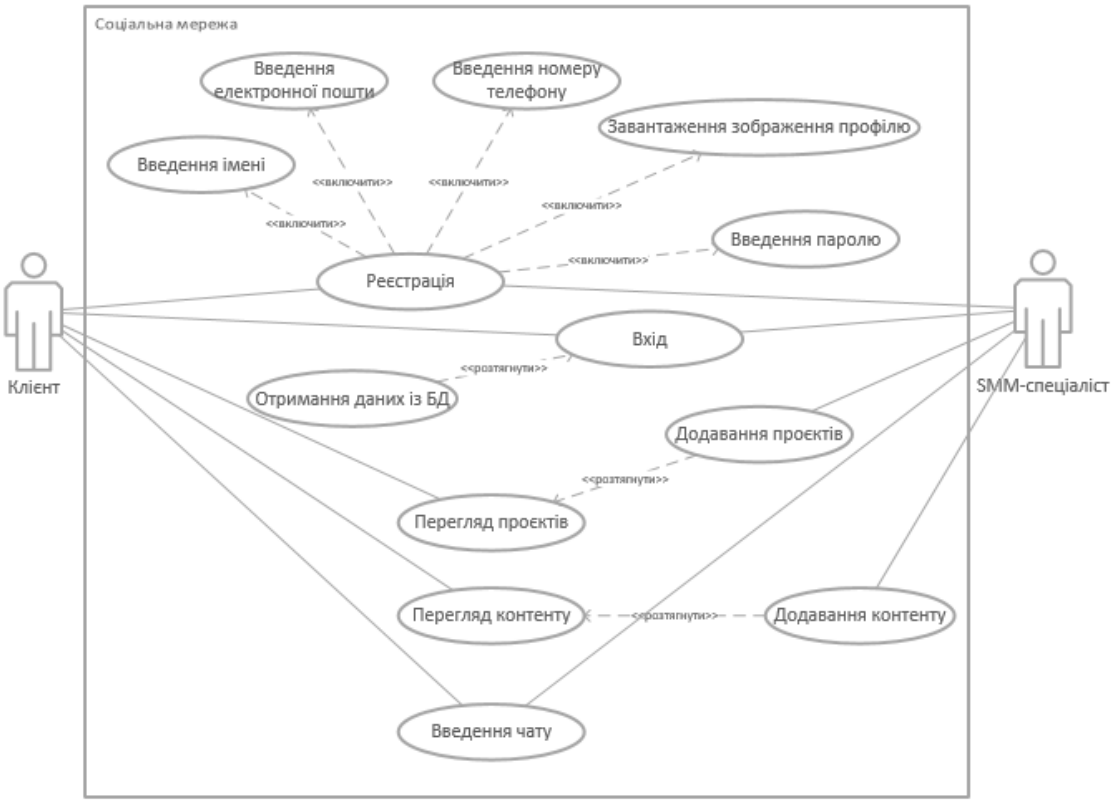

Рисунок 2.1 – UML-діаграма варіантів використання системи

Основними користувачами цієї системи є клієнт та SMM-спеціаліст. Для обох цих користувачів можуть бути використані такі прециденти, як реєстрація, що включає в себе введення імені, електронної пошти, номеру телефону, паролю та завантаження фото профілю, вхід, що отримує дані із бази даних, та ведення чату для комунікації в соціальній мережі.

Окремими прецидентами для користувача SMM-спеціаліст є додавання контенту та проєктів, які забезпечують функціонування таких прецидентів, як перегляд проєктів та контенту, що є притаманними для соціальної мережі.

Діаграма варіантів використання є потужним інструментом для аналізу та проектування системи, допомагаючи зрозуміти потреби користувачів та їх взаємодію з системою. Це важливий компонент процесу розробки програмного забезпечення, який дозволяє уточнити вимоги та функціональність системи.

2.2 Вибір методології розробки соціальної мережі

Створення соціальної мережі – складне завдання, яке вимагає глибокого розуміння технічних, дизайнерських, інформаційних та соціальних аспектів. Щоб ефективно розробити соціальну мережу, необхідно обрати правильний підхід до розробки, враховуючи деталі проекту, потреби користувачів та технічні вимоги. Беручи це до уваги, для цієї системи було обрано принцип чистої архітектури.

Чиста архітектура – це принцип розробки програмного забезпечення, який має на меті створити модульну, масштабовану кодову базу та визначати залежності між різними рівнями програми.

Дотримуючись принципів чистої архітектури можна досягти таких переваг, як:

– модульність частин кодової бази;

– можливість тестування окремих рівнів;

– полегшена модифікація та доповнення кодової бази в міру зростання програми.

Концепція чистої архітектури зазвичай складається з наступних рівнів:

– рівень презентації – це рівень, що містить компоненти інтерфейсу користувача, такі як віджети, екрани та представлення. Він відповідає за обробку взаємодії користувача та відтворення інтерфейсу користувача. Бізнес-логіка та деталі реалізації доступу до даних не повинні використовуватися на рівні презентації;

– рівень домену – це рівень домену, що представляє основну бізнес-логіку програми. Він містить варіанти використання, сутності та бізнес-правила. Варіанти використання визначають операції або дії, які можна виконувати в програмі. Сутності представляють основні об’єкти в домені та інкапсулюють їх поведінку та стан. Рівень домену не повинен залежати від будь-якої конкретної структури чи технології;

– рівень даних – це рівень даних, який відповідає за пошук і зберігання даних. Він складається зі сховищ і джерел даних. Репозиторії забезпечують рівень абстракції для доступу та маніпулювання даними. Вони визначають договір або інтерфейс для операцій з даними, які реалізуються джерелами даних. Джерелами даних можуть бути віддалені API, локальні бази даних або інші зовнішні постачальники даних. Рівень даних захищає рівень домену від деталей зберігання та вилучення даних [6].

Візуальне зображення структури чистої архітектури, яка була описана вище, зображено на рисунку 2.2 [7].

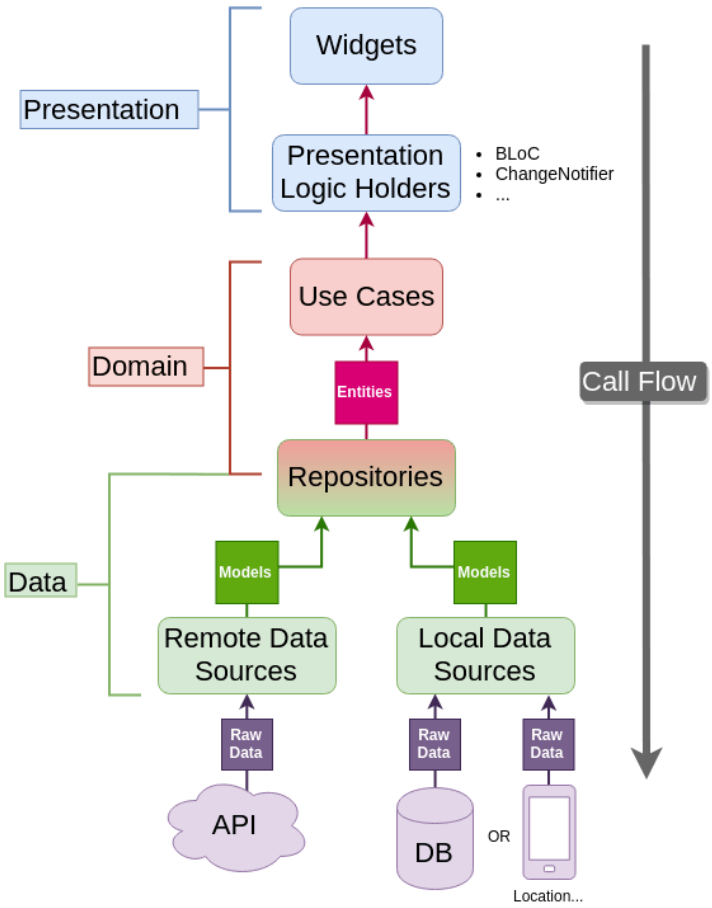

Рисунок 2.2 – Структура чистої архітектури

Загальна робота чистої архітектури полягає в наступному. Потік викликів починається з віджета, який отримує подію від користувача, наприклад, натискання кнопки, і передає цю подію тримачу презентаційної логіки, який в свою чергу обробляє подію від віджета і викликає відповідний випадок використання для виконання бізнес-логіки. Варіант використання виконує бізнес-логіку і взаємодіє з репозиторіями для отримання або зберігання даних. Репозиторій звертається до віддалених або локальних джерел даних для отримання необхідної інформації та повертає отримані дані випадку використання. Віддалене джерело даних або локальне джерело даних здійснюють запити до API або до локальної бази даних для отримання одноманітних даних, які перетворюють в моделі і повертають їх до репозиторію. Варіант використання отримує дані від

репозиторію, обробляє дані і повертає результат тримачу презентаційної логіки. Тримач логіки представлення отримує результат від випадку використання і оновлює відповідний віджет з новими даними.

Таким чином, чиста архітектура забезпечує чіткий поділ відповідальностей між різними компонентами. Потік викликів демонструє, як взаємодіють між собою різні рівні системи — від користувацького інтерфейсу до джерел даних. Така структура дозволяє легко масштабувати додаток, додавати нові функції і змінювати джерела даних без значних змін у всій системі. Це робить архітектуру гнучкою і придатною для різноманітних сценаріїв використання.

Щоб реалізувати чисту архітектуру у Flutter, ви можете організувати свою кодову базу за допомогою пакетів або папок для кожного шару та забезпечити потік залежностей від рівня презентації до рівня домену, а потім до рівня даних.

Архітектурні шаблони, такі як BLoC, можна використовувати в поєднанні з принципами чистої архітектури для реалізації взаємодії між рівнями та управління станом у програмах Flutter.

BLoC є одним із патернів керування станом для додатків Flutter. Його можна використовувати для легкої обробки всіх можливих станів програми [8].

Під час використання BLoC користувач збирається створювати події для ініціювання взаємодії з програмою, а відповідальний блок збирається видавати запитувані дані зі станом. На прикладі реєстрації у соціальній мережі це буде виглядати так:

1) користувач натискає кнопку для входу в систему;

2) подія запускається, і вона інформує, що користувач хоче отримати відповідні дані для входу;

3) блок запитує ці дані зі сховища, яке відповідає за підключення до API для отримання даних;

4) коли блок має дані, він визначить, чи отримання даних є успішним чи помилковим, а потім видає стан;

5) представлення прослуховуватиме всі можливі стани, які блок може видавати, щоб відреагувати на них. Наприклад, якщо блок видає статус «Успіх», подання збирається перебудувати його зі даними для входу, але якщо стан «Помилка», подання буде перебудовано з повідомленням про помилку.

Візуальне зображення принципу роботи патерну BLoC зображено на рисунку 2.3 [8].

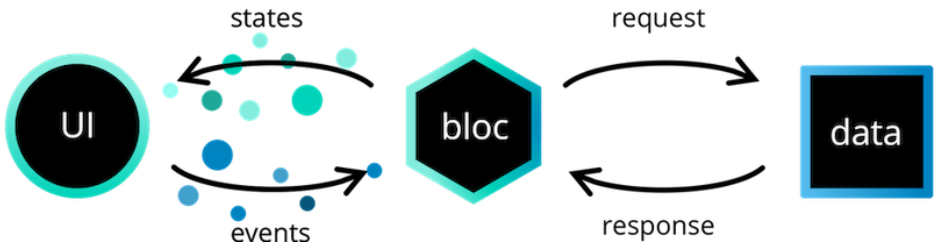

Рисунок 2.3 – Принцип роботи патерну BLoC

Для проєктування соціальної мережі для локального бізнесу на основі чистої архітектури та принципу роботи патерна BLoC, було створено структуру директорій, яка зображена на рисунку 2.4.

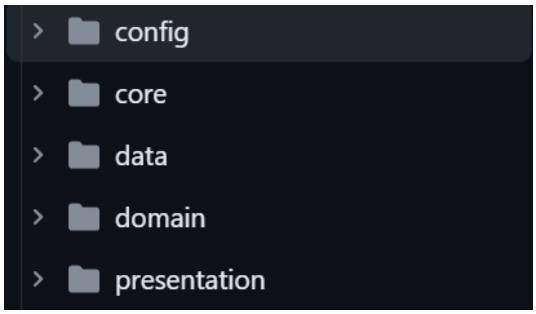

Рисунок 2.4 – Структура папок проєкту

Структура директорій передбачає розділення коду на частини, кожна з якої відповідає за свою частину функціональності, що дозволяє легко

змінювати та масштабувати проєкт. Розташування компонентів у цьому проєкті виглядає наступним чином:

– папка config містить конфігураційні файли, які використовуються для налаштування проєкту, зокрема файли з ключами API та налаштування бази даних;

– папка core розміщує в собі базову логіку та утиліти, які використовуються у всьому проєкті: константи, загальні функції, інтерфейси та базові класи;

– папка data місить всі класи, які відповідають за роботу з даними: репозиторії, моделі даних і джерела даних;

– папка domain містить основну бізнес-логіку (use cases), бізнес-об'єкти та репозиторії інтерфейсів;

– папка presentation містить все, що пов'язано з відображенням даних користувачам та взаємодією з ними.

Застосування патерну Bloc у розробці знаменує собою значний крок у напрямку створення більш надійних, придатних для обслуговування та масштабованих програм. Наголошуючи на розділенні завдань, чітких межах рівнів і ефективному управлінні станом, він встановлює високий стандарт практики розробки додатків. Заглядаючи вперед, еволюція шаблонів архітектури у Flutter, ймовірно, зосередиться на підвищенні вашої продуктивності, продуктивності програми та адаптації до нових технологій. Оскільки екосистема Flutter продовжує розвиватися, використання таких комплексних архітектурних структур стане вкрай важливим для створення складних програм, які витримують випробування часом [9].

2.3 Проєктування користувацького інтерфейсу

Створення привабливого, інтуїтивно зрозумілого та зручного графічного інтерфейсу для мобільних додатків є важливим кроком у процесі

розробки програмного забезпечення. Добре розроблений інтерфейс забезпечує плавну взаємодію з користувачем, що є важливими елементами успіху продукту на ринку.

Для користувацького дизайну соціальної мережі необхідно передбачити макети екранів, які допоможуть користувачам легко реєструватися та здійснювати вхід у додаток, налаштовувати профіль, здійснювати пошук, переглядати контент та ділитися повідомленнями.

Для реєстрації в мережі побудовано макет екрану, що на рисунку 2.5.

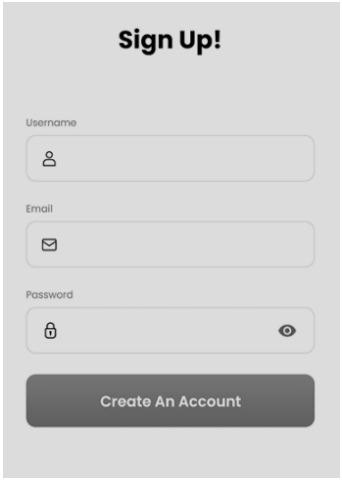

Рисунок 2.5 – Макет сторінки для реєстрації

Для входу в мережу побудовано макет екрану, що на рисунку 2.6.

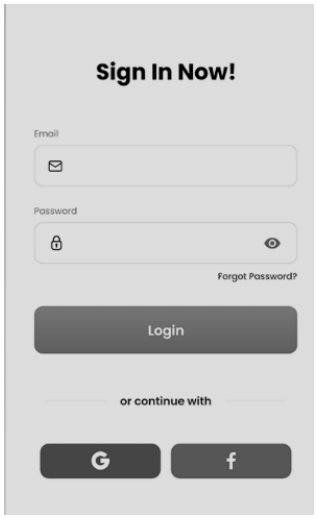

Рисунок 2.6 – Макет сторінки для входу

Також побудовано макет головної сторінки, що зображений на рисунку 2.7.

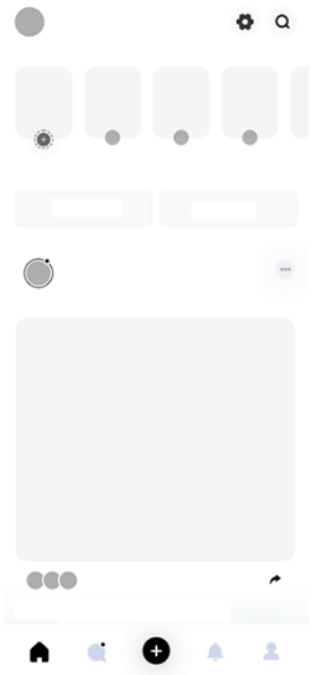

Рисунок 2.7 – Макет вигляду головної сторінки

Макет сторінки профілю можна переглянути на рисунку 2.8.

