МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені І.І.МЕЧНИКОВА

(повне найменування вищого навчального закладу)

Факультет математики, фізики та інформаційних технологій

(повне найменування інституту, назва факультету (відділення))

Кафедра комп'ютерної алгебри та дискретної математики

(повна назва кафедри (предметної, циклової комісії))

Дипломна робота

на здобуття ступеня вищої освіти «бакалавр»

(освітньо-кваліфікаційний рівень)

на тему

«Реалізація frontend закритої соціальної мережі HypeFans»

«Implementation of the frontend of the closed social network HypeFans»

Виконав: студент денної форми навчання
спеціальності 123 – Комп'ютерна інженерія .

(шифр і назва напряму підготовки, спеціальності)

Грабовецький Фелікс Олександрович

(прізвище, ім'я, по-батькові)

Керівник канд. фіз-мат. наук, доц. Савастру О.В

(науковий ступінь, вчене звання, прізвище та ініціали, **підпис**)

Рецензент канд. тех.наук, доц. Якімова Н.А

(науковий ступінь, вчене звання, прізвище та ініціали)

Рекомендовано до захисту:

Протокол засідання кафедри

№ від « » 2021 р.

Завідувач кафедри

(підпис) Варбанець П.Д (прізвище, ініціали)

Захищено на засіданні ЕК №

протокол № від « » 2021 р.

Оцінка / /

(за національною шкалою, шкалою ECTS, бали)

Голова ЕК

(підпис) Казакова Н.Ф (прізвище, ініціали)

Одеса - 2021

АНОТАЦІЯ

У дипломній роботі розробляється тема «Реалізація frontend закритої соціальної мережі HypeFans».

Мета роботи – створення клієнтської частини з можливістю обміну інформацією між людьми, перегляду історій інших користувачів, розміщенням публікацій, можливості заробітку на створенні контенту.

Відмінні риси дипломного проекту полягають у можливості оформлення платної підписки на певного користувача для перегляду його публікацій та можливості спілкування з ним.

АННОТАЦИЯ

В дипломной работе разрабатывается тема «Реализация frontend закрытой социальной сети HypeFans».

Цель работы – создание клиентской части с возможностью обмена информацией между людьми, просмотра других пользователей, размещением публикаций, возможности заработка на создании контента.

Отличительные особенности дипломного проекта заключаются в возможности оформления платной подписки на определенного пользователя для просмотра его публикаций и возможностей общения с ним.

ANNOTATION

In this thesis the topic «Implementation of the frontend of the closed social network HypeFans» is being developed.

The purpose of the work is to create a client side with the ability to exchange information between people, view other users, post publications and make money on content creation.

Distinctive features of the thesis are the ability to issue a paid subscription to a specific user to view his publications and the ability to communicate with him.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ

Скорочення

API – Application Programming Interface (інтерфейс створення застосунків)

AUTH – загальна назва блоку реєстрації та входу у соціальну мережу

DOM – Document Object Model (специфікація прикладного програмного інтерфейсу для роботи зі структурованими документами)

PC – персональний комп’ютер

ООП – об’єктно-орієнтоване програмування

ПЗ – програмне забезпечення

Терміни

Дейтинг – сайт для знайомств

Content maker – людина, яка створює та публікує контент

Верстка – процес в якому дизайн макета переходить у програмний код

Cookie – поняття, яке використовується для опису інформації у вигляді текстових або бінарних даних, отриманих від веб-сайту на веб-сервері

CVV – зашифрований в магнітній смузі номер, який використовується в картах міжнародної платіжної системи

Authorization token – токени доступу використовуються при аутентифікації на основі маркерів, щоб дозволити програмі отримати доступ до API.

ВСТУП

У наш час складно зустріти людину, яка б не користувалась хоча б однієї з існуючих соціальних мереж для обміну текстовою або графічної, аудіо та відео інформацією, для знайомств або спілкування за інтересами [8]. Три мільярди людей в усьому світі, тобто близько 40% населення, користуються соціальними мережами в інтернеті [4]. Люди витрачають на них в середньому дві години щодня: публікують дописи, обмінюються фото, реагують на пости друзів. Соціальні медіа відіграють у нашому житті таку велику роль, то дуже важливо усвідомлювати, як вони впливають на нас.

Соціальні мережі – це місце, де люди нерідко висловлюють своє обурення з приводу будь-чого, від неякісних послуг до політичних проблем. Опитування в якому взяли участь 1800 людей показало, що жінки набагато більше, ніж чоловіки, відчувають стрес від соцмереж [5]. Дослідники дійшли висновку, що користування соцмережами пов’язане із «порівняно низьким рівнем стресу».

Існує багато типів соціальних мереж:

- Контактні соціальні мережі («Однокласники», «Facebook» тощо);
- Блог-платформи («LiveJournal», «LiveInternet» тощо);
- Тематичні соціальні мережі («MySpace.com, «kroogi.com» тощо);
- Дейтинги («Badoo», «OkCupid», «Loveplanet» тощо).

Соціальна мережа HypeFans відноситься до контактних соціальних мереж. Користувач має можливість спілкуватись з іншими користувачами, обмінюватися інформацією, фото, публікувати різноманітні пости, реагувати на пости друзів тощо.

У наш час для того, щоб почати заробляти в соціальних мережах, необхідно мати достатню аудиторію, щоб користувачі почали купувати твій продукт особистого бренду, або ж щоб почали замовляти рекламу [7].

Проблема, яку вирішує закрита соціальна мережа – немає необхідності мати велику аудиторію для стартового заробітку, а саме можна встановлювати ціни і отримувати гроші з перших фанатів.

Мета дипломної роботи – створення клієнтської частини закритої соціальної мережі для спілкування між користувачами, обміну інформації за допомогою повідомлень, нових історій і т.п.

Особливістю HypeFans є можливість донатів, платних підписок і т.п.

Для досягнення треба вирішити наступні завдання:

- проектування архітектури;
- створення дизайну соціальної мережі;
- верстка сторінок соціальної мережі;
- підключення готового API.

Методи дослідження для досягнення поставленої в роботі мети:

- виявлення цільової аудиторії;
- виявлення необхідного аудиторією функціонала шляхом соцопитування цільової аудиторії;
- аналіз функціоналу подібних випущених продуктів.

1. ТЕОРЕТИЧНА ЧАСТИНА

React Native – багатоплатформовий фреймворк з відкритим вихідним кодом для розробки мобільних і настільних застосунків за допомогою JavaScript та TypeScript. Фреймворк дозволяє використовувати можливості бібліотеки React поза браузера для створення нативних застосунків, які мають повний доступ до системних платформ. Принципи роботи React Native практично ідентичні React, за винятком того, що React Native не керує DOM через віртуальну DOM.

Компоненти React обгортають існуючий власний код і взаємодіють з власними API через парадигму декларативного призначеного для користувача інтерфейсу React і Java Script. Це дозволяє розробляти власні програми для нових команд розробників і дозволяє існуючим групам працювати набагато швидше.

Для React Native характерні такі переваги [1]:

- Загальна кодова база - відбувається розробка двох окремих версій програми під IOS та Android (код в них не повністю однаковий). Загальний код мінімізує кількість багів по ходу розробки і значно спрощує підтримку продукту в майбутньому;
- Підтримка TypeScript – статична типизація це менше багів, підтримка проекту та можливість легко створити додаток з шаблону.

TypeScript – мова програмування, представлена Microsoft. Вона є зворотньо сумісною з JavaScript. Після компіляції, програму на TypeScript можна виконувати в будь-якому сучасному браузері. TypeScript відрізняється від JavaScript можливістю явного статичного призначення типів, підтримкою використання повноцінних класів, підтримкою підключення модулів, що підвищує швидкість розробки, полегшає рефакторинг і повне використання коду, допомогає здійснювати пошук помилок на етапі розробки і компіляції, і прискорити використання програм [6].

TypeScript має такі переваги [2]:

- Можливість явного визначення типів (статична типізація);
- Підтримка використання повноцінних класів (як в традиційних ООП мовах);
- Підтримка підключення модулів.

Java Script – динамічна, об’єктно-орієнтована прототипна мова програмування. Найчастіше використовується для створення сценаріїв веб сторінок, шо надає можливість на боці клієнта взаємодіяти з користувачем, керувати браузером, асинхронно обмінюватися даними з сервером, змінювати структуру та зовнішній вигляд веб сторінки. JS класифікують як прототипну скриптову мову програмування з динамічною типізацією. Окрім прототипної, JavaScript також частково підтримує інші парадигми програмування (імперативну та часткову функціональну) і деякі відповідні архітектурні властивості, зокрема: динамічна та слабка типізація, автоматичне керування пам’яттю, прототипне наслідування функції як об’єкти першого класу.
JavaScript містить декілька десятків вбудованих об’єктів, які поділяються на групи [3]:

- Фундаментальні (Object, Function, Boolean, Symbol);
- Помилки (група об’єктів Error);
- Числа та дати (Number, Math Date);
- Текстові (String, RegExp);
- Індексовані (група об’єктів Array);
- Ключові (Map, Set);
- Для роботи з структурованими даними (JSON).

Мова JavaScript використовується для написання сценаріїв веб сторінок для надання їм інтерактивності, створення односторінкових та прогресивних вебзастосунків, програмування на боці сервера, стаціонарних застосунків, мобільних застосунків, сценаріїв в прикладних програмах.

За допомогою JS доступні до виконання наступні функції:

- Можливість змінювати сторінки браузерів;
- Інформація про дії користувача на сторінці;
- Запит доступу до випадкової частини вихідного коду сторінки;
- Виконання дій з cookie-файлами.

Область застосування цієї мови програмування нічим не обмежена: присутні тестові редактори, застосунки (як для PC, мобільні, серверні) і прикладне ПЗ.

API – забезпечує взаємодію між двома системами. «Веб-сервіс» - це веб-додаток, що надає ресурси в форматі, використовуваному іншими комп'ютерами. Веб-сервіси – це, в основному, запити та відповіді між клієнтами і серверами (комп'ютер запитує ресурс, а веб-сервіс відповідає на запит). API, що використовують протокол HTTP для передачі запитів в відповідей, розглядаються як «веб-сервіси». Між клієнтом і сервером API існує запит і відповідь. Клієнт і сервер можуть бути засновані на будь-якій мові, але HTTP – це протокол, який використовується для передачі повідомлення [9]. Цей шаблон запиту і відповіді, по суті, є принципом роботи моделі API REST (рис.1).

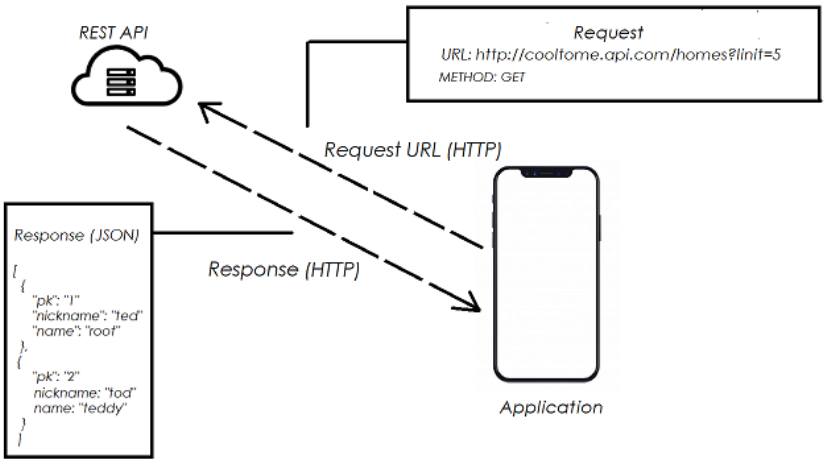

Рисунок 1. Загальна модель REST API

2. ПОСТАНОВКА ЗАВДАННЯ

Мета дипломної роботи - створення та реалізація клієнтської частини соціальної мережі.

Для того, щоб виконати поставлену мету, необхідно було вирішити наступні завдання:

- Визначення типу застосунку (мобільний застосунок);
- Формування складання списку необхідного функціоналу;
- Створення дизайну соціальної мережі;
- Визначення патерна проектування (frontend – Container and View паттерн);
- Верстка сторінок мобільного застосунка;
- Підключення готового API.

За допомогою розробленої соціальної мережі користувачі мають можливість поширити свій контент на платній або безкоштовній основі без необхідності нарощування своєї аудиторії.

3. ОБГРУНТУВАННЯ ПРОЕКТНОГО РІШЕННЯ

У сучасному суспільстві в ізоляції люди стали активніше користуватися мобільними девайсами для роботи, розваг і соціалізації. Частина цього часу припала на мобільні застосунки: вже у 2020 році користувачі стали проводити в застосунках на 20% більше часу в день. Індустрія мобільних застосунків пережила вибухове зростання з початку карантину. Саме із цих причин вибір був зроблений на користь створення мобільного застосунка.

Дизайн був створений на підставі рейтингу популярності соціальних мереж і однією із найпопулярніших є Інстаграмм. Такий рейтинг користувачів соціальними мережами є також наслідком того, що люди велику увагу приділяють зовнішньому вигляді продукту.

На відміну від багатьох соціальних мереж у HypeFans люди можуть почати заробляти гроші вже у перші часи користування. Наприклад, в Інстаграммі для того, щоб заробити гроші треба набрати достатню кількість підписників, вкладатися у рекламу, що вже має на увазі під собою достатню кількість часу і грошей. Головна особливість HypeFans – мати свій контент та почати заробляти з самого початку.

Переваги соціальної мережі для людини, яка публікує контент по-перше це зручний інтерфейс, по-друге менший необхідний обсяг підписників для початку заробітку за рахунок того, що користувач може почати заробляти з дій першого передплатника. У застосунку HypeFans при створенні акаунта будуть висвітлюватися в рекомендаціях люди з піковою популярністю, що дозволяє зі старту проаналізувати нинішні найбільш трендові течії в сфері Інтернет контенту. При переході в даний сервіс з іншої соціальної мережі перевага полягає в тому, що у людини, яка публікує контент з’являється додаткова можливість заробляти гроші на власній аудиторії за рахунок публікації контенту з умовним «раннім доступом», можливість створення тематичних бесід з ціною для входу звичайний користувач, спрощені фінансові транзакції (оскільки в мережі тільки цифровой контент, оплата в межах проекту стає

автоматизованої і content maker немає необхідності відстежувати кожен куплений у нього товар, оскільки гроші за операцію автоматично опиняються на внутрішньому балансі).

Переваги соціальної мережі для користувача соціальної мережі це зручний інтерфейс, можливість підтримати улюбленого content maker'а з отриманням контенту з раннім доступом, можливість підписки на бесіди групових заходів content maker'а. Також спрощений перехід від звичайного користувача до автора контенту за допомогою прив'язки банківської картки та створення та публікації контенту.

4. ПРАКТИЧНА ЧАСТИНА ПРОЕКТУ

4.1. Верстка основних сторінок соціальної мережі

Для верстки сторінок соціальної мережі HypeFans було використано паттерн Container and View, який є найбільш ефективним і широко використовуваним шаблоном побудови функцій в середовищі React Native.

Для початку був спроектований блок AUTH, у якому знаходяться сторінки реєстрації (рис. 4.1) (сторінка, де користувач вперше використовуючи соціальну мережу реєструється) та входу (рис. 4.2) (сторінка, де користувач маючи вже аккаунт у соціальній мережі заходить у неї). Програмна реалізація цього блогу наведена в додатку А.

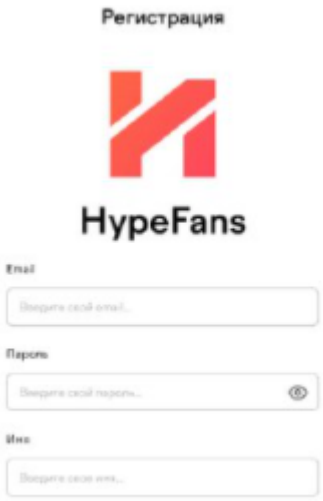

Рисунок 4.1 – Фрагмент сторінки реєстрації

Рисунок 4.2 – Фрагмент сторінки входу

Після того, як користувач увійшов у свій аккаунт або зареєструвався, він потрапляє на основну сторінку блоку - блог (рис. 4.3), де знаходяться «історії» інших користувачів та їх публікації. Біля кожної публікації можна поставити like та залишити коментар. Програмна реалізація цього блогу наведена в додатку А.

Рисунок 4.3 – Фрагмент сторінки блогу

Після верстки блогу був зверстан блок повідомлень (рис. 4.4). Програмна реалізація цього блогу наведена в додатку А.

Користувач буде отримувати різноманітні види повідомлень, такі як:

- Пост, який сподобався іншому користувачеві;
- Коментарі, які залишають користувачі під постами;
- Підписка на профіль користувача;
- Донат.

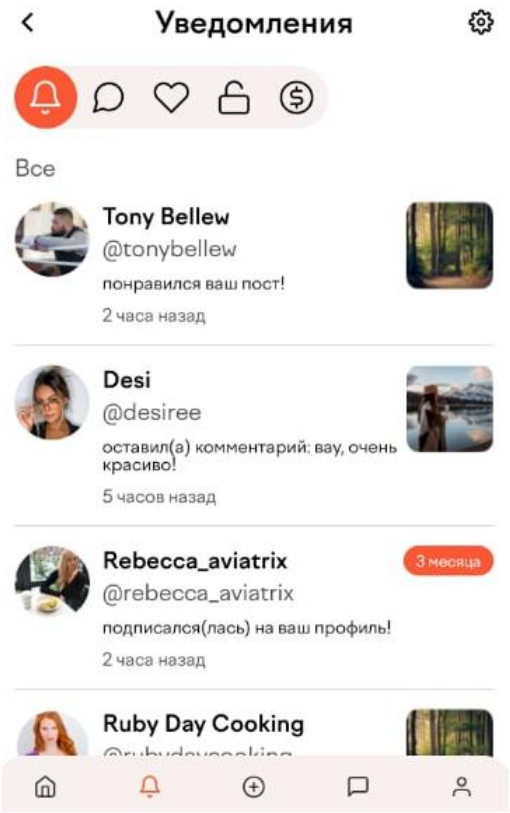

Рисунок 4.4 – Фрагмент сторінки повідомлень

Після верстки блоку повідомлень був зверстан блок створення постів та історій (рис. 4.5). Програмна реалізація цього блогу наведена в додатку А. Користувач може викласти на свою сторінку різну інформацію: картинка, фотографія, документ тощо. Також у користувача буде можливість поставити обмеження (коли він хоче, щоб пост зник).

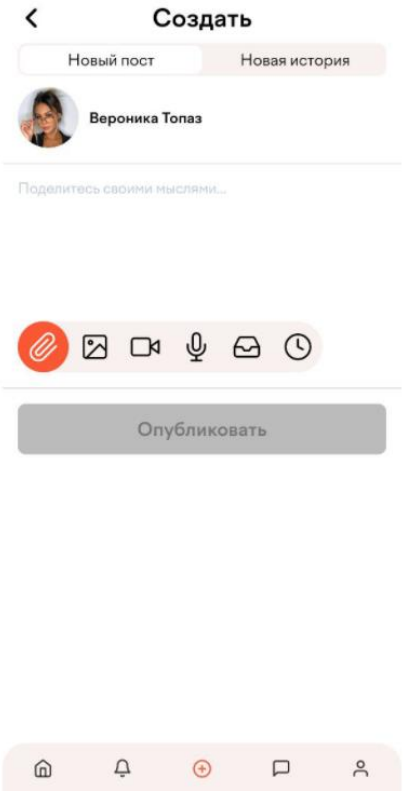

Рисунок 4.5 – Створення посту

Окрім, створення посту користувач має нагоду записати «історію» та опублікувати її на своїй сторінці. Для цього додаток запитує дозвіл на роботу камери, і якщо користувач дає позитивну відповідь, то він зможе зняти та опублікувати історію. (рис. 4.6 – рис. 4.8)

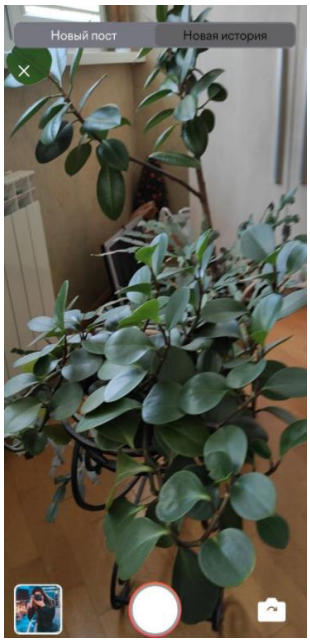

Рисунок 4.6 – Створення історії

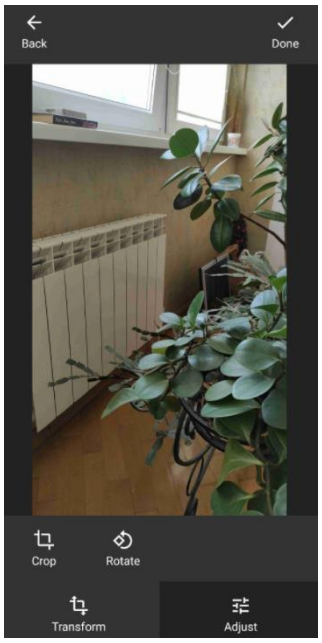

Рисунок 4.7 – Підтвердження публікації історії

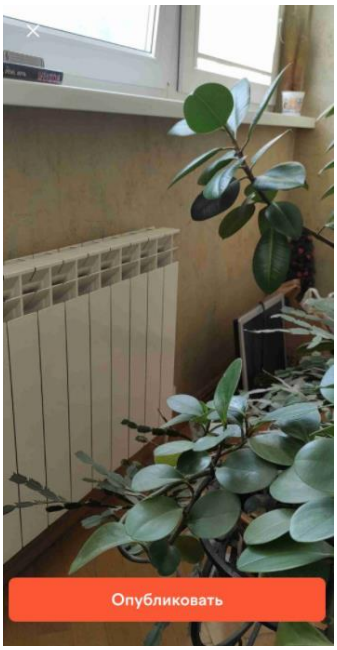

Рисунок 4.8 – Публікація історії

Після верстки блоку посту та історії було зверстано блок Messages (рис. 4.9). У цьому блоці знаходяться чати між власником акаунту та іншими користувачами соціальної мережі. Програмна реалізація цього блогу наведена в додатку А.