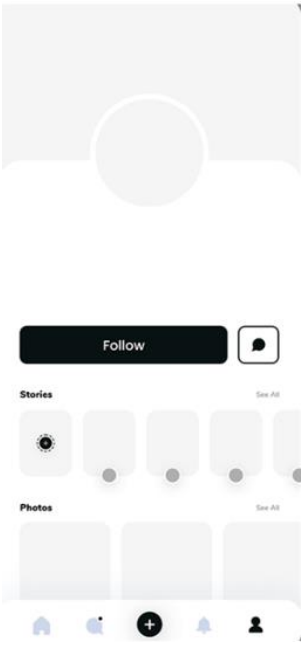

Рисунок 2.8 – Макет вигляду сторінки профілю

Макет чату між користувачами системи можна переглянути на рисунку 2.9.

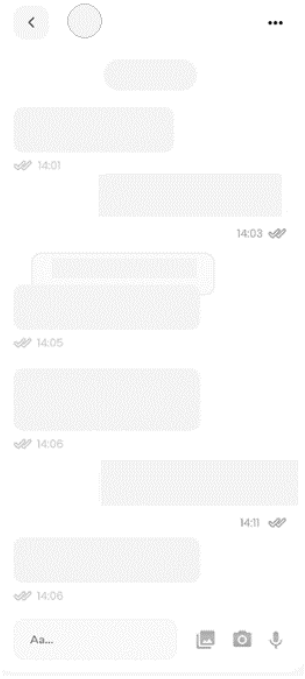

Рисунок 2.9 – Макет вигляду чату

Створені макети побудовано у мінімалістичному стилі, який дозволяє користувачам системи не відволікатися на непотрібні елементи дизайну.

Загалом, графічний інтерфейс користувача для мобільних застосунків вимагає уваги до деталей і врахування потреб користувача. Створення привабливого і функціонального графічного інтерфейсу, який відповідатиме потребам користувачів, допоможе програмі досягти успіху за допомогою цих аспектів.

2.4 Вибір зовнішньої бази даних відповідно до спроєктованої системи

Вибір бази даних є важливим кроком у проектуванні та впровадженні системи. Правильний вибір бази даних визначає ефективність, продуктивність і надійність системи. Основними критеріями вибору бази даних є відповідність потребам системи, масштабованість, продуктивність,

безпека та інші аспекти, що впливають на функціональність та ефективність.

Нереляційні бази даних мають кілька ключових переваг порівняно з реляційними базами даних:

– не вимагають строго визначеної схеми, що дозволяє зберігати дані з різною структурою без необхідності зміни базової схеми;

– забезпечують вищу швидкодію при доступі до даних, оскільки вони оптимізовані для певних типів операцій, таких як пошук, читання або запис;

– пропонують простіші механізми масштабування, особливо горизонтального масштабування, що дозволяє легко додавати нові вузли або сервери для збільшення обробки обсягів даних;

– дозволяють зберігати дані у вигляді документів, ключ-значення, колонок або графів, що відповідає різним потребам додатків та типам даних;

– добре підтримують розподілений доступ до даних для застосувань з високими вимогами до доступності та масштабованості;

– надають інструменти для зручного управління обсягами даних та витратами на обробку.

Ці переваги роблять нереляційні бази даних привабливим вибором для сучасних додатків, які мають високі вимоги до швидкодії, гнучкості схеми та масштабованості. Враховуючи ці фактори, для зберігання даних було обрано хмарне середовище Firebase.

Firebase — це комплексна платформа, яка дозволяє створювати мобільні програми з широким набором функцій і можливостей. Спочатку це був невеликий стартап, що перетворився на важливий інструмент для створення більш універсальних міжплатформних програм. Основна перевага полягає в тому, що розробникам не потрібно створювати серверний компонент проекту, що дозволяє їм зосередитися на взаємодії з

користувачем і дизайні інтерфейсу, тим самим прискорюючи процес розробки. Інтегрувавши Firebase із фреймворком Flutter, можна швидко розробляти ефективні програми як для Android, так і для iOS, зосереджуючись на вирішенні широкого кола завдань. Firebase є одним з рішень BaaS (Backend as a Service), яке надає розробникам широкі можливості. Вона об'єднує в собі сервер, базу даних, хостинг та систему аутентифікації в одній платформі [10].

Беручи до уваги всі переваги, згадані раніше, для зберігання даних, що використовуються в системі, була обрана нереляційна база даних Realtime Database, яка пропонує API для синхронізації даних між клієнтами та їх зберігання в хмарному середовищі.

Усі дані Firebase Realtime Database зберігаються як об’єкти JSON. Базу даних можна порівняти з хмарним деревом JSON. На відміну від бази даних SQL, тут немає структурованих таблиць або окремих записів. Коли дані включаються в дерево JSON, вони стають вузлом у існуючій структурі JSON, оснащеній власним унікальним ключем, який можна призначити різним ідентифікаторам, таким як ідентифікатори користувачів або семантичні імена.

У Firebase Realtime Database одиницею зберігання є документ. Документ представляє собою невеликий запис, який містить поля, що зберігаються разом з їхніми значеннями. Кожен документ ідентифікується унікальною назвою. В ієрархічній структурі бази даних Firebase, документи зазвичай організовані в колекції, а кожна колекція може містити один або більше документів. Кожен документ може мати різні поля, які зберігають різноманітну інформацію, таку як рядки, числа, списки або навіть вкладені об'єкти. Така структура, що зображена на рисунку 2.10, дозволяє організовувати дані у відповідності з потребами додатку і ефективно взаємодіяти з ними через Firebase API.

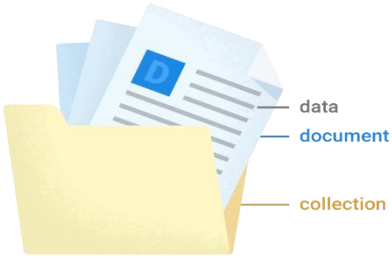

Рисунок 2.10– Принцип зберігання даних у Firebase Realtime Database

Відповідно до цієї структури зберігання даних у Firebase Realtime Database, база даних для соціальної мережі для локального бузнесу в сфері SMM має чотири колекції.

Перша колекція – це колекція SMM_Specialists. Тут з’являються дані SMM-спеціалістів, після реєстрації у системі. У ній є п’ять полів:

– name;
– surname;
– email;
– password_hash;
– uuid.

Друга колекція – це колекція Clients. Тут з’являються дані клієнтів, що прагнуть знайти SMM-спеціаліста, після реєстрації у системі. У ній є п’ять полів:

– name;
– surname;
– email;
– password_hash;
– uuid.

Третя колекція – це колекція Projects. Тут з’являються дані про проєкти, які додані SMM-спеціалістами, після входу їх у систему. У ній є вісім полів:

– name;
– description;
– tasks;
– status;
– specialist_id;
– client_id;
– password_hash;
– uuid.

Третя колекція – це колекція Messages. Тут зберігаються надіслані повідомлення у чаті, який ведуть SMM-спеціалісти та клієнти. У ній є вісім полів:

– text;
– time;
– specialist_id;
– client_id.

Враховуючи тісну взаємодію між Flutter та Firebase, Realtime Database є дуже потужним інструментом для зберігання та синхронізації даних у хмарі. Однією з головних особливостей Realtime Database є можливість ефективної синхронізації даних між різними клієнтами та сховищем у хмарі. Це означає, що якщо один клієнт змінює дані, зміни автоматично синхронізуються з іншими клієнтами, які працюють з тими ж даними. Це дозволяє користувачам бачити оновлення в режимі реального часу і спілкуватися з іншими користувачами без затримок, забезпечуючи високу продуктивність і зручність використання.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВННЯ СИСТЕМИ

3.1 Опис необхідного програмного забезпечення

Для розробки соціальної мережі для місцевого бізнесу в сфері SMM потрібне надійне та універсальне середовище розробки, тому в якості основного середовища було обрано Visual Studio Code[11].

Це середовище володіє рядом переваг над своїми конкурентами, зокрема:

– здійснює підтримку широкого спектру мов програмування;

– широкі можливості користувацького налаштування, дозволяючи налаштовувати інтерфейс і комбінації клавіш на власний розсуд і пристосувати середовище кодування до своїх конкретних потреб;

– велика спільнота розробників, які створюють і підтримують розширення та плагіни, що додають нові функції до редактора для допомоги під час написання коду;

– швидка та ефективна робота в середовищі розробки на слабких комп’ютерах – чудовий вибір для розробників, яким потрібен редактор коду, який не сповільнює комп’ютер.

Незважаючи на переваги, Visual Studio Code включає в себе такий недолік, як потреба додаткової конфігурації для користувачів, які вперше знайомляться із цим середовищем.

Окрім середовища розробки, важливим інструментом для створення та тестування застосунку без фізичного пристрою є емулятор. Одним із найдоступніших на даний момент є Android Virtual Device (AVD), який можна створювати та налаштовувати у середовищі Android Studio [12]. Створений віртуальний телефон можна побачити на рисунку 3.1.

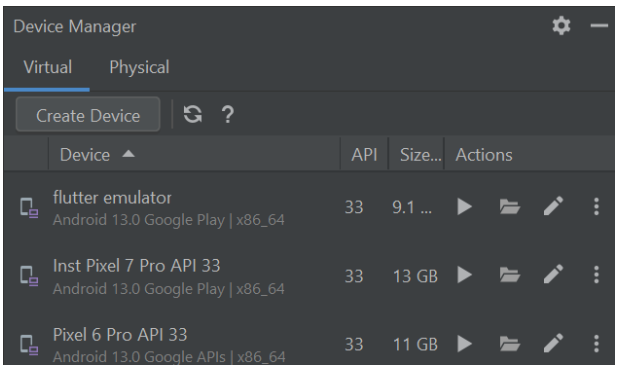

Рисунок 3.1 – Створений емулятор в середовищі Android Studio

Також для розробки системи було обрано Flutter – фреймворк із відкритим кодом, що розроблений і підтримується Google. Flutter використовується для розробки інтерфейсу користувача різних програм на різних платформах з використанням єдиної кодової бази. Цей фреймворк використовує мову програмування з відкритим кодом Dart, яку розробила Google. Під час запуску у 2018 році, він в основному підтримував розробку мобільних додатків. Тепер Flutter підтримує розробку програм на наступних платформах: iOS, Android, Web, Windows, MacOS та Linux. [13]

Переваги цього кросплатформеного фреймворку наступні:

– дозволяє розробникам використовувати єдину мову програмування та кодову базу для розробки програм для різних платформ.

– використовує мову програмування Dart, що забезпечує швидку та ефективну роботу;

– отримання узгоджених візуальних ефектів незалежно від платформи, через доступ до програми, завдяки використанню відкритої графічної бібліотеки Skia від Google;

– надання зручних для розробника інструментів, такі як hot reload, яке дозволяє розробникам переглядати зміни коду без перезапуску програми, і інспектор віджетів, який спрощує візуалізацію та усунення несправностей макетів інтерфейсу користувача.

Вибір цих інструментів та середовищ дозволив створити ефективне та надійне рішення для створення соціальної мережі для локального бізнесу в сфері SMM, забезпечуючи високу якість розробки та зручність використання для кінцевих користувачів.

3.2 Процес розробки соціальної мережі

Процес розробки соціальної мережі для локального бізнесу в сфері SMM включає в себе створення наступних програмних вікон:

– вікно входу, яка має необхідні поля для введення основної інформації для реєстрації нового користувача, що забезпечує доступ до функцій соціальної мережі;

– головний екран – основна стрічка новин, де користувачі бачать пости від тих, на кого вони підписані та події, які доступні добу, забезпечуючи основну взаємодію в мережі;

– вікно для створення тимчасової події, що доступна 24 години;

– вікно особистого профілю користувачі, де можна переглядати та редагувати інформацію про себе, бачити власні пости і події;

– вікно пошуку користувачів ім'ям для швидкої комунікації із ними;

– вікно для публікування контенту, що притаманний соціальним мережам, а саме нових фото та відео постів;

– вікно активності для перегляду всіх дій користувача та взаємодій з його контентом від інших користувачів.

При створенні соціальної мережі для локального бізнесу в сфері SMM в процесі розробки враховуються кілька ключових функцій, серед яких:

– дозвіл обмінюватися миттєвими повідомленнями, сприяючи безперебійному спілкуванню між користувачами;

– можливість залишати коментарі до постів інших користувачів, сприяючи обговоренням та взаємодії між користувачами;

– функція, що дозволяє ділитися власними постами або постами інших користувачів у своїй стрічці або в особистих повідомленнях;

– можливість підписуватися на інших користувачів для отримання їх створеного контенту у стрічці новин;

– функція для відписки від інших користувачів при попередній підписці на них;

– завантаження та зміна фото профілю користувача;

– можливість редагувати вже раніше опубліковані пости.

Загалом, процес розробки соціальної мережі для локального бізнесу в сфері SMM є багатоступеневим і охоплює створення різноманітних вікон та функцій, що забезпечують зручну взаємодію користувачів з системою. Все це спрямовано на створення ефективної та зручної платформи для взаємодії користувачів і підтримки їхньої активності в мережі.

3.3 Реалізація користувацького інтерфейсу

Для розроблювальної соціальної мережі для локального бізнесу в сфері SMM була обрана назва «КОМЕНТ», яка розшифровується, як «Комунікації для ефективного маркетингу та налагодженні таргету», що відображає основну мету створення – збільшення продажів за рахунок соціальних мереж для бізнесу. Цей застосунок створено для операційних системи Android та iOS.

Основними елементами користувацького інтерфейсу у фреймворку Flutter є віджити, які розроблені таким чином, щоб їх можна було легко налаштувати. Flutter досягає цього завдяки композиційному підходу. Це означає, що більшість віджетів складаються з менших віджетів, а

найпростіші віджети мають певне призначення. Це дозволяє комбінувати або редагувати віджети для створення нових відповідно до вимог.

Flutter відтворює віджети за допомогою власного графічного механізму замість того, щоб покладатися на вбудовані віджети платформи. Таким чином, користувачі матимуть схожий вигляд програми Flutter на різних операційних системах. Цей підхід також надає розробникам гнучкість, оскільки деякі віджети Flutter можуть виконувати функції, які не можуть виконувати віджети для певної платформи.

Flutter включає в себе обширний каталог віджетів, який включає в себе два основні стилі: Cupertino (віджети в стилі iOS) і Material Components (віджити в стилі Android). Візуальну різницю цих віджетів можна побачити на рисунку 3.2.