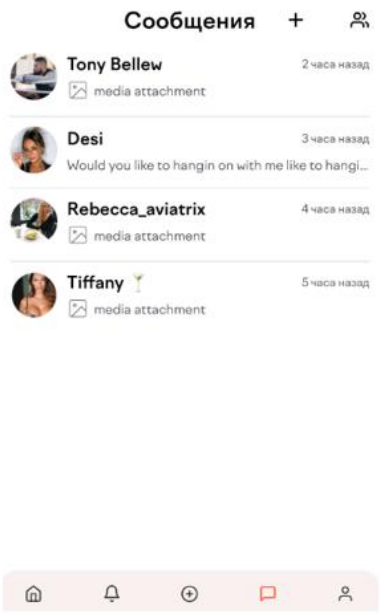

Рисунок 4.9 – Фрагмент сторінки Messages

Заходячи в повідомлення з іншим користувачем HypeFans є можливість написати йому приватне повідомлення (рис. 4.10). Окрім приватних повідомлень є можливість створення групових бесід (рис. 4.11). Автор групової бесіди може встановити донат за вхід або зробити її безкоштовною. Також додана можливість змінити фотографію групової бесіди. (рис. 4.12)

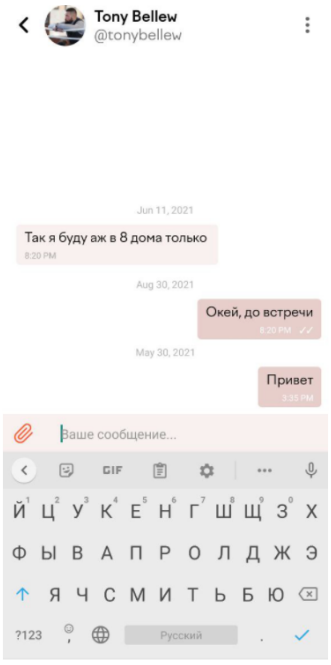

Рисунок 4.10 – Приватне повідомлення

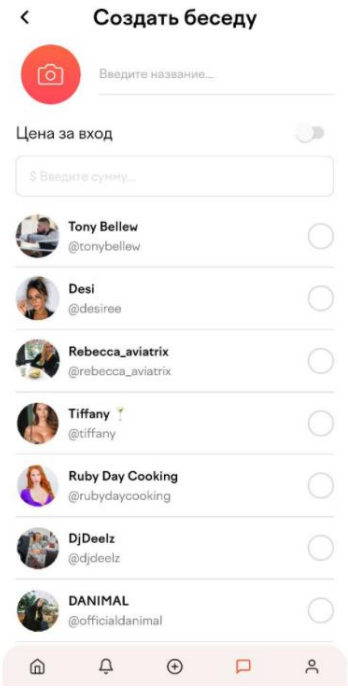

Рисунок 4.11 – Процес створення групових бесід

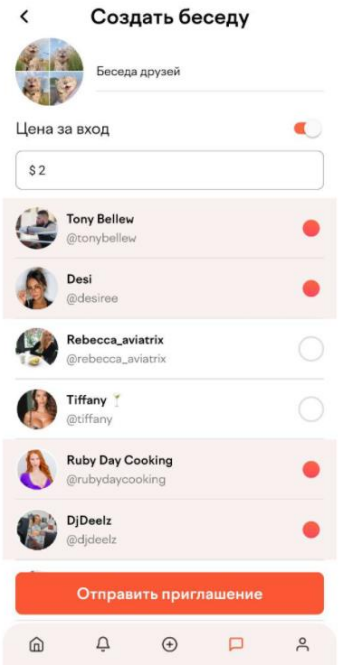

Рисунок 4.12 – Створення групової бесіду з донатом

Після верстки блоку Messages було зверстано блок Profile (рис. 4.13). Програмна реалізація цього блогу наведена в додатку А. У цьому блоці знаходиться уся інформація про користувача.

- Кількість постів, яку виклав користувач;
- Кількість підписників;
- Ім’я користувача та його нікнейм;
- Пости, які публікував користувач.

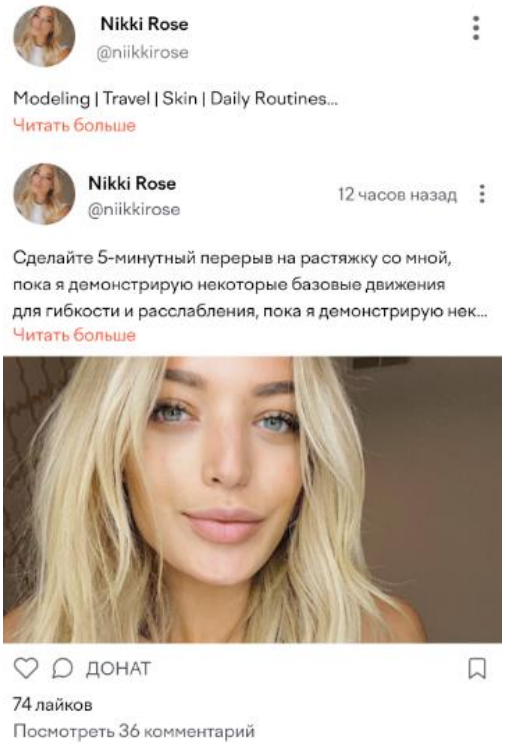

Рисунок 4.13 – Фрагмент сторінки профілю

4.2. Налаштування параметрів сторінки Profile

Після створення блоку Profile необхідно зверстати компоненти налаштувань (рис. 4.14), для можливості:

- Змінити налаштування профілю;
- Змінити інформацію акаунту (загальна інформація, безпека акаунту, дії з акаунтом);
- Конфіденційність;
- Ціна підписки;
- Повідомлення.

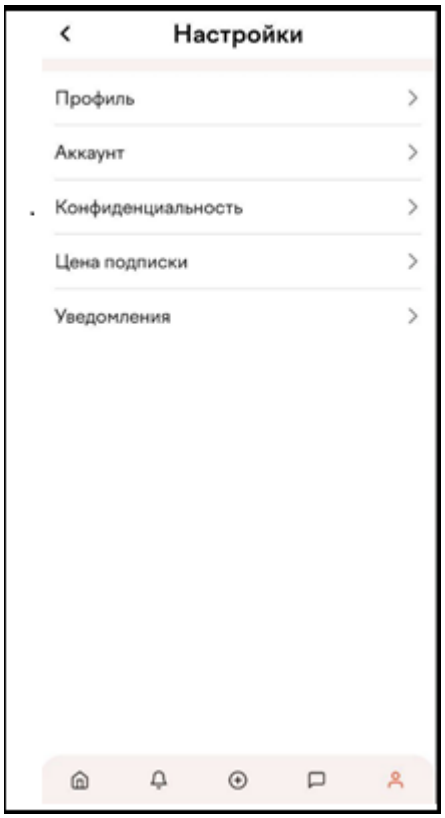

Рисунок 4.14 – Компоненти налаштувань профілю

Для того, щоб змінити налаштування профілю було додано ряд функцій, які може виконати користувач (рис. 4.15) . Користувач може змінити фотографію профілю, своє ім’я, свій нікнейм, додати особисту інформацію, додати ціну за персональні messages, ціну за підписку і т.п.

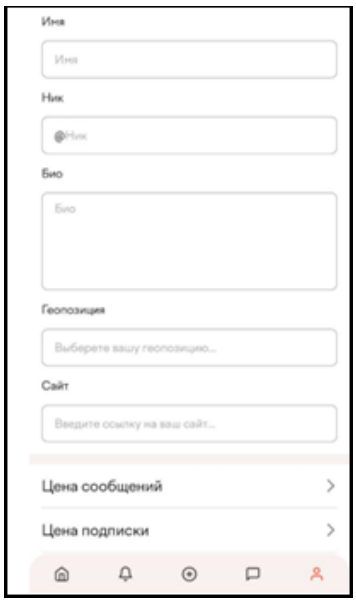

Рисунок 4.15 – Функції налаштування профілю

Для того, щоб змінити інформацію акаунту було додано ряд функцій, які може виконати користувач (рис. 4.16).

- Змінити загальну інформацію (змінити нікнейм, email, вказати номер телефону);
- Встановити безпеку акаунту (змінити пароль, переглянути останні сеанси входу в акаунт);
- Видалити акаунт.

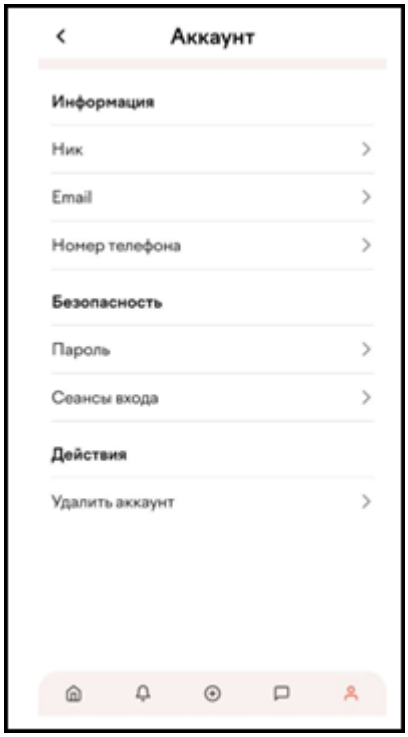

Рисунок 4.16 – Функції налаштування акаунту

Для того, щоб налаштувати конфіденційність було додано ряд функцій, які може виконати користувач (рис. 4.17).

- Показувати статус активності;
- Показувати кількість підписників;
- Показувати кількість постів;
- Побачити заблоковані акаунти і при бажанні їх розблокувати.

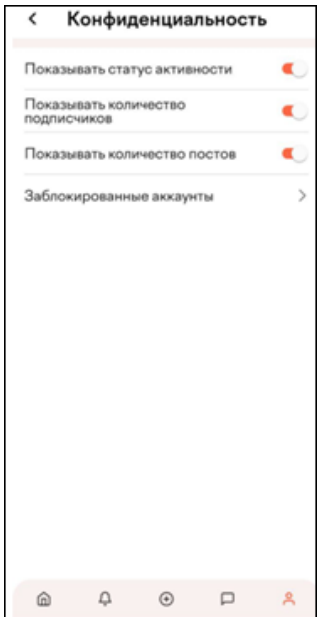

Рисунок 4.17 – Функції налаштування конфіденційності

Для того, щоб змінити налаштування ціни підписки було додано функцію, яка додавала можливість відправляти донат користувачу (рис. 4.18).

Рисунок 4.18 – Функція налаштування ціни підписки

Для того, щоб налаштувати повідомлення було додано ряд функцій, які може виконати користувач (рис. 4.19).

- Повідомлення, які приходять, коли користувач не знаходиться на HypeFans;
- Отримання електронних листів, коли користувач не знаходиться на HypeFans;
- Повідомлення на сайті (нові коментарі, нові лайки, нові підписки та нові донати).

Рисунок 4.19 – Функція налаштування повідомленнь

4.3. Налаштування параметрів картки для можливості донату

Для можливості заробляти гроші у мобільному застосунку Hypefans користувач має можливість додати банківську картку (рис. 4.20). Налаштування має такі параметри:

- Номер картки;
- Ім’я людини, яка є власником банківської картки;
- Термін та спеціальний код CVV.

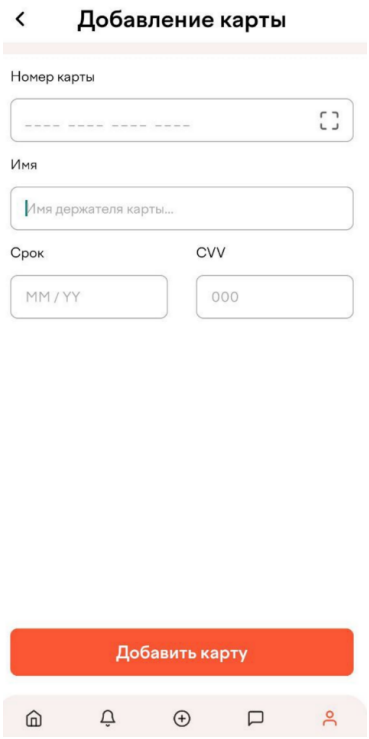

Рисунок 4.20 – Функція додавання карти

Після того, як користувач зареєстрував банківську картку, він потрапляє на сторінку (рис. 4.21), де може побачити саму картку, її номер, ім’я отримувача картки та її термін. Також він може перевірити рахунок картки.

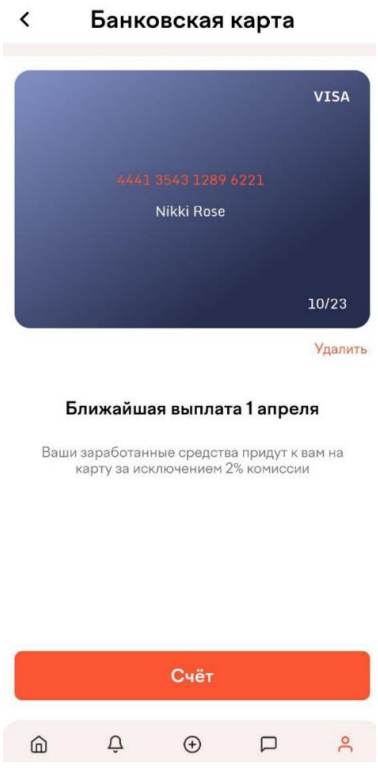

Рисунок 4.21 – Сторінка банківської картки

4.4. Налаштування параметрів донату та витратів

Для можливості перегляду заробітку на HypeFans користувач має можливість (рис. 4.22 – 4.23) :

- Переглядати історію донату;
- Витрати;
- Загальний баланс.

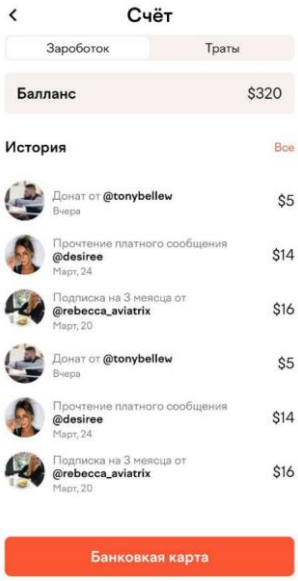

Рисунок 4.22 – Сторінка рахунку, загальний баланс, історія донатів

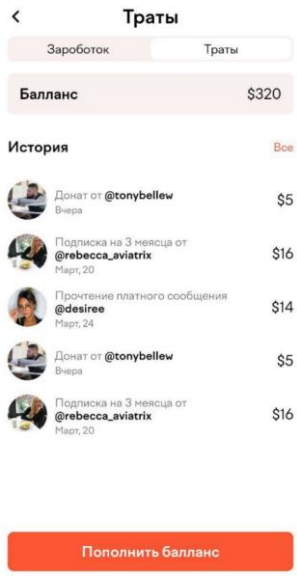

Рисунок 4.23 – Витрати користувача

4.5. Налаштування параметрів повідомленнь

Для можливості користувачу соціальної мережі вибирати чи хоче він отримувати різні види повідомленнь була створена автоматичне налаштування перемикання повідомленнь (рис. 4.24 – 4.26).

- Перемикання PUSH-повідомлень (користувач буде отримувати PUSH повідомлення, щоб дізнатися, що відбувається, коли він не знаходиться на HypeFans;
- Перемикання EMAIL-повідомлень (користувач буде отримувати електронні листи, щоб дізнатися, що відбувається коли він не знаходиться на HypeFans;
- Перемикання повідомленнь на сайті (нові коментарі, нові лайки, нові підписки, нові донати).

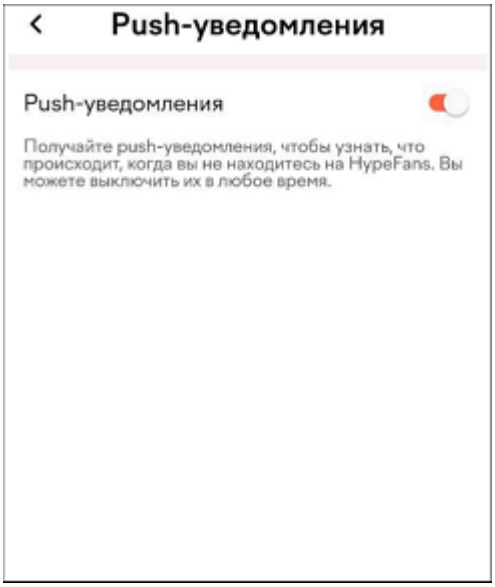

Рисунок 4.24 – Перемикання PUSH-повідомлень

Рисунок 4.25 – Перемикання EMAIL-повідомленнь

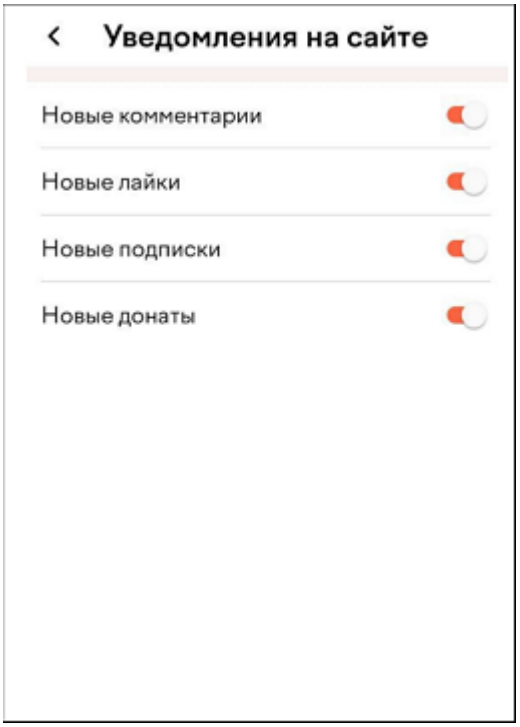

Рисунок 4.26 – Перемикання повідомлень на сайті

4.6. Налаштування параметрів конфіденційності

Конфіденційність – особиста інформація користувача, яку не один з інших користувачів соціальної мережі не повинен виявити. У конфіденційність входить:

- Показувати статус активності;
- Показувати кількість підписників;
- Показувати кількість постів;
- Заблоковані акаунти.

У цьому блоці буде приведене налаштування заблокованих акаунтів (рис 4.27).

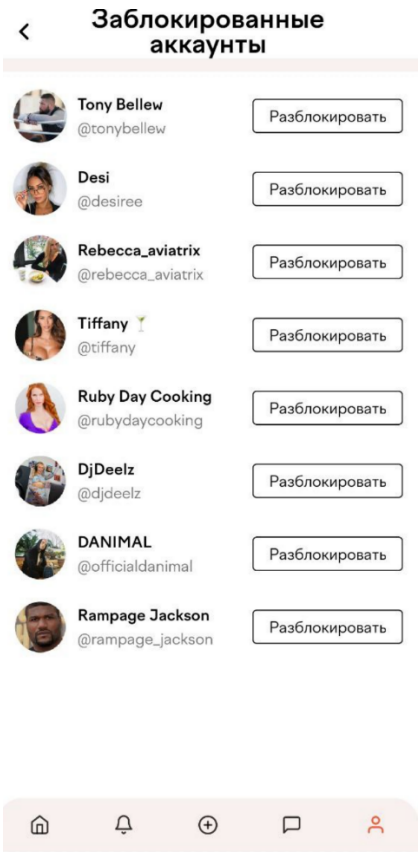

Рисунок 4.27 – Заблоковані акаунти

4.7. Підключення API для реєстрації та авторизації

Для того, щоб користувач мав можливість увійти в систему було реалізовано підключення методів API логіну, для реєстрації – створення користувача. Для того, щоб система розуміла, що користувач авторизований, в подальших запитах був використаний Authorization token. У цьому блоці приведено запит на авторізацію та реєстрацію в результаті цього запросу користувач отримує токен (рис 4.28, рис 4.29). Програмна реалізація цього блогу наведена в додатку В.

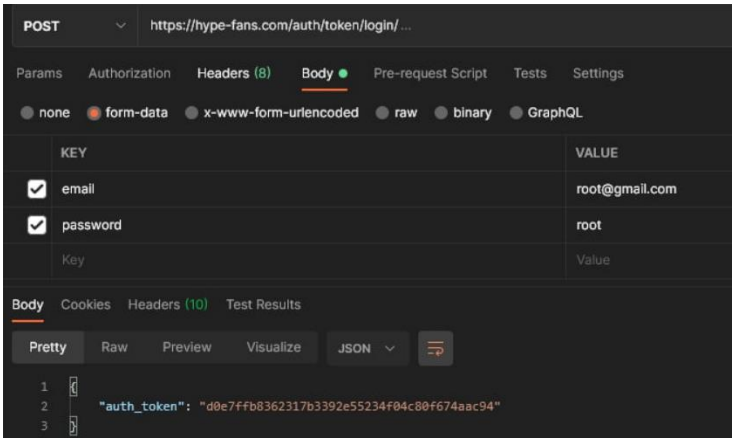

Рисунок 4.28 – Результат запиту авторизації та отримання authentication token

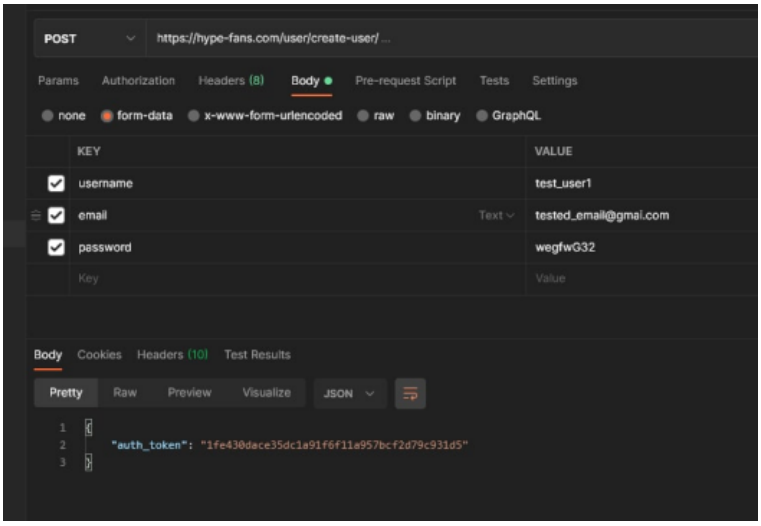

Рисунок 4.29 – Результат запиту реєстрації та отримання authentication token

4.8. Підключення API для отримання даних користувача

Для того, щоб користувач мав можливість подивитися інформацію, яка міститься в його профілі був реалізований метод API для отримання даних про користувача, щоб знати актуальну інформацію. У цьому блоці приведено запит на отримані даних про користувача соціальної мережі, в результаті цього запиту була отримана повна інформація щодо користувача застосунку HypeFans (рис 4.30). Програмна реалізація цього блоку наведена у додатку В.