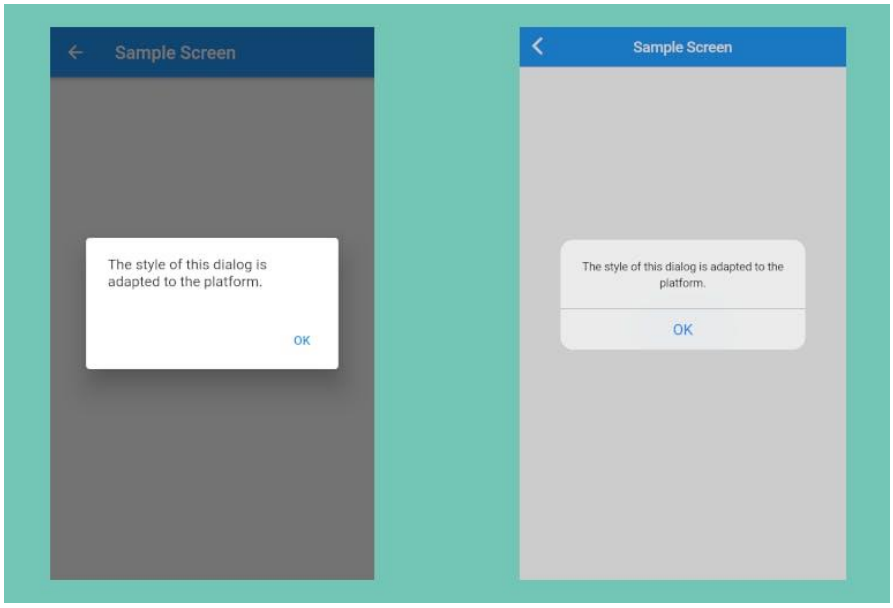

Рисунок 3.2 – Візуальна різниця між Material Components (зліва) та Cupertino (справа)

У Flutter запуск програми завжди починається із файлу main.dart (лістинг 3.1).

```
import 'package:flutter/material.dart';
import 'package:google_fonts/google_fonts.dart';
import 'config/colors.dart';
import 'screens/onboarding/onboarding_screen.dart';
void main() {
  runApp(const MyApp());
}
class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'КОМЕНТ',
      debugShowCheckedModeBanner: false,
      theme: kAppThemeData,
      home: const OnboardingScreen(),
    );
  }
}
```

Лістинг 3.1 – Ініціалізація проєкту

Для підтримки пакетних залежностей проєкту було створено файл pubspec.yaml, вміст якого зображено в лістингу 3.2.

```
name: KOMENT
description: A new Flutter project.
publish_to: 'none'
version: 1.0.0+1
environment:
  sdk: ">=2.17.3 <3.0.0"
dependencies:
  flutter:
```

```
    sdk: flutter
  flutter_staggered_grid_view: ^0.6.2
  flutter_svg: ^1.1.1+1
  google_fonts: ^3.0.1
  intl: ^0.17.0
  cupertino_icons: ^1.0.2
dev_dependencies:
  flutter_test:
    sdk: flutter
  flutter_lints: ^2.0.0
flutter:
  uses-material-design: true
  assets:
    - assets/images/
    - assets/icons/
```

Лістинг 3.2 – Пакетні залежності, які використовуються в проєкті

Для встановлення колірної гами проєкту було створено файл colors.dart, програмний код якого можна побачити на лістингу 3.3.

```
import 'package:flutter/material.dart';
const kBlack = Color(0xFF000000);
const kWhite = Color(0xFFFFFFFF);
const k1Gray = Color(0xFF4E4E4E);
const k1LightGray = Color(0xFFC4C4C4);
const k2AccentStroke = Color(0xFF25A0B0);
const k2MainThemeColor = Color(0xFFE1F6F4);
const k3GradientAccent = RadialGradient(
  colors: [Color(0xffffe1e0), k2MainThemeColor],
  center: Alignment.topCenter,
  radius: 0.8,
);
const k3Pink = Color(0xFFF5C8C6);
```

```
const k3AccentLines = Color(0xFFEEF2E2);
const kCaption = Color(0xFF656565);
const kSelectedTabColor = Color(0xFF7DB9B3);
```

Лістинг 3.2 – Колірна гама проєкту

Для зручної навігації в додатку було створено файл mobile_screen_layout.dart, вміст якого зображено додатку А.

Для розробки вікна входу були створені поля для введення електронної пошти та паролю, а також кнопку для входу. Підключення до Firebase Authentication забезпечує обробку аутентифікації користувачів. Користувацький інтерфейс цього вікна можна переглянути на рисунку 3.3.

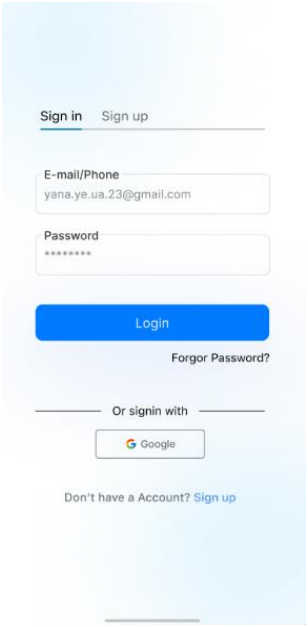

Рисунок 3.3 – Вікно входу

Програмний код для розробки вікна входу міститься в додатку Б.

Головний екран включає в себе віджети, які відображають пости користувачів, та поді. Використання StreamBuilder дозволяє оновлювати стрічку новин у реальному часі. Переглянути головний екран можна на рисунку 3.4.

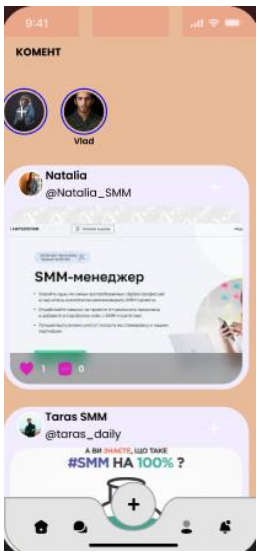

Рисунок 3.4 – Головний екран

Програмний код для головного екрану можна побачити в додатку В.

Для створення вікна події застосовується інтеграція з Firebase Realtime Database, яка забезпечує збереження та видалення подій. Вікно створення події можна переглянути на рисунку 3.5.

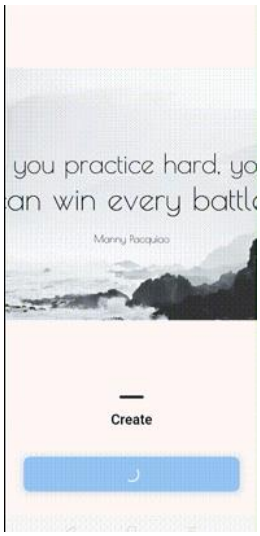

Рисунок 3.5 – Вікно створення події

Програмний код для створення події знаходиться в додатку Г.

Вікно профілю користувача включає відображення інформації про користувача, його постів і подій, а також можливість редагування профілю. Використання ListView та інтерфейсу для редагування даних дозволяє користувачам зручно переглядати та змінювати свої дані. Вікно профілю можна побачити на рисунку 3.6.

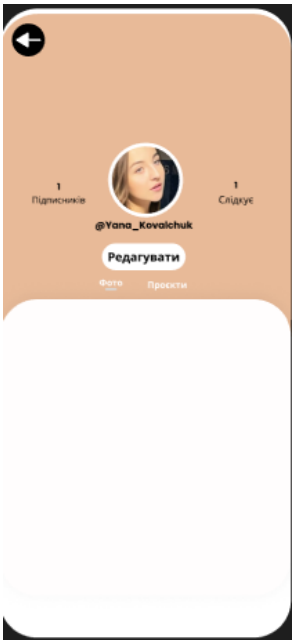

Рисунок 3.6 – Вікно профілю користувача

Програмний код для перегляду профілю користувача можна побачити в додатку Д.

Вікно пошуку включає в себе текстове поле для введення імені користувача та відображення результатів пошуку у вигляді списку. Використання функції запиту до Firebase забезпечує швидкий пошук та відображення користувачів. Вікно пошуку зображено на рисунку 3.7.

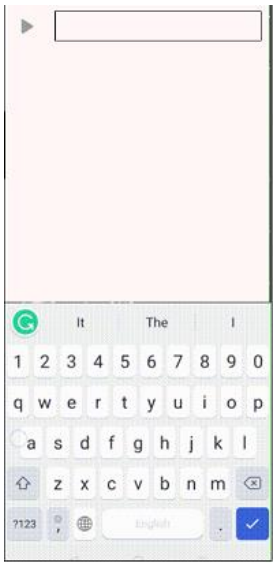

Рисунок 3.7 – Вікно пошуку

Програмний код для перегляду вікна пошуку можна побачити в додатку Е.

Вікно для завантаження фото або відео включає інтеграцію з Firebase Storage та Realtime Database та забезпечує збереження контенту в мережі, рисунок 3.8.

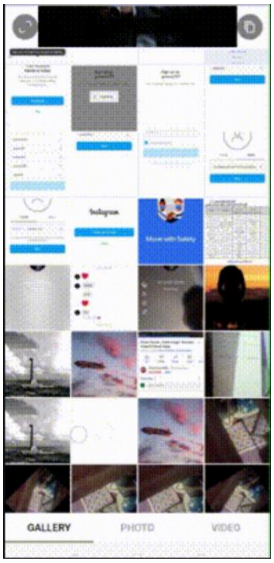

Рисунок 3.8 – Вікно додавання контенту

Програмний код для перегляду вікна пошуку можна побачити в додатку Ж.

Вікно активності відображає такі дії користувача, як лайки, коментарі та нові підписки, у вигляді списку. Використання StreamBuilder дозволяє динамічно оновлювати активність у реальному часі, забезпечуючи користувачів актуальною інформацією. Це вікно зображено на рисунку 3.9.

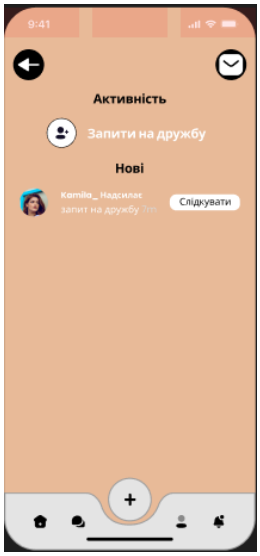

Рисунок 3.9 – Вікно активності

Програмний код для вікна актиності знаходиться в додатку К.

Вікно чату, що зображене на рисунку 3.10, відображає усі початі чати, які підтягуються за допомогою бази даних Realtime Database.

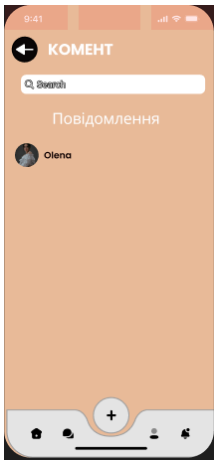

Рисунок 3.10 – Вікно списку чатів

Програмний код для вікна актиності знаходиться в додатку Л.

Під час розробки основних екранів соціальної мережі для створення зручного та інтуїтивно зрозумілого інтерфейсу були наступні віджети Flutter:

– Text, що використовується для відображення тексту на екрані та підтримує різні стилі, розміри шрифту та колір тексту.

– Container, який дозволяє створювати прямокутні області на екрані з можливістю налаштування власних властивостей, таких як колір фону, відступи та межі.

– ListView, який дозволяє створювати прокручувані списки елементів, підтримує вертикальну та горизонтальну прокрутку.

– Column і Row, які використовуються для розміщення віджетів у стовпці або рядку відповідно.

– AppBar, що використовується для створення верхньої панелі.

– TextField, який дозволяє користувачам вводити текст.

– TextButton, яка дозволяє створювати кнопки з різними стилями натискання.

– Image, що використовується для відображення зображень на екрані.

– Stack, який дозволяє розміщати віджети один поверх одного, що дає більшу гнучкість у розміщенні елементів на екрані.

– Navigator, який використовується для управління навігацією між вікнами додатку.

Ці віджети дозволяють створювати складні та динамічні користувацькі інтерфейси, які ефективно взаємодіють з користувачем і надають потрібний функціонал.

3.4 Тестування готового програмного продукту

При створенні соціальної мережі для локального SMM-бізнесу мануальне тестування є важливим кроком. Воно забезпечує виявлення недоліків у системі, що не завжди можливо автоматизованими методами. Мануальне тестування дозволяє перевірити взаємодію користувача з інтерфейсом, логіку програми та відповідність реалізованих функцій заданим вимогам.

Для початку слід пройти авторизацію у системі, а саме ввести логін та пароль або увійти за допомогою акаунту Google (рисунок 3.10).

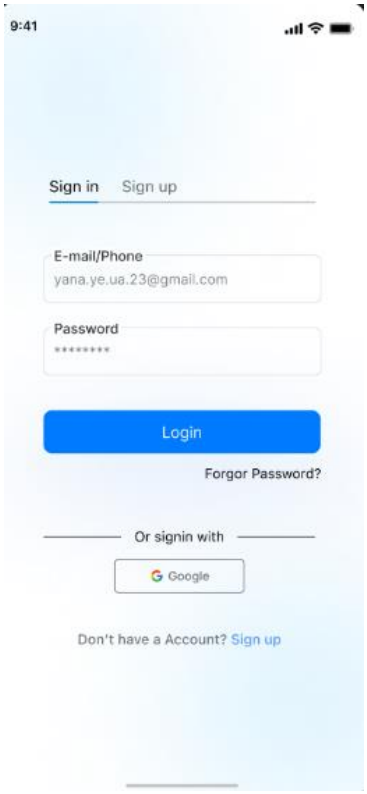

Рисунок 3.10 – Вхід в систему

Якщо користувач не має акаунту, то він може зареєструватися у вікні, що зображене на рисунку 3.11.

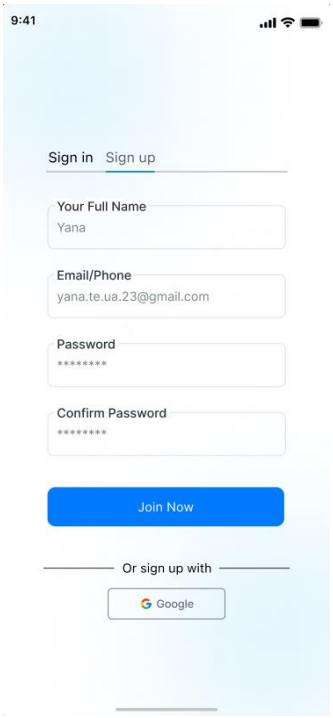

Рисунок 3.11 – Створення акаунту в мережі

При неправильному поданні даних для реєстрації з’являється повідомлення, яке зображено на рисунку 3.12.

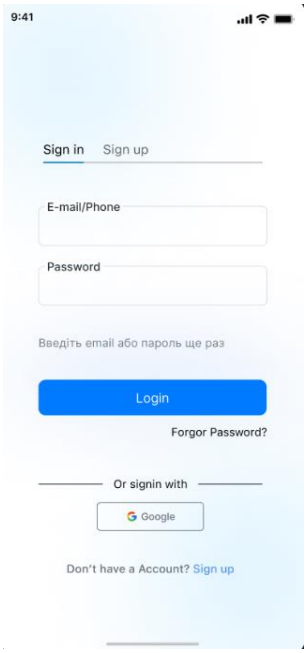

Рисунок 3.12 – Невдалий вхід до системи

Створення події, що на рисунку 3.13, являє собою додавання тимчасового контенту.

Рисунок 3.13 – Створення події

Для публікації контенту користувач вибирає необхідне фото чи відео і натискає стрілочку, рисунок 3.14.

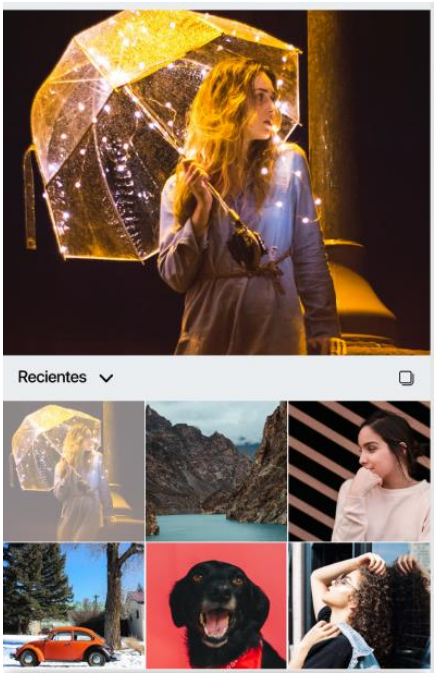

Рисунок 3.14 – Публікація контенту

Обмін миттєвими повідомленнями дозволяє комунікацію між двома користувачами системи, рисунок 3.15.

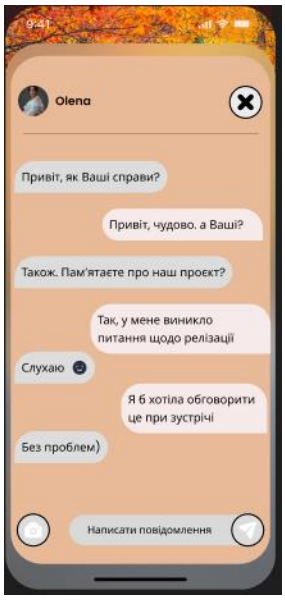

Рисунок 3.15 – Вікно чату в реальному часі

На основі цього тестування створено тестові сценарії, що зображені в таблиці 3.1, які містить назву тесту, опис кроків для його виконання, очікувані та отримані результати. Такий підхід дозволяє послідовно тестувати всі аспекти функціонування програми, забезпечуючи високу якість і задоволеність користувачів.