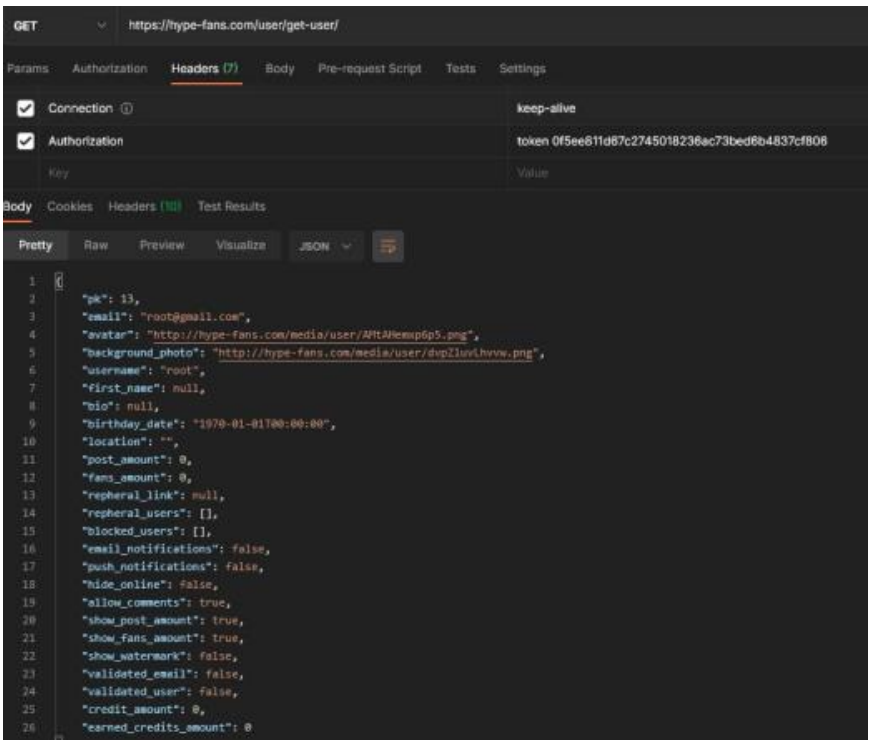

Рисунок 4.30 – Результат запиту отримання даних та отримання загальної інформації щодо користувача

4.9. Підключення API для поновлення даних користувача

Для того, щоб користувач мав можливість поновити інформацію, яка міститься в його профілі був реалізований метод API для отримання поновлених даних про користувача, щоб знати актуальну інформацію. У цьому блоці приведено запит на поновлення даних про користувача соціальної мережі, в результаті цього запиту була отримана повна поновлена інформація щодо користувача застосунку HypeFans (рис 4.31). Програмна реалізація цього блоку наведена у додатку В.

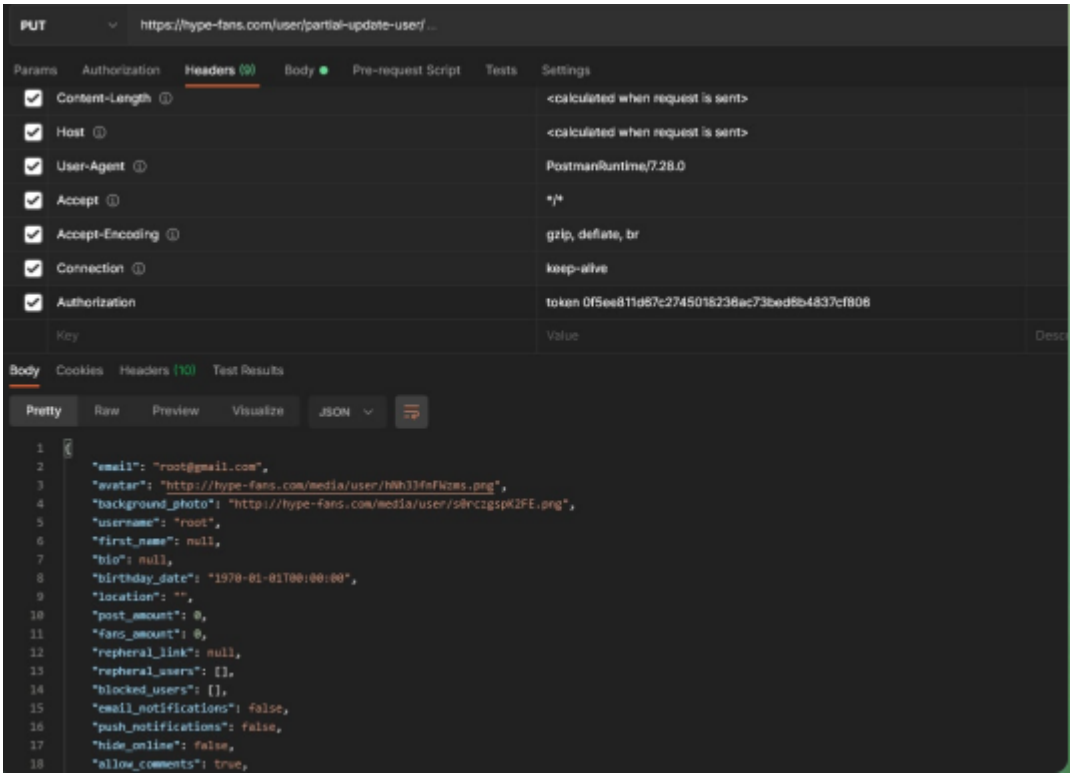

Рисунок 4.31 – Результат запиту отримання поновлених даних та отримання загальної інформації щодо користувача

4.10. Підключення API для формування стрічки користувача

Після реєстрації або входу в акаунт, користувач знаходиться на сторінці блогу, де він може побачити інших користувачів соціальної мережі та їх пости, які набули популярності останнім часом. Для цього був реалізований метод API для отримання стрічки користувача, яка складається з користувачів які викладають пости і самими постами. У цьому блоці приведено запит на отримання стрічки соціальної мережі, в результаті цього запиту була отримана стрічка користувача застосунку HypeFans (рис 4.32). Програмна реалізація цього блоку наведена у додатку В.

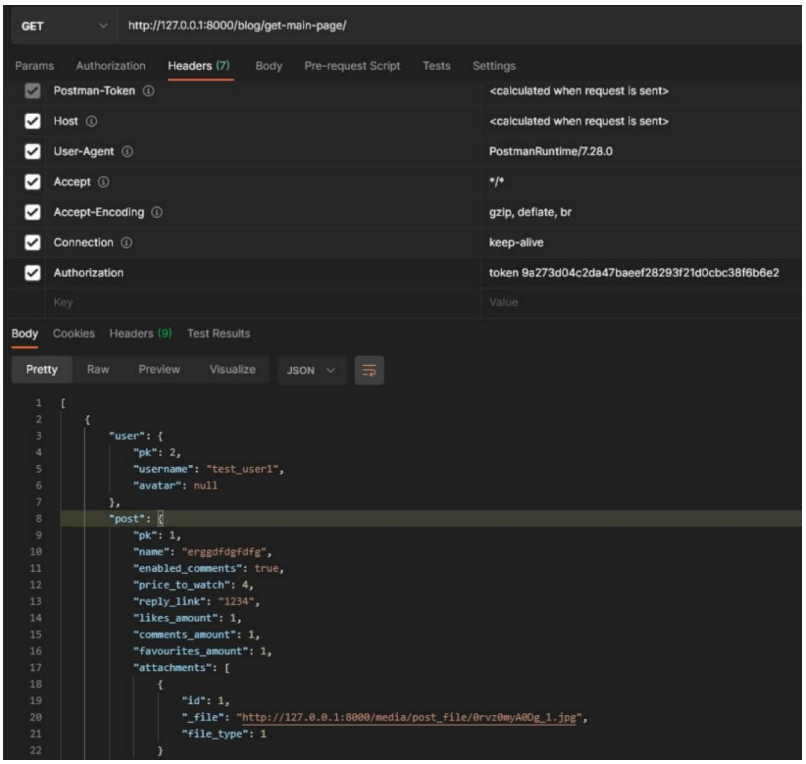

Рисунок 4.32 – Результат запиту отримання стрічки соціальної мережі користувача

4.11 Підключення API для створення публікації

Для того, щоб користувач мав можливість створити публікацію був реалізований метод API для її створення . У цьому блоці приведено запит на створення посту (рис 4.33). Програмна реалізація цього блоку наведена у додатку В.

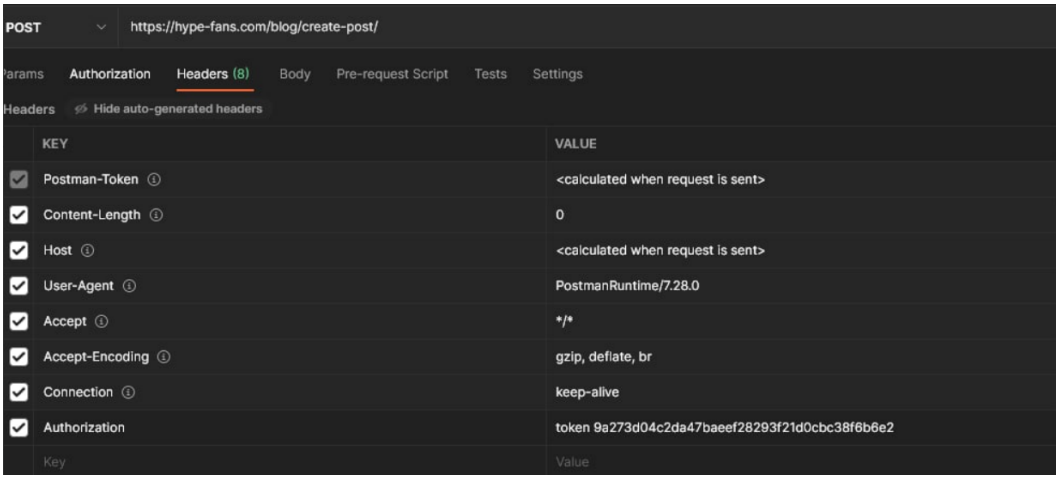

Рисунок 4.33 – Результат запиту створення посту

4.12 Підключення API для отримання діалогів користувача

Для того, щоб користувач мав можливість переглядати поточний стан блогу messages, а саме бачити всі бесіди, які він має з іншими користувачами HypeFans було реалізовано метод API для отримання діалогів користувача. У цьому блоці приведено запит на отримання діалогів користувача (рис 4.34). Програмна реалізація цього блоку наведена у додатку В.

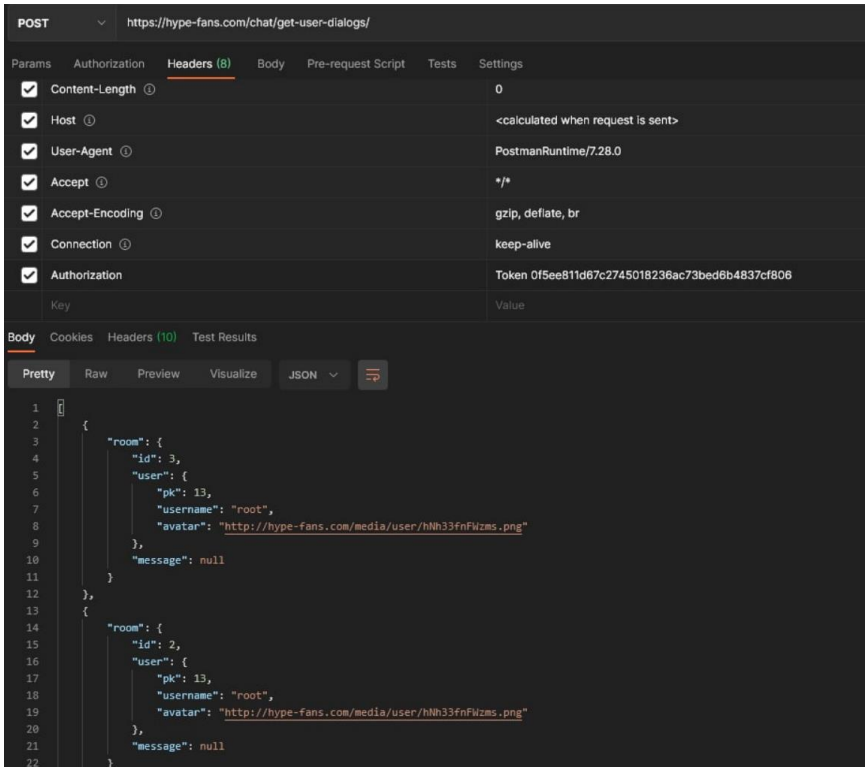

Рисунок 4.34 – Результат запиту отримання діалогів користувача

4.13 Підключення API для отримання повідомлень у діалогах

Для того, щоб користувач мав можливість переглядати поточний стан , листування в момент перегляду діалогу в чаті користувача HypeFans було реалізовано метод API для отримання поточного стану діалогу. У цьому блоці приведено запит на отримання поточного стану (рис 4.35). Програмна реалізація цього блоку наведена у додатку В.