Таблиця 3.1 – Мануальне тестування соціальної мережі

Назва тесту	Кроки для виконання	Очікуваний результат	Отриманий результат
Вхід до системи	1. Відкрити додаток. 2.Ввести правильний email і пароль. 3. Натиснути кнопку "Вхід".	Користувач успішно входить до системи та перенаправляється на головний екран.	Виконано успішно

Продовження таблиці 3.1

Реєстрація в системі	1. Відкрити додаток. 2.Ввести дані, які необхідні для реєстрації. 3. Натиснути кнопку "Приєднатися".	Користувач успішно реєструється в системі.	Виконано успішно
Невдалий вхід до системи	1. Відкрити додаток. 2. Ввести неправильний email або пароль. 3. Натиснути кнопку "Вхід".	Показується повідомлення про помилку "Введіть email або пароль ще раз".	Виконано успішно
Створення події	1. Відкрити додаток. 2. Перейти на екран створення події. 3. Ввести деталі події. 4. Натиснути кнопку "Створити".	Подія зберігається у відображається в стрічці новин.	Виконано успішно
Публікація контенту	1. Відкрити додаток. 2. Перейти до вікна публікації. 3. Завантажити фото або відео. 4. Натиснути кнопку для публікації.	Фото або відео відображається в стрічці новин.	Виконано успішно

Продовження таблиці 3.1

Обмін миттєвими повідомленнями	1. Відкрити додаток. 2. Вибрати користувача для повідомлення. 3. Ввести повідомлення у текстове поле. 4. Натиснути кнопку "Надіслати".	Повідомлення успішно надсилається користувачу і відображається у чаті.	Виконано успішно

Під час проведення тестування були розглянуті основні сценарії взаємодії користувачів з системою, включаючи вхід, створення подій, редагування профілю, пошук користувачів, публікацію контенту, перегляд активності та обмін миттєвими повідомленнями. Завдяки використанню тестових сценаріїв стало можливо точно виявляти та виправляти помилки, тим самим підвищуючи стабільність і надійність програми.

Результати ручного тестування підтвердили, що впроваджена система відповідає визначеним вимогам і пропонує зручний та інтуїтивно зрозумілий інтерфейс для кінцевих користувачів.

4 ТЕХНІКО-ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ

4.1 Аналіз ринку

Соціальна мережа для локального бізнесу в сфері SMM виступає посередником між SMM-спеціалістами та їх потенційними клієнтами, що демонструє її необхідність у бізнесі. Полегшена комунікація допомагає значно заощадити такий важливий ресурс як час, який можна використати для інших цілей, що допоможуть отримати більший прибуток для обох сторін.

На ринку соціальних мереж існує безліч програмних рішень, деякі з яких було проаналізовано в пункті 1.3 розділу 1, що демонструють свою популярність серед користувачів, незалежно від віку, статті, країни чи матеріального благополуччя. Згідно із цим, локальному та глобальному бізнесу важливо вести сторінки соціальних мережах, які будуть мати красиве оформлення та зможуть заохотити користувачів щодо покупки своїх товарів чи послуг.

Серед найпопулярніших ринків збуту мобільних додатків виступають такі популярні маркетплейси Google Play для Android-пристроїв та App Store для пристроїв на базі IOS. Не зважаючи на схожий функціонал, обидві торгові площадки мають свої переваги та недоліки, які зображено у таблиці 4.1.

Таблиця 4.1 – Стандартні елементи діаграми варіантів використання

Google Play		App Store	
Переваги	Недоліки	Переваги	Недоліки
Велика кількість користувачів	Фрагментація пристроїв	Безпечне середовище розробки	Незначна можливість кастомізації розробки
Гнучкі правила публікування	Менш строгий контроль якості	Строгий контроль якості	Жорсткі правила модерації

Продовження таблиці 4.1

Низька комісія	Випадки шахрайства	Безпечне середовище завантаження застосунків	Висока комісія

Конкурентні переваги розробленої соціальної мережі для локального бізнесу в сфері SMM полягають у ціновій політиці, що на початковому етапі запуску є не комерційною. Цей підхід створює значну перевагу над конкурентами, сприяючи залученню нових користувачів.

Зважаючи на вищеподані інформацію, соціальна мережа для локального бізнесу у сфері SMM є необхідним посередником між SMM-спеціалістами та їх потенційними клієнтами, що дозволяє ефективно здійснювати комунікацію та залучати цільову аудиторію. Використання спеціалізованих програмних рішень для управління соціальними мережами дозволяє значно економити час, що є важливим ресурсом для збільшення прибутковості обох сторін. Крім того, підтримка сторінок у соціальних мережах з красивим оформленням сприяє покращенню привабливості продуктів та послуг для потенційних клієнтів.

4.2 Розрахунок витрат на проєктування

Ретельний розрахунок витрат на проєктування є критичним етапом у будь-якому проєкті, особливо в контексті створення соціальної мережі для локального бізнесу в сфері SMM. Ретельний аналіз витрат дозволяє не лише точно оцінити загальну вартість проекту, а й забезпечити фінансову стійкість і успішне впровадження соціальної мережі обраній сфері.

Загальний кошторис створення соціальної мережі для локального бізнесу в сфері SMM становить 3 177 649 грн. Детальний розрахунок зображено у додатку М.

Важливою частиною розробки кошторису був розрахунок заробітної плати проєктувальників згідно законодавства України [14]. В Україні відрахування із заробітної плати включають Єдиний соціальний внесок (ЄСВ), податок на доходи фізичних осіб (ПДФО) та військовий збір. Детальна інформація про кожен з цих податків наступна:

– єдиний соціальний внесок (ЄСВ) (ставка: 22% від заробітної плати, формула: ЄСВ = Оклад × 22%);

– податок на доходи фізичних осіб (ПДФО) (ставка: 18% від заробітної плати після вирахування ЄСВ, формула: ПДФО = (Оклад - ЄСВ) × 18%);

– військовий збір (ставка: 1.5% від заробітної плати після вирахування ЄСВ, формула: Військовий збір = (оклад - ЄСВ) × 1.5%)

Тепер необхідно розрахувати відрахування для кожної посади. Для керівника проєкту відрахунок наступний:

– оклад: 79000 грн/міс;

– ЄСВ: 79000 × 22% = 17380 грн/міс;

– ПДФО: (79000 - 17380) × 18% = 11037.6 грн/міс;

– військовий збір: (79000 - 17380) × 1.5% = 919.8 грн/міс;

– відрахування всього: 17380 + 11037.6 + 919.8 = 29337.4 грн/міс.

Для програміста-розробника відрахунок наступний:

– оклад: 98000 грн/міс;

– ЄСВ: 98000 × 22% = 21560 грн/міс;

– ПДФО: (98000 - 21560) × 18% = 13771.2 грн/міс;

– військовий збір: (98000 - 21560) × 1.5% = 1147.8 грн/міс;

– відрахування всього: 21560 + 13771.2 + 1147.8 = 36479 грн/міс.

Для тестувальника відрахунок буде обчислюватися наступним чином:

– оклад: 71000 грн/міс;

– ЄСВ: 71000 × 22% = 15620 грн/міс;

– ПДФО: (71000 - 15620) × 18% = 9952.8 грн/міс;

– військовий збір: (71000 - 15620) × 1.5% = 829.2 грн/міс;

– відрахування всього: 15620 + 9952.8 + 829.2 = 26302 грн/міс.

Для архітектора програмного забезпечення відрахунок буде обчислюватися так:

– оклад: 50800 грн/міс;

– ЄСВ: 50800 × 22% = 11176 грн/міс;

– ПДФО: (50800 - 11176) × 18% = 7112.64 грн/міс;

– військовий збір: (50800 - 11176) × 1.5% = 592.08 грн/міс;

– відрахування всього: 11176 + 7112.64 + 592.08 = 18880.72 грн/міс.

Для UI/UX дизайнера відрахунок наступний:

– оклад: 64000 грн/міс;

– ЄСВ: 64000 × 22% = 14080 грн/міс;

– ПДФО: (64000 - 14080) × 18% = 8996.4 грн/міс;

– військовий збір: (64000 - 14080) × 1.5% = 749.6 грн/міс;

– відрахування всього: 14080 + 8996.4 + 749.6 = 23826 грн/міс.

Для адміністратора бази даних відрахунок буде обчислюватися таким чином:

– оклад: 39000 грн/міс;

– ЄСВ: 39000 × 22% = 8580 грн/міс;

– ПДФО: (39000 - 8580) × 18% = 5474.4 грн/міс;

– військовий збір: (39000 - 8580) × 1.5% = 456.3 грн/міс;

– відрахування всього: 8580 + 5474.4 + 456.3 = 14510.7 грн/міс.

Заробітна плата проектувальників була визначена відповідно до чинного законодавства та враховує всі необхідні податкові відрахування. Це забезпечує фінансову стійкість проекту та сприяє його успішному впровадженню в сфері SMM для локального бізнесу. Детальний розпис кошторису заробітної плати, що знаходиться у додатку Н, підкріплює обґрунтованість розрахунків.

Таким чином, детальний розрахунок підтверджує прозорість і обґрунтованість витрат на проектування соціальної мережі, що є ключовим етапом у формуванні загального кошторису проекту.

4.3 Обґрунтування доцільності розробки

Розроблена соціальна мережа для місцевого бізнесу в сфері SMM покликана задовольнити потреби бізнесу, що звертається за допомогою до SMM-фахівців. Ця платформа є інструментом, який дозволяє бізнесам значно збільшити свої продажі за рахунок покращення видимості у соціальних мережах та залучення нових клієнтів. Вона забезпечує ефективне спілкування з аудиторією, пропонуючи можливість взаємодіяти з клієнтами, відповідати на їхні запити та отримувати зворотний зв'язок у режимі реального часу. Така комунікація сприяє підвищенню задоволеності клієнтів, що, в свою чергу, позитивно впливає на лояльність до бренду та збільшення повторних покупок.

Стратегічна важливість цієї платформи полягає в її здатності допомогти бізнесу залишатися конкурентоспроможним в умовах швидких змін на ринку. Соціальні мережі є потужним інструментом для маркетингу, і можливість ефективно використовувати їх є ключовою для досягнення успіху. Платформа дозволяє бізнесу знаходити SMM-спеціалістів, які допомагатимуть адаптуватися до нових тенденцій та вимог ринку.

Загалом, розробка соціальної мережі для місцевого бізнесу в сфері SMM є важливим стратегічним рішенням. Вона не тільки сприяє збільшенню продажів та покращенню взаємодії з клієнтами, але й допомагає бізнесу адаптуватися до цифрового середовища та використовувати новітні технології для досягнення своїх цілей. Це забезпечує стійке зростання та розвиток бізнесу, підвищуючи його конкурентоспроможність та забезпечуючи довготривалий успіх на ринку.

ВИСНОВКИ

Кваліфікаційна робота спрямована на створення соціальної мережі, спеціалізованої на потреби локального бізнесу в сфері SMM. Метою дослідження було розроблення додатку під назвою «КОМЕНТ», який може покращувати комунікацію між SMM-спеціалістами та їх клієнтами.

Процес виконання проєкту включав такі етапи: встановлення основного завдання додатку – ефективне залучення потенційних клієнтів бізнесу через соціальні мережі за допомогою SMM-спеціалістів, розробка кросплатформленого додатку для таких операційних систем, як Android ті IOS і забезпеченя інтерфейсу, що максимально враховує специфіку соціальних мереж, мануальне тестування прототипу та вдосконалення функціоналу застосунку на основі тест-кейсів.

Результати аналізу предметної області та постановки завдання програмного продукту відповідно вимог та потреб замовника висвітлювались на онлайн засіданні обласного методичного об’єднання викладачів дисциплін «Інформаційні системи та технології» на тему: «Актуальні аспекти автоматизації бізнес процесів».

Результати дослідження та розробки соціальної мережі для локального бізнесу в сфері SMM показали значний потенціал для підвищення ефективності маркетингу в соціальних мережах та можуть бути вдосконалені наступним чином: розширення функціоналу для інтеграції з іншими соціальними мережами, додавання календаря, який дозволить відмічати дедлайни проєктів, інтеграція сервісів, які надаватимуть доступ до відео та аудіо дзвінків для користувачів соціальної мережі.

Ця кваліфікаційна робота стала важливим кроком у напрямку покращення соціальної взаємодії між SMM-спеціалістами та бізнесом. Результати цієї кваліфікаційної роботи можуть бути використані у розвитку та підтримці малого та середнього бізнесу, полегшенні їх комунікації та наданні нових можливостей їх взаємодії.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. LaFleur G. What is social media marketing (SMM)?. WhatIs. URL: https://www.techtarget.com/whatis/definition/social-media-marketing-SMM (дата звернення: 09.01.2024).

2. Petrosyan A. Internet and social media users in the world 2024 | Statista. Statista. URL: https://www.statista.com/statistics/617136/digital-population-worldwide/ (дата звернення: 10.02.2024).

3. Lin Y. Global ecommerce sales growth report (2024). Shopify. URL: https://www.shopify.com/blog/global-ecommerce-sales (дата звернення: 03.02.2024).

4. UkrOpen – соцмережа. Українопедія. URL: https://ukrainopedia.fandom.com/wiki/UkrOpen_–_соцмережа (дата звернення: 11.01.2024).

5. What is Use Case Diagram?. Ideal Modeling & Diagramming Tool for Agile Team Collaboration. URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/ (дата звернення: 22.02.2024).

6. Khan S. Flutter – clean architecture. Medium. URL: https://medium.com/@samra.sajjad0001/flutter-clean-architecture-5de5e9b8d093 (дата звернення: 01.03.2024).

7. Devina F. Flutter TDD clean architecture. Medium. URL: https://medium.com/@fakhiradevina/flutter-tdd-clean-architecture-272373727699 (дата звернення: 24.02.2024).

8. Scheidmeir A. App State-Management mit Flutter BLoC - exensio GmbH. Individuelle Softwarelösungen und IT-Beratung - exensio GmbH. URL: https://www.exensio.de/news-medien/newsreader-blog/app-state-management-mit-flutter-bloc (дата звернення: 13.03.2024).

9. Polo A. Flutter bloc for beginners. Medium. URL: https://medium.com/flutter-community/flutter-bloc-for-beginners-839e22adb9f5 (дата звернення: 03.03.2024).

10. Ayaz M. Flutter clean architecture with bloc: the ultimate guide. eTechViral. URL: https://etechviral.com/flutter-clean-architecture-with-bloc/ (дата звернення: 06.03.2024).

11. Firebase | google’s mobile and web app development platform. Firebase. URL: https://firebase.google.com/ (дата звернення: 02.04.2024).

12. Microsoft. Visual studio code. Версія 1.90. Редмонд : Microsoft, 2024. URL: https://code.visualstudio.com/updates/v1_90#_accessibility (дата звернення: 20.03.2024).

13. Android. Android studio. Jellyfish. Версія 8.4.2. Пало-Альто : Android, 2023. URL: https://developer.android.com/studio (дата звернення: 29.02.2024).

14. "Про оплату праці" : Закон України від 09.11.2023 р. № 3460-IX : станом на 1 квіт. 2024 р.

15. Ковальчук Я. Соціальні мережі як інструмент автоматизації маркетингу локального бізнесу. Актуальні аспекти автоматизації бізнес процесів : МАТЕРІАЛИ СТУДЕНТ. НАУКОВО-ПРАКТ. ON-LINE КОНФ., м. Заліщики, 18 квіт. 2024 р. Заліщики, 2024. С. 67–70. URL: https://collegeradate.wordpress.com/wp-content/uploads/2024/06/d097d0b1d196d180d0bdd0b8d0ba-2024-d086d0a1d0a2-d0b2-d0b1d196d0b7d0bdd0b5d181d196.pdf (дата звернення: 04.06.2024).

ДОДАТКИ

Додаток А

Лістинг файлу mobile_screen_layout.dart

```
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:flutter_svg/flutter_svg.dart';
class MobileScreenLayout extends StatefulWidget {
  final String userId;
  const MobileScreenLayout(this.userId, {Key? key}) :
super(key: key);

  @override
  State<MobileScreenLayout> createState() =>
_MobileScreenLayoutState();
}

class _MobileScreenLayoutState extends
State<MobileScreenLayout> {
  ValueNotifier<bool> playHomeVideo = ValueNotifier(false);
  ValueNotifier<bool> playMainReelVideos =
ValueNotifier(false);
  CupertinoTabController controller = CupertinoTabController();

  @override
  Widget build(BuildContext context) {
    return ValueListenableBuilder(
      valueListenable: playMainReelVideos,
      builder: (BuildContext context, bool value, __) {
        return CupertinoTabScaffold(
            tabBar: CupertinoTabBar(
                backgroundColor:
```

```
                    value ? ColorManager.black :
Theme.of(context).primaryColor,
                height: 45,
                border: Border.all(color:
ColorManager.transparent),
                items: [
                  navigationBarItem(IconsAssets.home, value),
                  navigationBarItem(IconsAssets.search, value),
                  navigationBarItem(IconsAssets.video, value,
smallIcon: true),
                  navigationBarItem(IconsAssets.shop, value),
                  personalImageItem(),
                ]),
            controller: controller,
            tabBuilder: (context, index) {
              WidgetsBinding.instance.addPostFrameCallback((_)
{
                playMainReelVideos.value = controller.index ==
2 ? true : false;
                playHomeVideo.value = controller.index == 0 ?
true : false;
              });

              switch (index) {
                case 0:
                  return homePage();
                case 1:
                  return allUsersTimLinePage();
                case 2:
                  return videoPage(value);
                case 3:
                  return shopPage();
                default:
                  return personalProfilePage();
```

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

    CupertinoTabView allUsersTimLinePage() => CupertinoTabView(
          builder: (context) =>
              CupertinoPageScaffold(child:
AllUsersTimeLinePage()),
        );

    CupertinoTabView shopPage() => CupertinoTabView(
          builder: (context) => const CupertinoPageScaffold(
            child: ShopPage(),
          ),
        );

    CupertinoTabView personalProfilePage() => CupertinoTabView(
          builder: (context) => CupertinoPageScaffold(
            child: BlocProvider<PostCubit>(
              create: (context) => injector<PostCubit>(),
              child: PersonalProfilePage(personalId:
widget.userId),
            ),
          ),
        );

    Widget videoPage(bool value) => CupertinoTabView(
        builder: (context) => CupertinoPageScaffold(
              child: BlocProvider<PostCubit>(
                create: (context) => injector<PostCubit>(),
                child: VideosPage(stopVideo: playMainReelVideos),
              ),
```

```
          ));

  Widget homePage() => CupertinoTabView(
        builder: (context) => CupertinoPageScaffold(
            child: BlocProvider<PostCubit>(
          create: (context) => injector<PostCubit>(),
          child: ValueListenableBuilder(
            valueListenable: playHomeVideo,
            builder: (context, bool playVideoValue, child) =>
HomePage(
              userId: widget.userId,
              playVideo: playVideoValue,
            ),
          ),
        )),
      );

  BottomNavigationBarItem personalImageItem() =>
      const BottomNavigationBarItem(icon: PersonalImageIcon());
  BottomNavigationBarItem navigationBarItem(String icon, bool
value,
      {bool smallIcon = false}) {
    return BottomNavigationBarItem(
      icon: SvgPicture.asset(
        icon,
        height: smallIcon ? 23 : 25,
        colorFilter: ColorFilter.mode(
            value ? ColorManager.white :
Theme.of(context).focusColor,
            BlendMode.srcIn),
      ),
    );
  }
}
```

Додаток Б

Лістинг файлу sign_up_page.dart

```
import '../../../core/functions/toast_show.dart';
import
'../../../data/models/parent_classes/without_sub_classes/user_p
ersonal_info.dart';
import
'../../cubit/firebaseAuthCubit/firebase_auth_cubit.dart';
import
'../../cubit/firestoreUserInfoCubit/add_new_user_cubit.dart';
class SignUpPage extends StatefulWidget {
  const SignUpPage({Key? key}) : super(key: key);
  @override
  State<SignUpPage> createState() => _SignUpPageState();
}
class _SignUpPageState extends State<SignUpPage> {
  final emailController = TextEditingController();
  final passwordController = TextEditingController();
  final fullNameController = TextEditingController();
  final bool validateControllers = false;
  ValueNotifier<bool> validateEmail = ValueNotifier(false);
  ValueNotifier<bool> validatePassword = ValueNotifier(false);
  ValueNotifier<bool> rememberPassword = ValueNotifier(false);
  @override
  Widget build(BuildContext context) {
    return RegisterWidgets(
      fullNameController: fullNameController,
      customTextButton: customTextButton(),
      emailController: emailController,
      passwordController: passwordController,
      isThatLogIn: false,
      validateEmail: validateEmail,
      validatePassword: validatePassword,
```

```
      rememberPassword: rememberPassword,
    );
  }
  Widget customTextButton() {
    return ValueListenableBuilder(
      valueListenable: rememberPassword,
      builder: (context, bool rememberPasswordValue, child) =>
          ValueListenableBuilder(
            valueListenable: validateEmail,
            builder: (context, bool validateEmailValue, child)
=>
                ValueListenableBuilder(
                  valueListenable: validatePassword,
                  builder: (context, bool
validatePasswordValue, child) {
                    bool validate = validatePasswordValue &&
                        validateEmailValue &&
                        rememberPasswordValue &&
                        fullNameController.text.isNotEmpty;
                    return CustomElevatedButton(
                      isItDone: true,
                      isThatSignIn: true,
                      nameOfButton: StringsManager.next.tr,
                      blueColor: validate ? true : false,
                      onPressed: () async {
                        if (validate) {
                          Get.to(
                              UserNamePage(
                                emailController:
emailController,
                                passwordController:
passwordController,
                                fullNameController:
fullNameController,
```

```
                                ),
                                duration: const Duration(seconds:
0));
                          }
                        },
                      );
                    },
                  ),
            ),
      );
    }
}
class UserNamePage extends StatefulWidget {
  final TextEditingController emailController;
  final TextEditingController passwordController;
  final TextEditingController fullNameController;
  const UserNamePage({
    Key? key,
    required this.emailController,
    required this.passwordController,
    required this.fullNameController,
  }) : super(key: key);

  @override
  State<UserNamePage> createState() => _UserNamePageState();
}
class _UserNamePageState extends State<UserNamePage> {
  final userNameController = TextEditingController();

  bool isToastShowed = false;

  bool validateEdits = false;

  bool isFieldEmpty = true;
```

```
  bool isHeMovedToHome = false;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: SafeArea(
        child: Center(
          child: isThatMobile
              ? SizedBox(
            height: MediaQuery.of(context).size.height,
            child: buildColumn(context),
          )
              : buildForWeb(context),
        ),
      ),
    );
  }
  Widget buildColumn(BuildContext context) {
    return Column(
      crossAxisAlignment: CrossAxisAlignment.center,
      mainAxisAlignment: MainAxisAlignment.start,
      children: [
        const SizedBox(height: 100),
        Text(
          StringsManager.createUserName.tr,
          style:
          getMediumStyle(color: Theme.of(context).focusColor,
fontSize: 15),
        ),
        const SizedBox(height: 10),
        Center(
          child: Text(
            StringsManager.addUserName.tr,
```

```
            style: getNormalStyle(color: ColorManager.grey,
fontSize: 13),
          ),
        ),
        Text(
          StringsManager.youCanChangeUserNameLater.tr,
          style: getNormalStyle(color: ColorManager.grey,
fontSize: 13),
        ),
        const SizedBox(height: 30),
        userNameTextField(context),
        customTextButton(),
      ],
    );
  }

  SizedBox buildForWeb(BuildContext context) {
    return SizedBox(
      width: 352,
      child: Column(
        mainAxisSize: MainAxisSize.min,
        children: [
          Container(
            width: double.infinity,
            height: 400,
            decoration: BoxDecoration(
              color: Colors.white,
              border: Border.all(color: Colors.grey, width:
0.2),
            ),
            child: buildColumn(context),
          ),
        ],
      ),
```

```
    );
  }

  Widget userNameTextField(BuildContext context) {
    return BlocBuilder<SearchAboutUserBloc,
SearchAboutUserState>(
      bloc: BlocProvider.of<SearchAboutUserBloc>(context)
        ..add(FindSpecificUser(userNameController.text,
            searchForSingleLetter: true)),
      buildWhen: (previous, current) =>
      previous != current && current is
SearchAboutUserBlocLoaded,
      builder: (context, state) {
        List<UserPersonalInfo> usersWithSameUserName = [];

        if (state is SearchAboutUserBlocLoaded) {
          usersWithSameUserName = state.users;
        }
        WidgetsBinding.instance.addPostFrameCallback((_) =>
setState(() {
          validateEdits = usersWithSameUserName.isEmpty;
          if (userNameController.text.isEmpty) {
            validateEdits = false;
            isFieldEmpty = true;
          } else {
            isFieldEmpty = false;
          }
        }));
        return customTextField(context);
      },
    );
  }

  Padding customTextField(BuildContext context) {
```

```
    return Padding(
      padding: const EdgeInsetsDirectional.only(start: 20, end:
20),
      child: SizedBox(
        height: isThatMobile ? null : 37,
        width: double.infinity,
        child: TextField(
          controller: userNameController,
          cursorColor: ColorManager.teal,
          style:
          getNormalStyle(color: Theme.of(context).focusColor,
fontSize: 15),
          decoration: InputDecoration(
            hintText: StringsManager.username.tr,
            hintStyle: isThatMobile
                ? getNormalStyle(color:
Theme.of(context).indicatorColor)
                : getNormalStyle(color: ColorManager.black54,
fontSize: 12),
            fillColor: const Color.fromARGB(48, 232, 232, 232),
            filled: true,
            focusedBorder: outlineInputBorder(),
            suffixIcon: isFieldEmpty
                ? null
                : (validateEdits ? rightIcon() : wrongIcon()),
            enabledBorder: outlineInputBorder(),
            contentPadding: EdgeInsets.symmetric(
                horizontal: 10, vertical: isThatMobile ? 15 :
5),
            errorText: (isFieldEmpty || validateEdits)
                ? null
                : (isThatMobile ?
StringsManager.thisUserNameExist.tr : null),
```

```
            errorStyle: getNormalStyle(color:
ColorManager.red),
          ),
          onChanged: (value) {
            SearchAboutUserBloc.get(context).add(FindSpecificUs
er(
                userNameController.text,
                searchForSingleLetter: true));
          },
        ),
      ),
    );
  }

  Icon rightIcon() {
    return const Icon(Icons.check_rounded, color:
ColorManager.green, size: 27);
  }

  Widget wrongIcon() {
    return const Icon(
      Icons.close_rounded,
      color: ColorManager.red,
      size: 27,
    );
  }

  OutlineInputBorder outlineInputBorder() {
    return OutlineInputBorder(
      borderRadius: BorderRadius.circular(isThatMobile ? 5.0 :
1.0),
      borderSide: BorderSide(
          color: ColorManager.lightGrey, width: isThatMobile ?
1.0 : 0.8),
```

```
    );
  }

  Widget customTextButton() {
    return Builder(builder: (context) {
      FireStoreAddNewUserCubit userCubit =
      FireStoreAddNewUserCubit.get(context);
      return BlocConsumer<FirebaseAuthCubit,
FirebaseAuthCubitState>(
        listenWhen: (previous, current) => previous != current,
        listener: (context, state) {
          if (state is CubitAuthConfirmed) {
            addNewUser(state, userCubit);
            moveToMain(state);
          } else if (state is CubitAuthFailed &&
!isToastShowed) {
            ToastShow.toastStateError(state.error);
          }
        },
        buildWhen: (previous, current) => previous != current,
        builder: (context, authState) {
          return CustomElevatedButton(
            isItDone: authState is! CubitAuthConfirming,
            nameOfButton: StringsManager.signUp.tr,
            blueColor: validateEdits,
            onPressed: () async {
              FirebaseAuthCubit authCubit =
FirebaseAuthCubit.get(context);

              if (validateEdits) {
                setState(() => isToastShowed = false);

                await authCubit.signUp(RegisteredUser(
                  email: widget.emailController.text,
```

```
                    password: widget.passwordController.text,
                  ));
                }
              },
            );
          },
        );
      });
  }

  moveToMain(CubitAuthConfirmed authState) async {
    myPersonalId = authState.user.uid;

    final SharedPreferences sharePrefs =
injector<SharedPreferences>();
    if (!isHeMovedToHome) {
      setState(() => isHeMovedToHome = true);

      if (myPersonalId.isNotEmpty) {
        await sharePrefs.setString("myPersonalId",
myPersonalId);
        Get.offAll(GetMyPersonalInfo(myPersonalId:
myPersonalId));
      } else {
        ToastShow.toast(StringsManager.somethingWrong.tr);
      }
    }
  }

  addNewUser(CubitAuthConfirmed authState,
FireStoreAddNewUserCubit userCubit) {
    String fullName = widget.fullNameController.text;
    List<dynamic> charactersOfName = [];
    String nameOfLower = fullName.toLowerCase();
```

```
    for (int i = 0; i < nameOfLower.length; i++) {
      charactersOfName = charactersOfName +
[nameOfLower.substring(0, i + 1)];
    }
    String userName = userNameController.text;
    UserPersonalInfo newUserInfo = UserPersonalInfo(
      name: fullName,
      charactersOfName: charactersOfName,
      email: authState.user.email!,
      userName: userName,
      bio: "",
      profileImageUrl: "",
      userId: authState.user.uid,
      followerPeople: const [],
      followedPeople: const [],
      posts: const [],
      chatsOfGroups: const [],
      stories: const [],
      lastThreePostUrls: const [],
    );
    userCubit.addNewUser(newUserInfo);
  }
}
```

Додаток В

Лістинг файлу home_page.dart

```
import 'dart:async';

import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:get/get.dart';
import 'package:image_picker_plus/image_picker_plus.dart';

class HomePage extends StatefulWidget {
  final String userId;
  final bool playVideo;

  const HomePage({Key? key, required this.userId,
this.playVideo = true})
      : super(key: key);

  @override
  State<HomePage> createState() => _HomePageState();
}

class _HomePageState extends State<HomePage> with
TickerProviderStateMixin {
  ValueNotifier<bool> isThatEndOfList = ValueNotifier(false);
  late UserPersonalInfo personalInfo;
  ValueNotifier<bool> reLoadData = ValueNotifier(false);
  Post? selectedPostInfo;
  bool rebuild = true;
  List postsIds = [];
  ValueNotifier<List<Post>> postsInfo = ValueNotifier([]);
  List<UserPersonalInfo>? storiesOwnersInfo;

  Future<void> _getData(int index) async {
```

```
    storiesOwnersInfo = null;
    reLoadData.value = false;
    UserInfoCubit userCubit =
        BlocProvider.of<UserInfoCubit>(context, listen: false);
    await userCubit.getUserInfo(widget.userId);
    if (!mounted) return;
    personalInfo = userCubit.myPersonalInfo;
    List usersIds = personalInfo.followedPeople;
    SpecificUsersPostsCubit usersPostsCubit =
        BlocProvider.of<SpecificUsersPostsCubit>(context,
listen: false);

    await usersPostsCubit.getSpecificUsersPostsInfo(usersIds:
usersIds);

    List usersPostsIds = usersPostsCubit.usersPostsInfo;

    postsIds = personalInfo.posts + usersPostsIds;
    if (!mounted) return;
    PostCubit postCubit = PostCubit.get(context);
    await postCubit
        .getPostsInfo(
            postsIds: postsIds, isThatMyPosts: true,
lengthOfCurrentList: index)
        .then((value) {
      reLoadData.value = true;
    });
  }

  @override
  void initState() {
    _getData(0);

    if (isThatMobile) {
```

```
      WidgetsBinding.instance.addPostFrameCallback(
          (_) async => await notificationPermissions(context));
    }
    super.initState();
  }

  @override
  void dispose() {
    isThatEndOfList.dispose();
    reLoadData.dispose();
    postsInfo.dispose();

    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    return SafeArea(
      child: Scaffold(
        appBar: isThatMobile ?
CustomAppBar.basicAppBar(context) : null,
        body: Center(
          child: blocBuilder(),
        ),
      ),
    );
  }

  ValueListenableBuilder<bool> blocBuilder() {
    return ValueListenableBuilder(
      valueListenable: reLoadData,
      builder: (context, bool value, child) =>
          BlocBuilder<PostCubit, PostState>(
        buildWhen: (previous, current) {
```