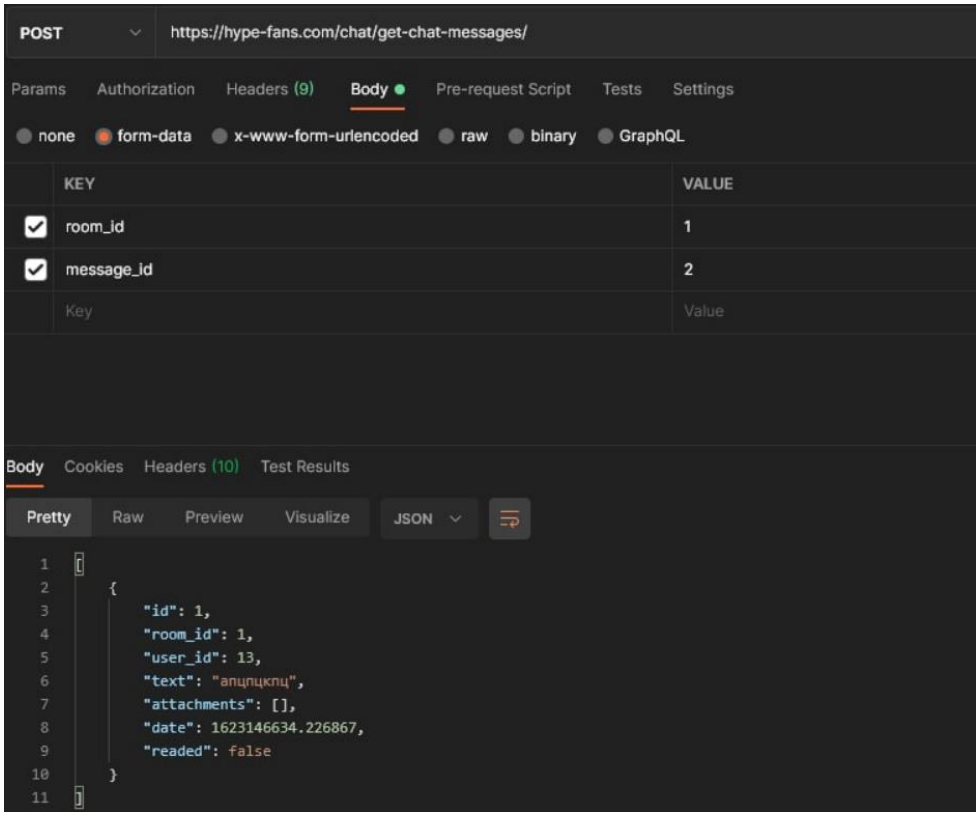

Рисунок 4.35 – Результат запиту отримання поточного стану діалогу

ВИСНОВОК

В процесі виконання дипломного проекту була створена клієнтська частина соціальної мережі HypeFans. Створення мобільного застосунку було обумовлено популярністю зростанням відсотку користування мобільними застосунками (у 2021 році кількість витрачених в день годин на використання мобільних застосунків зросло на 0,3%). Завдяки опитуванню людей був сформований список необхідного функціоналу для мобільного застосунка, а саме ті функції, які має виконувати застосунок, щоб людям було просто і комфортно їм користуватися.

Функціонал застосунку, який люди хотіли бачити в майбутньому вплинув на створення дизайну соціальної мережі. Поєднання палітри кольорів, які використовував дизайнер, має бути приємне для користувача HypeFans.

Завдяки технології React Native реалізований зовнішній вигляд соціальної мережі. Сторінки розміщені у різні блоки по змісту (ті сторінки , що відноситься до блоку повідомлень – вони знаходяться в загальній папці повідомлень і т.п.).

Завдяки технології REST API реалізовано підключення готового API для створення робочого функціоналу соціальної мережі HypeFans.

Подальша робота з розвитком та поліпшенням проекту може вестися в декількох напрямках. По-перше, після випуску застосунку, завдяки соцопитуванню людей, коментарям користувачів у Google Play можна встановити недоліки HypeFans, покращити дизайн і т.п. По-друге, якісне доопрацювання – наприклад, покращення коду прискорює роботу соціальної мережі. По-третє, постійні оновлення, такі як насамперед, поліпшення дизайну, додавання нових функцій застосунку і т.п.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Пуртов О.Н React Native: плюси і мінуси фреймворка в 2020 році / Розробка на React Native – 6 с. Режим доступу: https://www.purrweb.com/blog/ru/react-native-plyusy-i-minusy/
2. HypeHost Переваги використання мови TypeScript – 3 с. Режим доступу: https://hyperhost.ua/info/ru/preimushhestva-ispolzovaniya-yazyika-typescript
3. MDN WEB DOCS Керівництво Java Script – 12 с. Режим доступу: https://developer.mozilla.org/ru/docs/Web/JavaScript/Guide/Regular_Expressions
4. П. В. Сінґер, Емерсон Т. Брукінґ, Війна лайків. Зброя в руках соціальних мереж. – Книжковий Клуб «Клуб Сімейного дозвілля», 2019. – 320 с.
5. Девід Патрікаракос – Війна у 140 знаках. Як соціальні медіа змінюють конфлікт XXI століття. – Yakaboo Publishing, 2019. – 352 с.
6. Володимир Дронов – Angular 4. Швидка розробка сверхдинамічних Web-сайтів на TypeScript та PHP. – БХВ-Петербург, 2018. – 448 с.
7. Сергій Щербаков – Таргетована реклама в соціальних мережах. Отримуйте більше клієнтів з Facebook та Instagram. – Фоліо, 2018. – 256 с.
8. Валентин Холмогоров – Соціальні мережі. Додай інтернет в друзі. – Книжковий клуб «Клуб Сімейного дозвілля», 2012. – 312 с.
9. Mark Masse – REST API Design Rulebook. REST API. – O’Reilly Media, 2011. – 116 с.

ДОДАТОК А

ПРОГРАМНА РЕАЛІЗАЦІЯ СТОРІНОК СОЦІАЛЬНОЇ МЕРЕЖІ

СТОРІНКА ВХОДУ

У цьому фрагменті коду наведено опис поля email-address. У це поле для входу користувач вводить свій електронний адрес.

```
<TextInput
//autoFocus={true}
keyboardType={'email-address'}
secureTextEntry={secure}
style={[s.input, s.text14, s.factor, s.flex1, s.h46, s.mh15,
incorrect ? s.textRed : s.textBlack]}
placeholder={text[lang].emailDescr}
placeholderTextColor={'#aaa'}
//onFocus={() => setFocus(true)}
//onBlur={() => setFocus(false)}
/>
</View>
{incorrect
? <Text style={[s.text14, s.factor, s.mt5, s.textRed, s.mh15]}>{text[
lang].email}</Text>
: null
}
```

У цьому фрагменті коду наведено опис поля password. У це поле для входу користувач вводить пароль, який він вводив при реєстрації акаунту.

```
<TextInput
//autoFocus={true}
//keyboardType={'phone-pad'}
secureTextEntry={secure}
style={[s.input, s.text14, s.factor, s.flex1, s.h46, s.mh15,
incorrect2 ? s.textRed : s.textBlack]}
placeholder={text[lang].passDescr}
placeholderTextColor={'#aaa'}
onChangeText={(txt) => setPass(txt)}
//onFocus={() => setFocus(true)}
//onBlur={() => setFocus(false)}
onEndEditing={() =>
```

```
pass.length < 3 ? setIncorrect2(true) : props.onMainScreen()
}
/>
<TouchableOpacity
style={[s.btn40, s.center, s.mr5]}
activeOpacity={1}
onPress={() => toogleVisible(!secure)}
>
<SvgUri width="24" height="24"
source={secure
? require('../../assets/images/show.svg')
: require('../../assets/images/showNo.svg')
} />
</TouchableOpacity>
</View>
{incorrect2
? <Text style={[s.text14, s.factor, s.mt5, s.textRed, s.mh15]}>{text[
lang].wrongPass}</Text>
: null
```

У цьому фрагменті коду наведено роботу та опис кнопки входу в акаунт.

```
{
true
?
<TouchableOpacity style={[s.orangeBtn, s.center, s.mt40]}
activeOpacity={0.9}
onPress={() => {
pass.length < 3 ? setIncorrect2(true) : props.onMainScreen()
}}
>
<Text style={[s.text18, s.factorBold]}>{text[lang].enterBtn}</Text>
</TouchableOpacity>
:
<TouchableOpacity style={[s.disableBtn, s.center, s.mt40]}
activeOpacity={0.9}
//onPress={() => props.onSkip()}
>
<Text style={[s.text18, s.factorBold, s.textGrey]}>{text[lang].enterB
tn}</Text>
</TouchableOpacity>
}
<TouchableOpacity style={[s.h46, s.center, s.mt10, s.ip10mb]}
activeOpacity={0.9}
onPress={() => props.onRegister()}
>
<Text style={[s.text18, s.factor, s.textBlack]}>{text[lang].createAcc
1}
```

```
<Text style={[s.text18, s.factorBold, s.textOrange]}>{text[lang].crea
teAcc2}</Text>
</Text>
</TouchableOpacity>
<View style={{ height: 60 }} />
</ScrollView>
```

СТОРІНКА РЕЄСТРАЦІЇ

У цьому фрагменті коду наведено опис поля email-address. У це поле для реєстрації акаунту користувач вводить свій електронний адрес.

```
<TextInput
//autoFocus={true}
keyboardType={'email-address'}
secureTextEntry={secure}
style={[s.input, s.text14, s.factor, s.flex1, s.h46, s.mh15,
incorrect ? s.textRed : s.textBlack]}
placeholder={text[lang].emailDescr}
placeholderTextColor={'#aaa'}
//onFocus={() => setFocus(true)}
//onBlur={() => setFocus(false)}
/>
</View>
{incorrect
? <Text style={[s.text14, s.factor, s.mt5, s.textRed, s.mh15]}>{text[
lang].email}</Text>
: null
}
```

У цьому фрагменті коду наведено опис поля password. У це поле для реєстрації користувач вводить пароль, яким надалі він буде користуватися для входу в акаунт.

```
<TextInput
//autoFocus={true}
//keyboardType={'phone-pad'}
secureTextEntry={secure}
style={[s.input, s.text14, s.factor, s.flex1, s.h46, s.mh15,
incorrect2 ? s.textRed : s.textBlack]}
placeholder={text[lang].passDescr}
placeholderTextColor={'#aaa'}
//onFocus={() => setFocus(true)}
//onBlur={() => setFocus(false)}
//onEndEditing={() => props.onMainScreen()}
/>
```

```
<TouchableOpacity
style={[s.btn40, s.center, s.mr5]}
activeOpacity={1}
onPress={() => toogleVisible(!secure)}
>
<SvgUri width="24" height="24"
source={secure
? require('../../assets/images/show.svg')
: require('../../assets/images/showNo.svg')
} />
</TouchableOpacity>
</View>
{incorrect2
? <Text style={[s.text14, s.factor, s.mt5, s.textRed, s.mh15]}>{text[
lang].passWarn}</Text>
: null
}
```