```
            if (value && current is CubitMyPersonalPostsLoaded) {
              reLoadData.value = false;
              return true;
            }
            if (value) {
              reLoadData.value = false;
              return true;
            }

            if (previous != current && current is
CubitMyPersonalPostsLoaded) {
              return true;
            }
            if (previous != current && current is
CubitPostFailed) {
              return true;
            }
            return false;
          },
          builder: (BuildContext context, PostState state) {
            if (state is CubitMyPersonalPostsLoaded) {
              postsInfo.value = state.postsInfo;
              return postsInfo.value.isNotEmpty
                  ? inViewNotifier()
                  : WelcomeCards(onRefreshData: _getData);
            } else if (state is CubitPostFailed) {
              ToastShow.toastStateError(state);
              return Center(
                  child: Text(
                StringsManager.somethingWrong.tr,
                style: getNormalStyle(color:
Theme.of(context).focusColor),
              ));
            } else {
```

```
            return const ThineCircularProgress();
          }
        },
      ),
    );
  }

  Widget inViewNotifier() {
    return ValueListenableBuilder(
      valueListenable: postsInfo,
      builder: (context, List<Post> postsInfoValue, child) =>
          InViewNotifierList(
        onRefreshData: _getData,
        postsIds: postsIds,
        physics: const BouncingScrollPhysics(),
        isThatEndOfList: isThatEndOfList,
        initialInViewIds: const ['0'],
        isInViewPortCondition:
            (double deltaTop, double deltaBottom, double
vpHeight) {
          return deltaTop < (0.5 * vpHeight) && deltaBottom >
(0.5 * vpHeight);
        },
        itemCount: postsInfoValue.length,
        builder: (BuildContext context, int index) {
          return Center(
            child: Container(
              width: isThatMobile ? double.infinity : 450,
              margin: const EdgeInsetsDirectional.only(bottom:
.5, top: .5),
              child: LayoutBuilder(
                builder: (BuildContext context, BoxConstraints
constraints) {
                  return InViewNotifierWidget(
```

```
                    id: '$index',
                    builder: (_, bool isInView, __) {
                      bool checkForPlatform = isThatMobile
                          ? isInView && widget.playVideo
                          : isInView;
                      return columnOfWidgets(index,
checkForPlatform, isInView);
                    },
                  );
                },
              ),
            ),
          );
        },
      ),
    );
  }

  Widget columnOfWidgets(int index, bool playTheVideo, bool
isInView) {
    double storiesHeight = isThatMobile ? 672 : 500;
    return Column(
      mainAxisSize: MainAxisSize.min,
      children: [
        if (index == 0) ...[
          storiesOwnersInfo != null
              ? buildUsersStories(context)
              : storiesLines(storiesHeight),
          if (isThatMobile) customDivider(),
        ] else ...[
          if (isThatMobile) divider(),
        ],
        posts(index, playTheVideo),
```

```
        if (isThatEndOfList.value && index == postsIds.length -
1) ...[
          if (isThatMobile) divider(),
          const AllCatchUpIcon(),
        ]
      ],
    );
  }

  Divider divider() {
    return const Divider(color: ColorManager.lightGrey,
thickness: .15);
  }

  Container customDivider() => Container(
      margin: const EdgeInsetsDirectional.only(bottom: 8, top:
5),
      color: ColorManager.grey,
      width: double.infinity,
      height: 0.3);

  Widget posts(int index, bool playTheVideo) {
    Widget buildPost = ValueListenableBuilder(
      valueListenable: postsInfo,
      builder: (context, List<Post> postsInfoValue, child) =>
PostOfTimeLine(
        postInfo: ValueNotifier(postsInfo.value[index]),
        postsInfo: postsInfo,
        playTheVideo: playTheVideo,
        indexOfPost: index,
        reLoadData: reloadTheData,
        removeThisPost: removeThisPost,
      ),
    );
```

```
    return isThatMobile
        ? buildPost
        : _RoundedContainer(
            internalPadding: false,
            verticalPadding: true,
            child: buildPost,
          );
  }

  void removeThisPost(int index) {
    setState(() {
      postsIds.removeAt(index);
      postsInfo.value.removeAt(index);
      reLoadData.value = true;
    });
  }

  reloadTheData() => reLoadData.value = true;

  Widget buildUsersStories(BuildContext context) {
    Widget stories = _BuildStoriesLine(
        personalInfo: personalInfo,
        reLoadData: reLoadData,
        storiesOwnersInfo: storiesOwnersInfo,
        scrollController: ScrollController());
    return isThatMobile
        ? stories
        : _RoundedContainer(isThatStory: true, child: stories);
  }

  Widget storiesLines(double bodyHeight) {
    List<dynamic> usersStoriesIds =
        personalInfo.followedPeople +
personalInfo.followerPeople;
```

```
    return ValueListenableBuilder(
      valueListenable: reLoadData,
      builder: (context, bool value, child) =>
          BlocBuilder<StoryCubit, StoryState>(
        bloc: StoryCubit.get(context)
          ..getStoriesInfo(
              usersIds: usersStoriesIds, myPersonalInfo:
personalInfo),
        buildWhen: (previous, current) {
          if (value && current is CubitStoriesInfoLoaded) {
            reLoadData.value = false;
            return true;
          }
          if (value) {
            reLoadData.value = false;
            return true;
          }
          if (previous != current && current is
CubitStoriesInfoLoaded) {
            return true;
          }
          if (previous != current && current is
CubitStoryFailed) {
            return true;
          }
          return false;
        },
        builder: (context, state) {
          if (state is CubitStoriesInfoLoaded) {
            storiesOwnersInfo = state.storiesOwnersInfo;
            return buildUsersStories(context);
          } else if (state is CubitStoryFailed) {
            ToastShow.toastStateError(state);
            return Center(
```

```
                child: Text(
              StringsManager.somethingWrong.tr,
              style: getNormalStyle(color:
Theme.of(context).focusColor),
            ));
          } else {
            return const SizedBox();
          }
        },
      ),
    );
  }
}

class _RoundedContainer extends StatelessWidget {
  const _RoundedContainer({
    Key? key,
    required this.child,
    this.internalPadding = true,
    this.verticalPadding = false,
    this.isThatStory = false,
  }) : super(key: key);
  final Widget child;
  final bool internalPadding;
  final bool verticalPadding;
  final bool isThatStory;
  @override
  Widget build(BuildContext context) {
    return Padding(
      padding: const EdgeInsets.only(top: 15.0),
      child: Container(
        padding: internalPadding || verticalPadding
            ? const EdgeInsets.symmetric(vertical: 15)
            : null,
```

```
        decoration: BoxDecoration(
          color: ColorManager.white,
          borderRadius: BorderRadius.circular(10),
          border: Border.all(color:
ColorManager.lowOpacityGrey, width: 1),
        ),
        child: child,
      ),
    );
  }
}

class _BuildStoriesLine extends StatelessWidget {
  const _BuildStoriesLine({
    Key? key,
    required this.personalInfo,
    required this.reLoadData,
    required this.storiesOwnersInfo,
    required this.scrollController,
  }) : super(key: key);

  final UserPersonalInfo personalInfo;
  final ValueNotifier<bool> reLoadData;
  final ScrollController scrollController;
  final List<UserPersonalInfo>? storiesOwnersInfo;

  @override
  Widget build(BuildContext context) {
    final mediaQuery = MediaQuery.of(context);
    final bodyHeight = mediaQuery.size.height -
        AppBar().preferredSize.height -
        mediaQuery.padding.top;
    final storiesLength = storiesOwnersInfo?.length ?? 0;
    return Padding(
```

```
      padding: const EdgeInsetsDirectional.only(start: 10),
      child: SizedBox(
        width: double.infinity,
        height: 600 * 0.153,
        child: Stack(
          children: [
            CustomScrollView(
              scrollDirection: Axis.horizontal,
              controller: scrollController,
              slivers: [
                if (personalInfo.stories.isEmpty) ...[
                  SliverPadding(
                    padding: const
EdgeInsetsDirectional.only(end: 12),
                    sliver: SliverToBoxAdapter(
                        child: _MyOwnStory(
                            reLoadData: reLoadData,
                            personalInfo: personalInfo)),
                  ),
                ],
                SliverList(
                    delegate:
SliverChildBuilderDelegate((context, index) {
                  UserPersonalInfo publisherInfo =
storiesOwnersInfo![index];
                  String hashTag = isThatMobile
                      ? "${publisherInfo.userId.hashCode} for
mobile"
                      : "${publisherInfo.userId.hashCode} for
web";
                  return Hero(
                    tag: hashTag,
                    child: Padding(
                      padding: EdgeInsetsDirectional.only(
```

```
                            end: index != storiesLength - 1 ? 12
: 0),
                        child: GestureDetector(
                          onTap: () {
                            if (isThatMobile) {
                              Widget page = StoryPageForMobile(
                                  user: publisherInfo,
                                  hashTag: hashTag,
                                  storiesOwnersInfo:
storiesOwnersInfo!);
                              Go(context)
                                  .push(page: page,
withoutPageTransition: true);
                            } else {
                              Widget page = StoryPageForWeb(
                                  user: publisherInfo,
                                  hashTag: hashTag,
                                  storiesOwnersInfo:
storiesOwnersInfo!);
                              Get.to(page);
                            }
                          },
                          child: CircleAvatarOfProfileImage(
                            userInfo: publisherInfo,
                            bodyHeight:
                                isThatMobile ? bodyHeight :
bodyHeight * 0.69,
                            thisForStoriesLine: true,
                            nameOfCircle: index == 0 &&
                                    publisherInfo.userId ==
personalInfo.userId
                                ? StringsManager.yourStory.tr
                                : "",
                          ),
```

```
                    ),
                  ),
                );
              }, childCount: storiesLength))
            ],
          ),
          if (!isThatMobile && (storiesOwnersInfo?.length ??
0) > 5) ...[
            Padding(
              padding: const EdgeInsets.only(bottom: 15),
              child: GestureDetector(
                  onTap: () {
                    double pos = scrollController.offset -
500;
                    pos = pos < 0 ? 0 : pos;
                    scrollController.animateTo(pos,
                        duration: const
Duration(milliseconds: 500),
                        curve: Curves.easeInOutQuart);
                  },
                  child: const ArrowJump()),
            ),
            Padding(
              padding: const EdgeInsets.only(bottom: 15),
              child: GestureDetector(
                onTap: () {
                  double pos = scrollController.offset + 500;

                  scrollController.animateTo(pos,
                      duration: const Duration(milliseconds:
500),
                      curve: Curves.easeInOutQuart);
                },
                child: const ArrowJump(isThatBack: false),
```

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

class _MyOwnStory extends StatefulWidget {
  const _MyOwnStory({
    Key? key,
    required this.reLoadData,
    required this.personalInfo,
  }) : super(key: key);

  final ValueNotifier<bool> reLoadData;
  final UserPersonalInfo personalInfo;

  @override
  State<_MyOwnStory> createState() => _MyOwnStoryState();
}

class _MyOwnStoryState extends State<_MyOwnStory> {
  @override
  Widget build(BuildContext context) {
    return GestureDetector(
      onTap: () async {
        SelectedImagesDetails? details = await
CustomImagePickerPlus.pickImage(
          context,
          isThatStory: true,
        );
```

```
        if (!context.mounted) return;
        if (details == null) return;

        await Go(context).push(
          page: CreateStoryPage(storiesDetails: details),
        );
        widget.reLoadData.value = true;
      },
      child: _MyOwnStoryChild(personalInfo:
widget.personalInfo),
    );
  }
}

class _MyOwnStoryChild extends StatelessWidget {
  const _MyOwnStoryChild({Key? key, required
this.personalInfo})
      : super(key: key);

  final UserPersonalInfo personalInfo;

  @override
  Widget build(BuildContext context) {
    return Stack(
      alignment: Alignment.bottomRight,
      children: [
        CircleAvatarOfProfileImage(
          userInfo: personalInfo,
          bodyHeight: 700,
          moveTextMore: true,
          thisForStoriesLine: true,
          nameOfCircle: StringsManager.yourStory.tr,
        ),
        Positioned(
```

```
            top: 650 * .058,
            left: 650 * .058,
            right: 650 * .012,
            child: Container(
              decoration: BoxDecoration(
                  color: Theme.of(context).primaryColor, shape:
BoxShape.circle),
              padding: const EdgeInsets.all(2),
              child: CircleAvatar(
                radius: 15,
                backgroundColor: Theme.of(context).primaryColor,
                child: const CircleAvatar(
                  radius: 10,
                  backgroundColor: ColorManager.blue,
                  child: Icon(
                    Icons.add,
                    size: 14,
                  ),
                ),
              ),
            ),
          ),
        ],
      );
    }
}
```

Додаток Г

Лістинг файлу create_story.dart

```
import 'dart:io';
import 'dart:typed_data';
import 'package:get/get.dart';
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:flutter_svg/flutter_svg.dart';
import 'package:image_picker_plus/image_picker_plus.dart';
import 'package:shared_preferences/shared_preferences.dart';
import 'package:video_thumbnail/video_thumbnail.dart';

class CreateStoryPage extends StatefulWidget {
  final SelectedImagesDetails storiesDetails;

  const CreateStoryPage({Key? key, required
this.storiesDetails})
      : super(key: key);

  @override
  State<CreateStoryPage> createState() =>
_CreateStoryPageState();
}

class _CreateStoryPageState extends State<CreateStoryPage> {
  bool isItDone = true;
  late List<SelectedByte> selectedFiles;
  @override
  void initState() {
    selectedFiles = widget.storiesDetails.selectedFiles;
    super.initState();
  }
```

```
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Theme.of(context).primaryColor,
      body: SafeArea(
        child: Column(
          children: [
            Expanded(
              child: widget.storiesDetails.multiSelectionMode
                  ? CustomMultiImagesDisplay(selectedImages:
selectedFiles)
                  : Image.file(selectedFiles[0].selectedFile),
            ),
            Container(
              decoration: BoxDecoration(
                color: Theme.of(context).primaryColor,
                borderRadius:
                    const BorderRadius.vertical(top:
Radius.circular(25.0)),
              ),
              child: listOfAddPost(),
            ),
          ],
        ),
      ),
    );
  }

  Widget listOfAddPost() {
    return SingleChildScrollView(
      keyboardDismissBehavior:
ScrollViewKeyboardDismissBehavior.onDrag,
      child: Column(
```

```
        mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
          SvgPicture.asset(
            IconsAssets.minusIcon,
            colorFilter:
ColorFilter.mode(Theme.of(context).focusColor,
BlendMode.srcIn),
            height: 40,
          ),
          Text(StringsManager.create.tr,
              style: getMediumStyle(
                  color: Theme.of(context).focusColor,
fontSize: 20)),
          const Divider(),
          Padding(
            padding: const EdgeInsetsDirectional.only(bottom:
8.0),
            child: Builder(builder: (builderContext) {
              UserInfoCubit userCubit =
                  BlocProvider.of<UserInfoCubit>(builderContext
, listen: false);
              UserPersonalInfo? personalInfo =
userCubit.myPersonalInfo;

              return Container(
                margin: const EdgeInsetsDirectional.all(3.0),
                child: CustomElevatedButton(
                  onPressed: () async {
                    return await createStory(personalInfo,
userCubit);
                  },
                  isItDone: isItDone,
                  nameOfButton: StringsManager.share.tr,
                ),
```

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

  Future<void> createStory(
      UserPersonalInfo personalInfo, UserInfoCubit userCubit)
async {

      if (isItDone) {
        setState(() => isItDone = false);
        for (final storyDetails in selectedFiles) {
          Uint8List story = storyDetails.selectedByte;
          if (!storyDetails.isThatImage) {
            story = await
createThumbnail(storyDetails.selectedFile) ?? story;
          }
          String blurHash = await
CustomBlurHash.blurHashEncode(story);
          Story storyInfo =
          addStoryInfo(personalInfo, blurHash,
storyDetails.isThatImage);
          if (!mounted) return;
          StoryCubit storyCubit = StoryCubit.get(context);
          await storyCubit.createStory(storyInfo, story);
          if (storyCubit.storyId != '') {
            userCubit.updateMyStories(storyId:
storyCubit.storyId);
          }
          final SharedPreferences sharePrefs =
          await SharedPreferences.getInstance();
```

```
          WidgetsBinding.instance
              .addPostFrameCallback((_) => setState(() =>
isItDone = true));
          sharePrefs.remove(myPersonalId);
          if (!mounted) return;
          Navigator.of(context).pushAndRemoveUntil(
            CupertinoPageRoute(builder: (_) =>
PopupCalling(myPersonalId)),
                (route) => false,
          );
        }
    }
  }

  Future<Uint8List?> createThumbnail(File selectedFile) async {
    final Uint8List? convertImage = await
VideoThumbnail.thumbnailData(
      video: selectedFile.path,
      imageFormat: ImageFormat.PNG,
    );

    return convertImage;
  }

  Story addStoryInfo(
      UserPersonalInfo personalInfo, String blurHash, bool
isThatImage) {
    return Story(
      publisherId: personalInfo.userId,
      datePublished: DateReformat.dateOfNow(),
      caption: "",
      comments: [],
      likes: [],
      blurHash: blurHash,
```

```
        isThatImage: isThatImage,
      );
    }
  }
```

Додаток Д

Лістинг файлу personal_profile_page.dart

```
import 'dart:async';

import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:flutter_phoenix/flutter_phoenix.dart';
import 'package:flutter_svg/svg.dart';
import 'package:get/get.dart';
import 'package:image_picker_plus/image_picker_plus.dart';
import 'package:shared_preferences/shared_preferences.dart';
import '../../../core/functions/toast_show.dart';
import
'../../../data/models/parent_classes/without_sub_classes/user_p
ersonal_info.dart';
import
'../../cubit/firebaseAuthCubit/firebase_auth_cubit.dart';
import
'../../cubit/firestoreUserInfoCubit/user_info_cubit.dart';
import '../register/login_page.dart';
import 'edit_profile_page.dart';

class PersonalProfilePage extends StatefulWidget {
  final String personalId;
  final String userName;

  const PersonalProfilePage(
      {Key? key, required this.personalId, this.userName = ''})
      : super(key: key);

  @override
```

```
  State<PersonalProfilePage> createState() =>
_ProfilePageState();
}

class _ProfilePageState extends State<PersonalProfilePage> {
  final SharedPreferences sharePrefs =
injector<SharedPreferences>();
  final rebuildUserInfo = ValueNotifier(false);
  Size imageSize = const Size(0.00, 0.00);
  final darkTheme = ValueNotifier(false);
  List<Size> imagesSize = [];

  @override
  void initState() {
    darkTheme.value = ThemeOfApp.isThemeDark();

    super.initState();
  }

  @override
  Widget build(BuildContext context) {
    return scaffold();
  }

  Widget scaffold() {
    return WillPopScope(
      onWillPop: () async => true,
      child: ValueListenableBuilder(
        valueListenable: rebuildUserInfo,
        builder: (context, bool rebuildValue, child) =>
            BlocBuilder<UserInfoCubit, UserInfoState>(
              bloc: widget.userName.isNotEmpty
                  ? (BlocProvider.of<UserInfoCubit>(context)
                ..getUserFromUserName(widget.userName))
```

```
                  : (BlocProvider.of<UserInfoCubit>(context)
                ..getUserInfo(widget.personalId,
getDeviceToken: true)),
              buildWhen: (previous, current) {
                if (previous != current && current is
CubitMyPersonalInfoLoaded) {
                  return true;
                }
                if (previous != current && current is
CubitGetUserInfoFailed) {
                  return true;
                }
                if (rebuildValue) {
                  rebuildUserInfo.value = false;
                  return true;
                }
                return false;
              },
              builder: (context, state) {
                if (state is CubitMyPersonalInfoLoaded) {
                  return Scaffold(
                    appBar: isThatMobile
                        ?
appBar(state.userPersonalInfo.userName)
                        : null,
                    body: ProfilePage(
                      isThatMyPersonalId: true,
                      userId: state.userPersonalInfo.userId,
                      userInfo:
ValueNotifier(state.userPersonalInfo),
                      widgetsAboveTapBars: isThatMobile
                          ?
widgetsAboveTapBarsForMobile(state.userPersonalInfo)
```

```
                          :
widgetsAboveTapBarsForWeb(state.userPersonalInfo),
                    ),
                );
              } else if (state is CubitGetUserInfoFailed) {
                ToastShow.toastStateError(state);
                return Text(StringsManager.noPosts.tr,
                    style:
Theme.of(context).textTheme.bodyLarge);
              } else {
                return const ThineCircularProgress();
              }
            },
          ),
      ),
    );
  }

  AppBar appBar(String userName) {
    return AppBar(
        elevation: 0,
        backgroundColor: Theme.of(context).primaryColor,
        title: Text(userName,
            style: getMediumStyle(
                color: Theme.of(context).focusColor, fontSize:
20)),
        actions: [
          IconButton(
            icon: SvgPicture.asset(
              IconsAssets.addIcon,
              colorFilter: ColorFilter.mode(
                  Theme.of(context).focusColor,
BlendMode.srcIn),
              height: 22.5,
```

```
            ),
            onPressed: () => bottomSheet(),
          ),
          IconButton(
            icon: SvgPicture.asset(
              IconsAssets.menuIcon,
              colorFilter: ColorFilter.mode(
                  Theme.of(context).focusColor,
BlendMode.srcIn),
              height: 30,
            ),
            onPressed: () async => bottomSheet(createNewData:
false),
          ),
          const SizedBox(width: 5)
        ]);
  }

  Future<void> bottomSheet({bool createNewData = true}) {
    return showModalBottomSheet<void>(
      context: context,
      builder: (BuildContext context) => CustomBottomSheet(
        bodyText: bodyTextOfBottomSheet(createNewData),
        headIcon: bottomSheetHeadIcon(),
      ),
    );
  }

  ValueListenableBuilder<bool> bottomSheetHeadIcon() {
    return ValueListenableBuilder(
        valueListenable: darkTheme,
        builder: (context, bool themeValue, child) {
          Color themeOfApp =
          themeValue ? ColorManager.white : ColorManager.black;
```

```
          return Text(StringsManager.create.tr,
              style: getBoldStyle(color: themeOfApp, fontSize:
17));
        });
  }

  Padding bodyTextOfBottomSheet(bool createNewData) {
    return Padding(
      padding: const EdgeInsetsDirectional.only(start: 20.0),
      child: createNewData ? columnOfCreateData() :
columnOfThemeData(),
    );
  }

  Column columnOfCreateData() {
    return Column(
      children: [
        createPost(),
        customDivider(),
        createVideo(),
        customDivider(),
        createStory(),
        customDivider(),
        createNewLive(),
        customDivider(),
        Container(
          height: 50,
        )
      ],
    );
  }

  Divider customDivider() =>
```

```
      const Divider(indent: 40, endIndent: 15, color:
ColorManager.grey);

  Column columnOfThemeData() {
    return Column(
      children: [
        changeLanguage(),
        customDivider(),
        changeMode(),
        customDivider(),
        logOut(),
        customDivider(),
        Container(
          height: 50,
        )
      ],
    );
  }

  Widget changeLanguage() {
    return GestureDetector(
      onTap: () {
        AppLanguage.getInstance().changeLanguage();
        Phoenix.rebirth(context);
      },
      child: createSizedBox(StringsManager.changeLanguage.tr,
          icon: Icons.language_rounded),
    );
  }

  GestureDetector changeMode() {
    return GestureDetector(
      onTap: () async {
        await ThemeOfApp.switchTheme();
```

```
        darkTheme.value = ThemeOfApp.isThemeDark();
      },
      child: createSizedBox(StringsManager.changeTheme.tr,
          icon: Icons.brightness_4_outlined),
    );
  }

  Widget logOut() {
    return BlocBuilder<FirebaseAuthCubit,
FirebaseAuthCubitState>(
        builder: (context, state) {
          FirebaseAuthCubit authCubit =
FirebaseAuthCubit.get(context);
          if (state is CubitAuthSignOut) {
            WidgetsBinding.instance.addPostFrameCallback((_)
async {
              sharePrefs.clear();
              Navigator.of(context, rootNavigator:
true).pushAndRemoveUntil(
                CupertinoPageRoute(
                    builder: (_) => const LoginPage(),
maintainState: false),
                    (route) => false,
              );
            });
          } else if (state is CubitAuthConfirming) {
            ToastShow.toast(StringsManager.loading.tr);
          } else if (state is CubitAuthFailed) {
            ToastShow.toastStateError(state);
          }
          return GestureDetector(
            child: createSizedBox(StringsManager.logOut.tr,
                icon: Icons.logout_rounded),
            onTap: () async {
```

```
              await authCubit.signOut(userId:
widget.personalId);
            },
          );
        });
  }

  List<Widget> widgetsAboveTapBarsForMobile(UserPersonalInfo
userInfo) {
    return [
      editProfileButtonForMobile(userInfo),
      const SizedBox(width: 5),
      const RecommendationPeople(),
      const SizedBox(width: 10),
    ];
  }

  Expanded editProfileButtonForMobile(UserPersonalInfo
userInfo) {
    return Expanded(
      child: Builder(builder: (buildContext) {
        UserPersonalInfo myPersonalInfo =
        UserInfoCubit.getMyPersonalInfo(context);
        UserPersonalInfo? info =
        UsersInfoReelTimeBloc.getMyInfoInReelTime(context);
        if (isMyInfoInReelTimeReady && info != null)
myPersonalInfo = info;
        return InkWell(
          onTap: () async {
            Navigator.maybePop(context);
            Future.delayed(Duration.zero, () async {
              await Go(context).push(page:
EditProfilePage(userInfo));
              rebuildUserInfo.value = true;
```

```
              userInfo = myPersonalInfo;
            });
          },
          child: Container(
            height: 35.0,
            decoration: BoxDecoration(
              color: Theme.of(context).primaryColor,
              border: Border.all(
                  color:
Theme.of(context).bottomAppBarTheme.color!,
                  width: 1.0),
              borderRadius: BorderRadius.circular(20.0),
            ),
            child: Center(
              child: Text(
                StringsManager.editProfile.tr,
                style: TextStyle(
                    fontSize: 17.0,
                    color: Theme.of(context).focusColor,
                    fontWeight: FontWeight.w500),
              ),
            ),
          ),
        );
      }),
    );
  }

  List<Widget> widgetsAboveTapBarsForWeb(UserPersonalInfo
userInfo) {
    return [
      const SizedBox(width: 20),
      editProfileButtonForWeb(),
      const SizedBox(width: 10),
```

```
      GestureDetector(
        child: const Icon(Icons.settings_rounded, color:
ColorManager.black),
      ),
    ];
  }

  Widget editProfileButtonForWeb() {
    return GestureDetector(
      child: Container(
        padding: const EdgeInsets.symmetric(horizontal: 7,
vertical: 5),
        decoration: BoxDecoration(
          color: ColorManager.transparent,
          border: Border.all(
            color: ColorManager.lowOpacityGrey,
            width: 1,
          ),
          borderRadius: BorderRadius.circular(3),
        ),
        child: Text(
          StringsManager.editProfile.tr,
          style: getMediumStyle(color: ColorManager.black),
        ),
      ),
    );
  }

  Widget createNewLive() {
    return InkWell(
      onTap: () {},
      child: createSizedBox(StringsManager.live.tr,
          nameOfPath: IconsAssets.instagramHighlightStoryIcon),
    );
```

```
  }

  Widget createStory() {
    return InkWell(
        onTap: () async => createNewStory(true),
        child: createSizedBox(StringsManager.story.tr,
            nameOfPath: IconsAssets.addInstagramStoryIcon));
  }

  // TODO: handle the video selection (aspect ratio especially)
  Widget createVideo() {
    return InkWell(
        onTap: () async {
          Navigator.maybePop(context);

          await CustomImagePickerPlus.pickVideo(context);

          rebuildUserInfo.value = true;
        },
        child: createSizedBox(StringsManager.reel.tr,
            nameOfPath: IconsAssets.videoIcon));
  }

  createNewStory(bool isThatStory) async {
    Navigator.maybePop(context);

    SelectedImagesDetails? details = await
CustomImagePickerPlus.pickImage(
      context,
      isThatStory: true,
    );
    if (!mounted ) return;
    if (details == null) return;
```

```
    await Go(context).push(page:
CreateStoryPage(storiesDetails: details));
    rebuildUserInfo.value = true;
  }

  createNewPost() async {
    Navigator.maybePop(context);

    await CustomImagePickerPlus.pickFromBoth(context);

    rebuildUserInfo.value = true;
  }

  Widget createPost() {
    return InkWell(
        onTap: createNewPost, child:
createSizedBox(StringsManager.post.tr));
  }

  Widget createSizedBox(String text,
      {String nameOfPath = '', IconData icon =
Icons.grid_on_rounded}) {
    return SizedBox(
      height: 40,
      child: ValueListenableBuilder(
        valueListenable: darkTheme,
        builder: (context, bool themeValue, child) {
          Color themeOfApp =
          themeValue ? ColorManager.white : ColorManager.black;

          return Row(children: [
            nameOfPath.isNotEmpty
                ? SvgPicture.asset(
              nameOfPath,
```

```
                colorFilter: ColorFilter.mode(
                    Theme.of(context).dialogBackgroundColor,
                    BlendMode.srcIn),
                height: 25,
              )
                  : Icon(icon, color: themeOfApp),
              const SizedBox(width: 15),
              Text(
                text,
                style: getNormalStyle(color: themeOfApp,
fontSize: 15),
              )
            ]);
          },
        ),
      );
    }
}
```

Додаток Е

Лістинг файлу search_about_user.dart

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

  @override
  State<SearchAboutUserPage> createState() =>
_SearchAboutUserPageState();
}

class _SearchAboutUserPageState extends
State<SearchAboutUserPage> {
  final ValueNotifier<TextEditingController> _textController =
      ValueNotifier(TextEditingController());

  @override
  void dispose() {
    _textController.value.dispose();
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    final mediaQuery = MediaQuery.of(context);
    final bodyHeight = mediaQuery.size.height -
        AppBar().preferredSize.height -
        mediaQuery.padding.top;

    return Scaffold(
      appBar: isThatMobile ? buildAppBar(context) : null,
      body: BlocBuilder<SearchAboutUserBloc,
SearchAboutUserState>(
        bloc: BlocProvider.of<SearchAboutUserBloc>(context)
```

```
          ..add(FindSpecificUser(_textController.value.text)),
        buildWhen: (previous, current) =>
            previous != current && (current is
SearchAboutUserBlocLoaded),
        builder: (context, state) {
          if (state is SearchAboutUserBlocLoaded) {
            List<UserPersonalInfo> stateUsersInfo =
state.users;

            return ListView.separated(
                itemBuilder: (context, index) {
                  String hash =
"${stateUsersInfo[index].userId.hashCode}";

                  return ListTile(
                    title: Text(stateUsersInfo[index].userName,
                        style: getNormalStyle(
                            fontSize: 15, color:
Theme.of(context).focusColor)),
                    subtitle: Column(
                      crossAxisAlignment:
CrossAxisAlignment.start,
                      children: [
                        Text(stateUsersInfo[index].name,
                            style: getNormalStyle(
                                fontSize: 13,
                                color:
Theme.of(context).disabledColor)),
                        if (stateUsersInfo[index]
                            .followerPeople
                            .contains(myPersonalId)) ...[
                          Text(StringsManager.youFollowHim.tr,
                              style: getNormalStyle(
                                  fontSize: 10,
```

```
                                    color:
Theme.of(context).disabledColor)),
                          ] else if (stateUsersInfo[index]
                              .followedPeople
                              .contains(myPersonalId)) ...[
                            Text(StringsManager.followers.tr,
                                style: getNormalStyle(
                                    fontSize: 10,
                                    color:
Theme.of(context).disabledColor)),
                          ],
                        ],
                      ),
                      leading: Hero(
                        tag: hash,
                        child: CircleAvatarOfProfileImage(
                          bodyHeight: bodyHeight * 0.85,
                          hashTag: hash,
                          userInfo: stateUsersInfo[index],
                        ),
                      ),
                      onTap: () {
                        Go(context).push(
                            page: WhichProfilePage(
                                userId:
stateUsersInfo[index].userId),
                            withoutRoot: false);
                      },
                    );
                  },
                  itemCount:
                      //  _textController.text.isEmpty?0:
                      stateUsersInfo.length,
```

```
                separatorBuilder: (BuildContext context, int
index) =>
                    const SizedBox(
                      height: 10,
                    ));
            } else {
              return buildCircularProgress();
            }
          },
        ),
      );
  }

  AppBar buildAppBar(BuildContext context) {
    return AppBar(
      toolbarHeight: 50,
      iconTheme: IconThemeData(color:
Theme.of(context).focusColor),
      backgroundColor: Theme.of(context).primaryColor,
      title: Container(
        width: double.infinity,
        height: 35,
        decoration: BoxDecoration(
            color: Theme.of(context).shadowColor,
            borderRadius: BorderRadius.circular(10)),
        child: TextFormField(
          style: Theme.of(context).textTheme.bodyLarge,
          controller: _textController.value,
          textAlign: TextAlign.start,
          decoration: InputDecoration(
              contentPadding: const
EdgeInsetsDirectional.all(12.5),
              hintText: StringsManager.search.tr,
```

```
                hintStyle:
Theme.of(context).textTheme.displayLarge,
                border: InputBorder.none),
            onChanged: (_) => setState(() {}),
          ),
        ),
      );
    }

    Widget buildCircularProgress() => const
ThineCircularProgress();
}
```