У цьому фрагменті коду наведено опис поля, де користувач вводить своє ім’я для реєстрації.

```
<TextInput
//autoFocus={true}
keyboardType={'email-address'}
secureTextEntry={secure}
style={[s.input, s.text14, s.factor, s.flex1, s.h46, s.mh15,
incorrect2 ? s.textRed : s.textBlack]}
placeholder={text[lang].nameDescr}
placeholderTextColor={'#aaa'}
//onFocus={() => setFocus(true)}
//onBlur={() => setFocus(false)}
onEndEditing={() => props.onMainScreen()}
/>
</View>
{incorrect2
? <Text style={[s.text14, s.factor, s.mt5, s.textRed, s.mh15]}>{text[
lang].name}</Text>
: null
}
```

У цьому фрагменті коду наведено роботу та опис кнопки реєстрації акаунту.

```
{
true
?
<TouchableOpacity style={[s.orangeBtn, s.center, s.mt40]}
activeOpacity={0.9}
onPress={() => props.onMainScreen()}
><Text style={[s.text18, s.factorBold]}>{text[lang].regBtn}</Text>
</TouchableOpacity>
:
<TouchableOpacity style={[s.disableBtn, s.center, s.mt40]}
activeOpacity={0.9}
//onPress={() => props.onSkip()}>
```

СТОРІНКА БЛОГУ

У цьому фрагменті коду наведено опис та створення історій інших користувачів соціальної мережі.

```
users.map((user, index) => {
return (
<TouchableOpacity key={index}
style={[s.ava, s.center, s.mr5]}
activeOpacity={0.8}
onPress={() => navigation.navigate('StoryViewer')}
>
<View style={[s.personIcon, s.center]}>
<AvaGradient style={[s.pabsolute]} />
<View style={[s.whiteRound1, s.pabsolute]} />
<Image
style={[s.image60rw]}
source={user.ava}
/>
</View>
<Text numberOfLines={1} style={[s.text14, s.factor, s.textBlack, s.mt
3]}>{user.nick}</Text>
</TouchableOpacity>
)
```

У цьому фрагменті коду наведено опис та створення публікацій блогу.

```
<Text numberOfLines={full ? null : 3} style={[s.text14, s.textLine21,
 s.factor, s.textBlack]}
onTextLayout={({ nativeEvent: { lines } }) => {
setLines(lines.length)
}}
>{post.text}</Text>
{
full
?
null
: <View style={[]}>
{
lines > 3
? <Text style={[s.h15, s.text14, s.factor, s.textOrange]}>{text[lang]
.redmor}</Text>
: null
}
</View>}
```

У цьому фрагменті коду наведено опис та створення можливостей для користувача соціальної мережі поставити лайк, додати донат та залишити свій коментар під публікацією.

```
<View style={[s.bottomBtns, s.flexRow, s.spaceBtw, s.aCenter, s.mh10]
}>
<View style={[s.flexRow, s.aCenter]}>
<TouchableOpacity style={[s.h40, s.center, s.ph5]}
activeOpacity={0.8}
onPress={() => {
setLike(!like)
like ? setLikeCount(likeCount - 1) : setLikeCount(likeCount + 1)
}}
>
{
like
?
<SvgUri width="21" height="18"
source={require('../../../assets/images/liked.svg')} />
:
<SvgUri width="21" height="18"
source={require('../../../assets/images/heart.svg')} />
}

</TouchableOpacity>
<TouchableOpacity style={[s.h40, s.center, s.ph5]}
activeOpacity={0.8}
onPress={() => console.log('comment')}
>
<SvgUri width="18" height="18"
source={require('../../../assets/images/comment.svg')} />
</TouchableOpacity>
<TouchableOpacity style={[s.h40, s.center]}
activeOpacity={0.8}
//onPress={() => setSearch(!search)}
>
<Text style={[s.text14, s.factor, s.textGrey2, s.ph5]}>{text[lang].do
nut}</Text>
</TouchableOpacity>
</View>
```

СТОРІНКА ПОВІДОМЛЕННЬ

У цьому фрагменті коду наведено створення видів публікацій у вигляді drawer, тобто є:

- Сортування по LIKE
- Сортування по повідомленням
- Сортування по коментарям
- Підписка на профіль

```
<View style={[s.flexRow, s.mh15, s.mt10]}>
{
buttons
?
<View style={[s.buttonsDrawer, s.flexRow]}>
<TouchableOpacity style={[filter == 'all' ? s.orangeRoundBtn : s.draw
erBtn, s.center]}
activeOpacity={0.8}
onPress={() => {
setButtons(!buttons)
setFilter('all')
setActiveNotes(notes)
}}
>
<SvgUri width="30" height="30"
source={filter == 'all'
? require('../../../assets/images/bellW.svg')
: require('../../../assets/images/bellN.svg')}
/>
</TouchableOpacity>
<TouchableOpacity style={[filter == 'comments' ? s.orangeRoundBtn : s
.drawerBtn, s.center]}
activeOpacity={0.8}
onPress={() => {
setFilter('comments')
_sort('comments')
}}
>
<SvgUri width="30" height="30"
source={filter == 'comments'
? require('../../../assets/images/message-circleW.svg')
: require('../../../assets/images/message-circle.svg')
} />
</TouchableOpacity>
<TouchableOpacity style={[filter == 'likes' ? s.orangeRoundBtn : s.dr
awerBtn, s.center]}
activeOpacity={0.8}
```

```
onPress={() => {
setFilter('likes')
_sort('likes')
}}
>
<SvgUri width="30" height="30"
source={filter == 'likes'
? require('../../../assets/images/heartW.svg')
: require('../../../assets/images/heartB.svg')
} />
</TouchableOpacity>
<TouchableOpacity style={[filter == 'subscribs' ? s.orangeRoundBtn :
s.drawerBtn, s.center]}
activeOpacity={0.8}
onPress={() => {
setFilter('subscribs')
_sort('subscribs')}}
>
<SvgUri width="30" height="30"
source={filter == 'subscribs'
? require('../../../assets/images/unlockW.svg')
: require('../../../assets/images/unlock.svg')
} /></TouchableOpacity>
<TouchableOpacity style={[filter == 'donuts' ? s.orangeRoundBtn : s.d
rawerBtn, s.center]}
activeOpacity={0.8}
onPress={() => {
setFilter('donuts')
_sort('donuts')
}}
><SvgUri width="30" height="30"
source={filter == 'donuts'
? require('../../../assets/images/tipW.svg')
: require('../../../assets/images/tip.svg')
} />
</TouchableOpacity>
</View>
:<TouchableOpacity style={[filter == 'all' ? s.orangeRoundBtn : null,
 s.center]}
activeOpacity={0.8}
onPress={() => setButtons(!buttons)}
>
<SvgUri width="30" height="30"
source={filter == 'all'
? require('../../../assets/images/bellW.svg')
: require('../../../assets/images/bellW.svg')
} />
</TouchableOpacity>
}
</View>
```

СТОРІНКА ПОСТІВ ТА ІСТОРІЙ

У цьому фрагменті коду наведено створення текстового поля для написання нового посту користувачем.

```
<TextInput
multiline={true}
textAlignVertical={'top'}
style={[s.text14, s.factor, s.textLine21, s.textBlack, s.mh15, s.mt15
, { height: 120 }]}
placeholder={text[lang].shareMind}
placeholderTextColor={'#bcc9dc'}
onChangeText={(txt) => setMessage(txt)}
/>
```

У цьому фрагменті коду наведено створення видів публікацій посту у вигляді drawer, тобто є:

- Прикріплення відео
- Прикріплення будь-якого документу

```
<View style={[s.flexRow, s.mh15, s.mt10]}>
{
buttons
?
<View style={[s.buttonsDrawer, s.flexRow]}>
<TouchableOpacity style={[s.orangeRoundBtn, s.center]}
activeOpacity={0.8}
onPress={() => {
setButtons(!buttons)
}}
>
<SvgUri width="30" height="30"
source={require('../../../assets/images/skrepa.svg')} />
</TouchableOpacity>
<TouchableOpacity style={[s.drawerBtn, s.center]}
activeOpacity={0.8}
onPress={() => {
pickImage()
setButtons(!buttons)
}}
>
<SvgUri width="30" height="30"
source={require('../../../assets/images/image.svg')} />
</TouchableOpacity>
```

```
<TouchableOpacity style={[s.drawerBtn, s.center]}
activeOpacity={0.8}
onPress={() => {
pickVideo()
setButtons(!buttons)
}}
>
<SvgUri width="30" height="30"
source={require('../../../assets/images/video.svg')} />
</TouchableOpacity>
<TouchableOpacity style={[s.drawerBtn, s.center]}
activeOpacity={0.8}
onPress={() => {
pickDocument()
setButtons(!buttons)
}}
>
<SvgUri width="30" height="30"
source={require('../../../assets/images/mic.svg')} />
</TouchableOpacity>
<TouchableOpacity style={[s.drawerBtn, s.center]}
activeOpacity={0.8}
onPress={() => {
pickDocument()
setButtons(!buttons)
}}
>
<SvgUri width="30" height="30"
source={require('../../../assets/images/inbox.svg')} />
</TouchableOpacity>
<TouchableOpacity style={[s.drawerBtn, s.center]}
activeOpacity={0.8}
onPress={() => {
setButtons(!buttons)
setVanishModal(true)
//navigation.navigate('VanishModal')
}}
>
<SvgUri width="30" height="30"
source={require('../../../assets/images/clock.svg')} />
</TouchableOpacity>
</View>
:
<TouchableOpacity style={[s.orangeRoundBtn, s.center]}
activeOpacity={0.8}
onPress={() => setButtons(!buttons)}
>
<SvgUri width="30" height="30"
source={require('../../../assets/images/skrepa.svg')} />
</TouchableOpacity>
}</View>
```

У цьому фрагменті наведено створення історії

```
<Text style={[s.text15, s.factor]}>{text[this.state.lang].newPost}</T
ext>
</TouchableOpacity>
<TouchableOpacity style={[s.selector2Btn, s.flex1, s.center]}
activeOpacity={0.8}
//onPress={() => props.onSkip()}
>
<Text style={[s.text15, s.factor, s.textBlack]}>{text[this.state.lang
].newStory}</Text>
</TouchableOpacity>
</View>
<TouchableOpacity style={[s.btn56, s.center]}
activeOpacity={0.7}
onPress={() => {
this.state.navigation.goBack()
}}
>
<SvgUri width="25" height="25"
source={require('../../../assets/images/close.svg')} />
</TouchableOpacity>
<View style={[s.flex1]} />
<View style={[s.bottomControlls, s.flexRow, s.spaceBtw, s.aCenter, s.
mh15, platform ? s.ip10mb : s.mb50 ]}>
<TouchableOpacity style={[s.thumbBtnWr, s.center]}
activeOpacity={0.8}
onPress={this.pickImage}
>
<Image
style={[s.thumbBtn]}
source={require('../../../assets/images/thumb3.png')}
/>
</TouchableOpacity>
<TouchableOpacity style={[s.personIcon, s.center]}
activeOpacity={0.8}
onPress={this.__takePicture}
>
<AvaGradient style={[s.pabsolute]} />
<View style={[s.whiteRoundBtn]} />
</TouchableOpacity>
<TouchableOpacity style={[s.btn56, s.center, {width: 62}]}
activeOpacity={0.8}
onPress={() => this.setState({ cameraType: !this.state.cameraType })}
>
<SvgUri width="40" height="40"
source={require('../../../assets/images/camera.svg')} />
</TouchableOpacity>
</View>
```