Додаток Ж

Лістинг файлу post_cubit.dart

```
import 'dart:typed_data';
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:image_picker_plus/image_picker_plus.dart';
import '../../../domain/use_cases/post/create_post.dart';
part 'post_state.dart';

class PostCubit extends Cubit<PostState> {
  final CreatePostUseCase _createPostUseCase;
  final GetPostsInfoUseCase _getPostsInfoUseCase;
  final GetAllPostsInfoUseCase _getAllPostInfoUseCase;
  final UpdatePostUseCase _updatePostUseCase;
  final DeletePostUseCase _deletePostUseCase;

  Post? newPostInfo;
  List<Post>? myPostsInfo;
  List<Post>? userPostsInfo;

  List<Post>? allPostsInfo;

  PostCubit(
      this._createPostUseCase,
      this._getPostsInfoUseCase,
      this._updatePostUseCase,
      this._deletePostUseCase,
      this._getAllPostInfoUseCase)
      : super(CubitPostLoading());

  static PostCubit get(BuildContext context) =>
BlocProvider.of(context);
```

```
  Future<void> createPost(Post postInfo, List<SelectedByte>
files,
      {Uint8List? coverOfVideo}) async {
    newPostInfo = null;
    emit(CubitPostLoading());
    await _createPostUseCase
        .call(paramsOne: postInfo, paramsTwo: files,
paramsThree: coverOfVideo)
        .then((postInfo) {
      newPostInfo = postInfo;
      emit(CubitPostLoaded(postInfo));
    }).catchError((e) {
      emit(CubitPostFailed(e));
    });
  }

  Future<void> getPostsInfo(
      {required List<dynamic> postsIds,
      required bool isThatMyPosts,
      int lengthOfCurrentList = -1}) async {
    emit(CubitPostLoading());
    await _getPostsInfoUseCase
        .call(paramsOne: postsIds, paramsTwo:
lengthOfCurrentList)
        .then((postsInfo) {
      if (isThatMyPosts) {
        myPostsInfo = postsInfo;
        emit(CubitMyPersonalPostsLoaded(postsInfo));
      } else {
        userPostsInfo = postsInfo;
        emit(CubitPostsInfoLoaded(postsInfo));
      }
    }).catchError((e) {
      emit(CubitPostFailed(e));
```

```
    });
  }

  Future<void> getAllPostInfo(
      {bool isVideosWantedOnly = false, String skippedVideoUid
= ""}) async {
    emit(CubitPostLoading());
    await _getAllPostInfoUseCase
        .call(paramsOne: isVideosWantedOnly, paramsTwo:
skippedVideoUid)
        .then((allPostsInfo) {
      this.allPostsInfo = allPostsInfo;
      emit(CubitAllPostsLoaded(allPostsInfo));
    }).catchError((e) {
      emit(CubitPostFailed(e));
    });
  }

  Future<void> updatePostInfo({required Post postInfo}) async {
    emit(CubitUpdatePostLoading());
    await _updatePostUseCase.call(params:
postInfo).then((postUpdatedInfo) {
      if (myPostsInfo != null) {
        int index = myPostsInfo!.indexOf(postInfo);
        myPostsInfo![index] = postUpdatedInfo;
        emit(CubitMyPersonalPostsLoaded(myPostsInfo!));
      }
      emit(CubitUpdatePostLoaded(postUpdatedInfo));
    }).catchError((e) {
      emit(CubitPostFailed(e));
    });
  }

  Future<void> deletePostInfo({required Post postInfo}) async {
```

```
    emit(CubitDeletePostLoading());
    await _deletePostUseCase.call(params: postInfo).then((_) {
      if (myPostsInfo != null) {
        myPostsInfo!
            .removeWhere((element) => element.postUid ==
postInfo.postUid);
        emit(CubitMyPersonalPostsLoaded(myPostsInfo!));
      }
      emit(CubitDeletePostLoaded());
    }).catchError((e) {
      emit(CubitPostFailed(e));
    });
  }
}
```

Додаток К

Лістинг файлу activity_for_mobile.dart

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

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

  @override
  State<ActivityPage> createState() => _ActivityPageState();
}

class _ActivityPageState extends State<ActivityPage> {
  late UserPersonalInfo myPersonalInfo;
  final ValueNotifier<bool> rebuildUsersInfo =
ValueNotifier(false);

  @override
  void initState() {
    myPersonalInfo = UserInfoCubit.getMyPersonalInfo(context);
    super.initState();
  }

  @override
  Widget build(BuildContext context) {
    return isThatMobile
        ? Scaffold(
            appBar: AppBar(title:
Text(StringsManager.activity.tr)),
            body: buildBody(context),
          )
        : buildBody(context);
```

```
  }

  BlocBuilder<UserInfoCubit, UserInfoState>
buildBody(BuildContext context) {
    return BlocBuilder<UserInfoCubit, UserInfoState>(
      bloc:
UserInfoCubit.get(context)..getAllUnFollowersUsers(myPersonalIn
fo),
      buildWhen: (previous, current) =>
          (previous != current && current is
CubitAllUnFollowersUserLoaded),
      builder: (context, unFollowersState) {
        return BlocBuilder<NotificationCubit,
NotificationState>(
          bloc: NotificationCubit.get(context)
            ..getNotifications(userId: myPersonalId),
          buildWhen: (previous, current) =>
              (previous != current && current is
NotificationLoaded),
          builder: (context, notificationState) {
            if (unFollowersState is
CubitAllUnFollowersUserLoaded &&
                notificationState is NotificationLoaded) {
              return SingleChildScrollView(
                child: Column(
                  mainAxisAlignment: MainAxisAlignment.start,
                  crossAxisAlignment: CrossAxisAlignment.start,
                  children: [
                    if
(notificationState.notifications.isNotEmpty)
                      _ShowNotifications(
                        notifications:
notificationState.notifications,
                      ),
```

```
                    if (unFollowersState.usersInfo.isNotEmpty)
...[
                      suggestionForYouText(context),
                      ShowMeTheUsers(
                        usersInfo: unFollowersState.usersInfo,
                        showColorfulCircle: false,
                        emptyText:
StringsManager.noActivity.tr,
                        isThatMyPersonalId: true,
                      ),
                    ],
                  ],
                ),
              );
            } else if (notificationState is NotificationFailed
&&
                unFollowersState is
CubitAllUnFollowersUserLoaded) {
              return SingleChildScrollView(
                child: Column(
                  mainAxisAlignment: MainAxisAlignment.start,
                  crossAxisAlignment: CrossAxisAlignment.start,
                  children: [
                    suggestionForYouText(context),
                    ShowMeTheUsers(
                      usersInfo: unFollowersState.usersInfo,
                      showColorfulCircle: false,
                      emptyText: StringsManager.noActivity.tr,
                      isThatMyPersonalId: true,
                    ),
                  ],
                ),
              );
```

```
            } else if (unFollowersState is
CubitGetUserInfoFailed &&
                notificationState is NotificationLoaded) {
              return SingleChildScrollView(
                child: _ShowNotifications(
                  notifications:
notificationState.notifications,
                ),
              );
            } else if (notificationState is NotificationFailed
&&
                unFollowersState is CubitGetUserInfoFailed) {
              ToastShow.toast(notificationState.error);
              return Center(
                child: Text(
                  StringsManager.somethingWrong.tr,
                  style: getNormalStyle(color:
Theme.of(context).focusColor),
                ),
              );
            }
            return const Center(child:
ThineCircularProgress());
          },
        );
      },
    );
  }

  Padding suggestionForYouText(BuildContext context) {
    return Padding(
      padding: const EdgeInsetsDirectional.all(15),
      child: Text(
        StringsManager.suggestionsForYou.tr,
```

```
        style:
            getMediumStyle(color: Theme.of(context).focusColor,
fontSize: 16),
      ),
    );
  }
}

class _ShowNotifications extends StatefulWidget {
  final List<CustomNotification> notifications;

  const _ShowNotifications({Key? key, required
this.notifications})
      : super(key: key);

  @override
  State<_ShowNotifications> createState() =>
_ShowNotificationsState();
}

class _ShowNotificationsState extends State<_ShowNotifications>
{
  late UserPersonalInfo myPersonalInfo;

  @override
  initState() {
    myPersonalInfo = UserInfoCubit.getMyPersonalInfo(context);
    super.initState();
  }

  @override
  Widget build(BuildContext context) {
    if (widget.notifications.isNotEmpty) {
      return ListView.separated(
```

```
          shrinkWrap: true,
          primary: false,
          physics: const NeverScrollableScrollPhysics(),
          addAutomaticKeepAlives: false,
          addRepaintBoundaries: false,
          itemBuilder: (context, index) {
            return NotificationCardInfo(
                notificationInfo: widget.notifications[index]);
          },
          separatorBuilder: (context, index) => const
  SizedBox(height: 10),
          itemCount: widget.notifications.length,
        );
      } else {
        return Center(
          child: Text(
            StringsManager.noActivity.tr,
            style: Theme.of(context).textTheme.bodyLarge,
          ),
        );
      }
    }
  }
```

Додаток Л

Лістинг файлу chatting_page.dart

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

class ChattingPage extends StatefulWidget {
  final SenderInfo? messageDetails;
  final String chatUid;
  final bool isThatGroup;

  const ChattingPage(
      {Key? key,
      this.messageDetails,
      this.chatUid = "",
      this.isThatGroup = false})
      : super(key: key);

  @override
  State<ChattingPage> createState() => _ChattingPageState();
}

class _ChattingPageState extends State<ChattingPage>
    with TickerProviderStateMixin {
  final ValueNotifier<Message?> deleteThisMessage =
ValueNotifier(null);

  final unSend = ValueNotifier(false);

  @override
  Widget build(BuildContext context) {
    return widget.messageDetails != null
        ? scaffold(widget.messageDetails!)
```

```
        : getUserInfo(context);
  }

  Widget getUserInfo(BuildContext context) {
    return BlocBuilder<MessageCubit, MessageState>(
      bloc: MessageCubit.get(context)
        ..getSpecificChatInfo(
            isThatGroup: widget.isThatGroup, chatUid:
widget.chatUid),
      buildWhen: (previous, current) =>
          previous != current && current is
GetSpecificChatLoaded,
      builder: (context, state) {
        if (state is GetSpecificChatLoaded) {
          return scaffold(state.coverMessageDetails);
        } else if (state is GetMessageFailed) {
          ToastShow.toast(state.error);

          return Scaffold(
              body: Center(child:
Text(StringsManager.somethingWrong.tr)));
        } else {
          return const Scaffold(body: ThineCircularProgress());
        }
      },
    );
  }

  Scaffold scaffold(SenderInfo messageDetails) {
    return Scaffold(
      appBar: isThatMobile
          ?
CustomAppBar.chattingAppBar(messageDetails.receiversInfo!,
context)
```

```
          : null,
      body: GestureDetector(
          onTap: () {
            unSend.value = false;
            deleteThisMessage.value = null;
          },
          child: isThatMobile
              ? ChatMessages(messageDetails: messageDetails)
              : buildBodyForWeb(messageDetails)),
    );
  }

  Widget buildBodyForWeb(SenderInfo messageDetails) {
    return Column(
      children: [
        buildUserInfo(messageDetails.receiversInfo![0]),
        ChatMessages(messageDetails: messageDetails)
      ],
    );
  }

  Column buildUserInfo(UserPersonalInfo userInfo) {
    return Column(
      children: [
        circleAvatarOfImage(userInfo),
        const SizedBox(height: 10),
        nameOfUser(userInfo),
        const SizedBox(height: 5),
        userName(userInfo),
        const SizedBox(height: 5),
        someInfoOfUser(userInfo),
        viewProfileButton(userInfo),
      ],
    );
```

```
}

Widget circleAvatarOfImage(UserPersonalInfo userInfo) {
  return CircleAvatarOfProfileImage(
    bodyHeight: 1000,
    userInfo: userInfo,
    showColorfulCircle: false,
    disablePressed: false,
  );
}

Row userName(UserPersonalInfo userInfo) {
  return Row(
    mainAxisAlignment: MainAxisAlignment.center,
    children: [
      Text(
        userInfo.userName,
        style: TextStyle(
            color: Theme.of(context).focusColor,
            fontSize: 14,
            fontWeight: FontWeight.w300),
      ),
      const SizedBox(
        width: 10,
      ),
      Text(
        "Instagram",
        style: TextStyle(
            color: Theme.of(context).focusColor,
            fontSize: 14,
            fontWeight: FontWeight.w300),
      ),
    ],
  );
```

```
  }

  Text nameOfUser(UserPersonalInfo userInfo) {
    return Text(
      userInfo.name,
      style: TextStyle(
          color: Theme.of(context).focusColor,
          fontSize: 16,
          fontWeight: FontWeight.w400),
    );
  }

  Row someInfoOfUser(UserPersonalInfo userInfo) {
    return Row(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        Text(
          "${userInfo.followerPeople.length}
${StringsManager.followers.tr}",
          style: TextStyle(
              color:
Theme.of(context).textTheme.titleSmall!.color,
              fontSize: 13),
        ),
        const SizedBox(
          width: 15,
        ),
        Text(
          "${userInfo.posts.length}
${StringsManager.posts.tr}",
          style: TextStyle(
              fontSize: 13,
              color:
Theme.of(context).textTheme.titleSmall!.color),
```

```
          ),
        ],
      );
    }

    TextButton viewProfileButton(UserPersonalInfo userInfo) {
      return TextButton(
        onPressed: () {
          Go(context).push(page: UserProfilePage(userId:
  userInfo.userId));
        },
        child: Text(StringsManager.viewProfile.tr,
            style: TextStyle(
                color: Theme.of(context).focusColor,
                fontWeight: FontWeight.normal)),
      );
    }
  }
```

Додаток М

Кошторис витрат на проектування

Найменування статей витрат	Сума, грн	Обґрунтування
1 Зарплата проектувальників	901280	Заробітна плата працівників визначається виходячи з кількості виконавців, діючих посадових окладів та кількості місяців їх участі в розробці проекту
2. Відрахування на соціальні потреби	533510	Включають в себе відрахування на соціальні програми та потреби працівників, такі як податок на доходи фізичних осіб, військовий збір, єдиний внесок
3. Контрагентські роботи і послуги	0	Не передбачено
4. Витрати на відрядження	0	Не передбачено

Продовження таблиці «Кошторис витрат на проектування»

5. Інші прямі витрати	20178	Витрати на закупівлю матеріалів та програмного забезпечення
6. Усього прямих витрат	1454968	Сума загальної вартості усіх безпосередніх витрат, які пов'язані з реалізацією проекту
7. Накладні витрати	581987	Включає в себе витрати на оренду приміщень, комунальні послуги та інші загальні витрати
8. Планові накопичення	611086	Відображає заплановані накопичення коштів на майбутні потреби чи резерви
9.Усього, кошторисна вартість проекту	2648041	Сума всіх вищезазначених витрат, яка визначає загальну кошторисну вартість проекту
10. Податок на додану вартість	529608	Відображає оподаткування на додану вартість

Продовження таблиці «Кошторис витрат на проектування»

11. Загалом, договірна ціна розробки	3177649	Сума, за яку виконавець зобов'язаний виконати розробку в рамках договору, що включає усі вищезазначені витрати разом із прибутком

Додаток Н

Розрахунок заробітної плати проєктувальників

№ п/п	Посада виконавця	Оклад, грн/міс	Відрахування грн/міс	Кількість		Сума з/п, грн.
				чол.	місяців	
1	Керівник проєкту	79000	29337	1	4	198652
2	Програміст-розробник	98000	36479	2	4	492168
3	Тестувальник	71000	26302	1	2	89386
4	Архітектор програмного забезпечення	50800	18880	1	1	31920
5	UI/UX дизайнер	64000	23826	1	1	40174
6	Адміністратор бази даних	39000	14510	1	2	48980
		Усього зарплати:				901280