СТОРІНКА MESSAGES

У цьому фрагменті коду наведено створення та опис чатів. У коді також показано можливість вибору документа для відправлення користувачу, з яким йдеться спілкування та відправлення.

```
return (<Bubble
{...props}wrapperStyle={{
right: {
backgroundColor: '#e7cecb',
borderRadius: 5,
//bottom: 20
},
left: {
backgroundColor: '#F8F1F0',
borderRadius: 5,
//bottom: 20
}
}}
textStyle={{
right: {
fontFamily: 'factor_a',
color: '#000',
},
left: {
fontFamily: 'factor_a',
color: '#000',
},
}}
/>
)
}
renderInputToolbar = (props) => {
return (
<View style={[{ backgroundColor: '#F8F1F0', bottom: 2 }]}>
<View style={[s.inputContainer, s.flexRow, s.alignCenter, s.mb20]}>
<TouchableOpacity style={[s.btn50, s.center, { bottom: 2 }]}
onPress={() => { this.pickDocument() }}
>
<SvgUri width="25" height="25"
source={require('../../../assets/images/orangeScrep.svg')} />
</TouchableOpacity>
{/* <Actions {...props}/> */}
<Composer {...props}
multiline={false}
textInputStyle={[{
color: '#000',
backgroundColor: '#F8F1F0',
// borderRadius: 10,
```

```
paddingHorizontal: 10,
// composerHeight: 40,
}]}
textInputProps={{
...props.textInputProps,
onSubmitEditing: () => {
if (props.text && props.onSend) {
props.onSend({ text: props.text.trim() }, true);
}
}
}}
placeholderTextColor={'#969896'}
placeholder={text[this.state.lang].urMess}
/>

<Send {...props}
textStyle={{
color: '#fff'
}}
containerStyle={[s.center]}
label={''}
children={
<TouchableOpacity style={[s.btn50, s.center]}
onPress={() => {
if (props.text && props.onSend) {
props.onSend({ text: props.text.trim() }, true);
}
}}
onLongPress={() => {
this.setState({ priceModal: true })
}}
>
<SvgUri width="25" height="25"
source={require('../../../assets/images/send2.svg')} />
</TouchableOpacity>
}
/>
</View>
```

СТОРІКА PROFILE

У цьому фрагменті коду наведено створення головної інформації на сторінки профілю користувача, а саме

- Фотографія профілю користувача
- Його ім’я та нікнейм
- Його хоббі (додаткова інформація)
- Опис користувача (додаткова інформація)

```
<Text style={[s.text14, s.factorBold, s.textBlack, s.mh15]}>{user.nam
e}</Text>
<Text style={[s.text14, s.factor, s.textGrey, s.mt5]}>{user.nick}</Te
xt>
</Animated.View>
</View>
<Animated.View style={[s.mt15, {
//opacity: storyOpacity,
//height:
}]}>
<Text numberOfLines={1} style={[s.hobby, s.text14, s.factor, s.textBl
ack, s.mh15]}>{'Modeling | Travel | Skin | Daily Routines...'}</Text>
<TouchableOpacity style={[s.h25, s.jCenter, s.mb15]}
activeOpacity={0.6}
onPress={() => {
!bio && scrolls.current?.scrollTo({ x: 0, y: 0, animated: true })
setBio(!bio)
}}
>
{
bio
? <Text style={[s.text14, s.factor, s.textOrange, s.mh15]}>{text[lang
].hide}</Text>
: <Text style={[s.text14, s.factor, s.textOrange, s.mh15]}>{text[lang
].redmor}</Text>
}
</TouchableOpacity>
</Animated.View>
</Animated.View>
<ScrollView style={[]}
ref={scrolls}
showsVerticalScrollIndicator={false}
onScroll={
Animated.event([
{ nativeEvent: { contentOffset: { y: scrollOffsetY } } }
],
```

```
{ useNativeDriver: false })}
scrollEventThrottle={16}
>
<View style={[{
paddingTop: H_MAX_HEIGHT
}]}>
{
bio
? <Text style={[s.text14, s.factor, s.textLine21, s.textBlack, s.mh15
, s.mb15]}>{
'(Bio)Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do
 eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim
ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut al
iquip ex ea commodo consequat.'
}</Text>
: null
}
```

У цьому фрагменті коду наведено публікацію постів користувача, які він викладає.

```
return (
<View key={index} style={[s.post, s.mt25]}>

<View style={[s.groupView, s.flexRow, s.spaceBtw, s.aCenter, s.ml15]}
>
<View style={[s.flexRow, s.aCenter]}>
<Image
style={[s.image50r, s.mr10]}
source={group.ava}
/>
<View style={[]}>
<Text style={[s.text14, s.factorBold, s.textBlack, s.mt3]}>{group.nam
e}</Text>
<Text style={[s.text14, s.factor, s.textGrey2, s.mt3]}>{group.nick}</
Text>
</View>
</View>

<View style={[s.flexRow, s.aCenter]}>
<Text style={[s.text14, s.factor, s.textGrey2]}>{post.time}</Text>
<TouchableOpacity style={[s.btn40, s.center]}
activeOpacity={0.8}
onPress={(post) => {
setBlogModal(true)
setPost(post)
}}
>
<SvgUri width="5" height="15"
source={require('../../../assets/images/3dots.svg')} />
```

```
</TouchableOpacity>
</View>
</View>

<TouchableOpacity style={[s.textView, s.mh15, s.mt15]}
activeOpacity={1}
onPress={() => setFull(!full)}
>
<Text numberOfLines={full ? null : 3} style={[s.text14, s.textLine21,
 s.factor, s.textBlack]}
onTextLayout={({ nativeEvent: { lines } }) => {
setLines(lines.length)
}}
>{post.text}</Text>
{
full
?
null
: <View style={[]}>
{
lines > 3
? <Text style={[s.h15, s.text14, s.factor, s.textOrange]}>{text[lang]
.redmor}</Text>
: null
}
</View>
}
```

ДОДАТОК В

ПРОГРАМНА РЕАЛІЗАЦІЯ ПІДКЛЮЧЕННЯ API

БЛОК АВТОРИЗАЦІЇ ТА РЕЄСТРАЦІЇ

У цьому фрагменті коду були реалізовані сховища даних користувача та обгортка для виклику методу login із збереженням отриманого токена.

```
private setToken(token: string): void {
if (token) {
AsyncStorage.setItem('token',token);
console.log("settoken"+token);
this.api = new Api(token);
} else {
AsyncStorage.removeItem('token')
this.api = new Api();
}
}
public async login(login: string, password: string) {
const result = await this.api.login(login, password);
console.log(result);
this.setToken(result);
return result;
}
```

У цьому фрагменті коду був реалізований метод виклику API логін з подальшою валідацією отриманої відповіді.

```
public async login(email: string, password: string) {
console.log({
email,
password
});
const res = await this.elAxios.post('/auth/token/login/', {
email,
password
```

```
});
if (res.status !== 200) throw res;
const profile: TokenVerify = res.data; // deserialize(res.data,
TokenVerify);
this.token = profile.auth_token;
return this.token;
}
```

У цьому фрагменті коду були реалізовані сховища даних користувача та обгортка для виклику методу register із збереженням отриманого токена.

```
public async register(email: string, username: string, password:
string) {
const result = await this.elAxios.post('/user/create-user/', {
email,
username,
password
});
if (result.status !== 201) {
throw 'Registration error!';
return result;
}
```

У цьому фрагменті коду був реалізований метод виклику API реєстрації з подальшою валідацією отриманої відповіді.

```
public async register(email: string, username: string, password:
string) {
const result = await this.api.register(email, username, password);
this.setToken(result.data);
}
```

ОТРИМАННЯ ДАНИХ КОРИСТУВАЧА

У цьому фрагменті коду був реалізований метод API для отримання даних про користувача.

```
public async getProfile() {
this.dataProfile = await this.api.getProfile();
return this.dataProfile;
}
```

У цьому фрагменті коду були реалізована обгортка методу API для отримання даних про користувача і збереження їх в пристрої.

```
public async getProfile() {
const res = await this.elAxios.get('user/get-user/');
if (res.status !== 200) {
throw 'Error!';
}
const result: UserGet = res.data;
return result;
}
```

ПОНОВЛЕННЯ ДАНИХ КОРИСТУВАЧА

У цьому фрагменті коду був реалізований метод API для отримання поновлення даних про користувача.

```
public async userUpdate(email: string, username: string, password:
string, data: any) {
const dataSend: any = {
email,
username,
password
};
for (const key in data) {
dataSend[key] = data[key];
}
```

```
const result = await this.elAxios.put(user/partial-update-user/,
dataSend);

if (result.status !== 200) {

throw 'Error!';

}

return result;

}
```

У цьому фрагменті коду були реалізована обгортка методу API для отримання поновлених даних про користувача і збереження їх в пристрої.

```
public async updateUser(password: string, data: any) {await
this.api.userUpdate(this.dataProfile.email,
this.dataProfile.username, password, data);

}
```

ФОРМУВАННЯ СТРІЧКИ КОРИСТУВАЧА

У цьому фрагменті коду був реалізований метод API для формування стрічки користувача.

```
public async mainPage() {

const result = await this.elAxios.get(blog/get-main-page/);

return result;

}
```

У цьому фрагменті коду була реалізована обгортка методу API для формування стрічки користувача

```
public async mainPageGet() {

const result = await this.api.mainPage();

this.main_page = result.data;

return result;

}
```

СТВОРЕННЯ ПУБЛІКАЦІЙ

У цьому фрагменті коду був реалізований метод API для створення публікацій

```
public async postCreate(
time_to_archive: number,
reply_link: string,
name: string,
description: string,
price_to_watch: number,
enabled_comments: boolean,
user: number,
attachments: number[],
favourites: number[],
data: any
) {
const dataSend: any = {
time_to_archive,
reply_link,
name,
description,
price_to_watch,
enabled_comments,
user,
attachments,
favourites,
};
for (const key in data) {
dataSend[key] = data[key];
}
const result = await this.elAxios.post(blog/create-post/, data);
if (result.status !== 202) {
throw "Error!";
```

```
}
return result;
}
```

У цьому фрагменті коду була реалізована обгортка методу API для створення публікацій

```
public async postCreate(
time_to_archive: number,
reply_link: string,
name: string,
description: string,
price_to_watch: number,
enabled_comments: boolean,
user: number,
attachments: number[],
favourites: number[],
data: any
) {const result = await this.api.postCreate(
time_to_archive,
reply_link,
name,
description,
price_to_watch,
enabled_comments,
user,
attachments,
favourites,
data);}
```

ОТРИМАННЯ ДІАЛОГІВ КОРИСТУВАЧА

У цьому фрагменті коду був реалізований метод API для отримання діалогів користувача

```
public async getUserDialogs() {
const result = await this.api.getUserDialogs();
console.log(result);
}
```

У цьому фрагменті коду була реалізована обгортка методу API для створення публікацій

```
public async getUserDialogs() {
const result = await this.elAxios.post(get-user-dialogs/);
return result.data;
}
```

ОТРИМАННЯ ПОВІДОМЛЕНЬ КОРИСТУВАЧА

У цьому фрагменті коду був реалізований метод API для отримання повідомлень в діалогах

```
public async getChatMessages(
room_id: number,
message_id: number,
data: any
) {
const dataSend: any = {
room_id,
message_id,
};
for (const key in data) {
dataSend[key] = data[key];
}
```

```
const result = await this.elAxios.post(chat/get-chat-messages/,
data);

return result.data;

}
```

У цьому фрагменті коду була реалізована обгортка методу API для отримання повідомлень в діалогах

```
public async getChatMessages(

room_id: number,

message_id: number,

data: any

) {

const result = await this.api.getChatMessages(

room_id,

message_id,

data

);

console.log(result);

}
```