Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня
Бакалавр

(назва освітнього ступеня)

на тему: **Розробка крос платформного додатку обміну порадами з використанням ШІ-модерації контенту**

Виконав(ла): студент(ка) 4 курсу, групи СПс-41
спеціальності 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

		Петров Ю.С.
	(підпис)	(прізвище та ініціали)
Керівник		Бачинський М. В.
	(підпис)	(прізвище та ініціали)
Нормоконтроль		Стоянов Ю. М.
	(підпис)	(прізвище та ініціали)
Завідувач кафедри		Петрик М.Р.
	(підпис)	(прізвище та ініціали)
Рецензент		
	(підпис)	(прізвище та ініціали)

Тернопіль
2026

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ _____
(підпис) (прізвище та ініціали)
« » 20___ р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавра
(назва освітнього ступеня)

за спеціальністю 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

студенту Петрову Юрію Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка кросплатформного додатку обміну порадами з використанням ШІ-модерації контенту

Керівник роботи _____
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «___» _____ 20___ року № _____

2. Термін подання студентом завершеної роботи _____

3. Вихідні дані до роботи Предметна область, завдання, вимоги та специфікація, програмне рішення, методичні вказівки

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступна частина
Аналіз предметної області та теоретичних основ
Визначення методики реалізації моделі
Реалізація моделі
Визначення основних аспектів охорони праці та безпеки життєдіяльності
Висновки роботи

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
Слайди презентації та діаграми процесів

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності,			
основи охорони праці			
Нормоконтроль			

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1			
2			
3			
4			
5			
6			
7			
8			
9			

Студент _____ Петров Ю. С.
(підпис) (прізвище та ініціали)

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

АНОТАЦІЯ

Кваліфікаційна робота бакалавра, виконав Петров Юрій Сергійович, студент групи СПс-41, на тему «Розробка крос-платформного мобільного застосунку для анонімного обміну порадами з інтеграцією штучного інтелекту». Робота виконана мовами програмування Dart та Python з використанням фреймворків Flutter і FastAPI. Робота має обсяг 62 сторінок, включає 15 рисунків, 5 таблиць та бібліографія з 24 посилань.

Метою роботи є проектування та розробка розподіленої системи, що складається з мобільного клієнта, серверної частини та підсистеми штучного інтелекту, і реалізує платформу для обміну текстовими порадами між користувачами з автоматичною модерацією та персоналізацією контенту.

У роботі спроектовано трирівневу клієнт-серверну архітектуру: мобільний застосунок на Flutter із патерном BLoC та локальним кешуванням Hive; серверна частина на FastAPI з розгортанням на Google Cloud Run із підтримкою Firebase Authentication, Cloud Firestore та RevenueCat; підсистема штучного інтелекту на BentoML, що включає сервіс визначення токсичності та NLP-сервіс із fine-tuned моделлю DistilBERT для класифікації тематики публікацій за 5 категоріями. Для персоналізованих рекомендацій реалізовано алгоритм пошуку на основі 768-вимірних векторів із зберіганням у базі даних Qdrant.

Ключові слова роботи: Flutter, FastAPI, Firebase, BLoC, NLP, рекомендаційна система, мобільний застосунок, модерація контенту.

ABSTRACT

Bachelor's qualification thesis completed by Petrov Yurii Serhiiovych, student of group SPs-41, on the topic "Development of a Cross-Platform Mobile Application for Anonymous Advice Sharing with Artificial Intelligence Integration". The work was implemented in the Dart and Python programming languages using the Flutter and FastAPI frameworks. The work is 62 pages long, includes 15 figures, 5 tables, and a bibliography of 24 references.

The aim of the work is to design and develop a distributed system consisting of a mobile client, a server-side component, and an artificial intelligence subsystem, implementing a platform for sharing text-based advice between users with automatic moderation and content personalization.

The work presents a three-tier client-server architecture: a Flutter mobile application utilizing the BLoC pattern and local caching via Hive; a FastAPI backend deployed on Google Cloud Run with support for Firebase Authentication, Cloud Firestore, and RevenueCat; an artificial intelligence subsystem built on BentoML, comprising a toxicity detection service and an NLP service with a fine-tuned DistilBERT model for classifying post topics into 5 categories. For personalized recommendations, a similarity search algorithm based on 768-dimensional vectors stored in a Qdrant database was implemented.

Keywords: Flutter, FastAPI, Firebase, BLoC, NLP, recommendation system, mobile application, content moderation.

ЗМІСТ

ВСТУП 6
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОГЛЯД ІСНУЮЧИХ РІШЕНЬ 8
1.1 Характеристика предметної області та постановка задачі 8
1.2 Огляд існуючих рішень та аналіз конкурентів 9
1.3 Обґрунтування вибору технологічного стеку 11
1.4 Вимоги до системи 12
2 ПРОЕКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ 15
2.1 Загальна архітектура клієнт-серверної системи 15
2.2 Варіанти використання 17
2.3 Ключові сценарії 18
2.4 Модель даних 21
3 РОЗРОБКА МОБІЛЬНОГО КЛІЄНТУ 23
3.1 Архітектура Flutter-додатку та патерн BLoC 23
3.2 Система навігації та маршрутизації 24
3.3 Ін'єкція залежностей та репозиторний патерн 25
3.4 Локальне кешування даних 27
3.5 Інтеграція з Firebase Authentication 28
3.6 Система підписок RevenueCat 29
4 РОЗРОБКА СЕРВЕРНОЇ ЧАСТИНИ 30
4.1 REST API на основі FastAPI 30
4.2 Робота з Firebase Firestore 32
4.3 Інтеграція NLP-сервісу модерації контенту 33
4.5 Обробка вебхуків RevenueCat 34
4.6 Векторна база даних Qdrant 35
5 ШІ КОМПОНЕНТИ СИСТЕМИ 36
5.1 Архітектура NLP-сервісу на BentoML 36
5.2 Fine-tuning моделі DistilBERT для визначення тематики порад 38
5.3 Алгоритм векторних рекомендацій 40
6 РОЗГОРТАННЯ СИСТЕМИ 44
6.1 Розгортання серверної частини на Google Cloud 44
6.2 Налаштування Firebase та хмарної інфраструктури 47
6.3 Публікація та тестування додатку 49

7 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ 51
7.1 Психологічні чинники небезпеки..... 51
7.2 Вимоги до профілактичних медичних оглядів для працівників ПК 54

ВСТУП

Сучасний розвиток мобільних технологій та штучного інтелекту відкриває нові можливості для побудови платформ взаємодопомоги між людьми. Ринок мобільних застосунків демонструє зростаючий попит на рішення, що поєднують соціальну взаємодію з персоналізованим контентом та автоматичним аналізом інформації.

Метою цієї роботи є проектування та розробка крос-платформного мобільного застосунку для анонімного обміну порадами, що інтегрує модель штучного інтелекту для модерації та класифікації контенту, систему персоналізованих рекомендацій на основі векторного пошуку, а також хмарну інфраструктуру для забезпечення масштабованості та надійності системи.

Об'єктом дослідження є процеси публікації, модерації та рекомендації текстового контенту на мобільних платформах.

Предметом дослідження є методи та засоби розробки мобільних застосунків з інтеграцією NLP-моделей і хмарних сервісів.

Для досягнення поставленої мети у роботі вирішуються такі завдання: аналіз предметної області та існуючих рішень; проектування архітектури розподіленої системи; розробка мобільного клієнта на Flutter з патерном BLoC; розробка серверної частини на FastAPI; побудова NLP-підсистеми на BentoML; розгортання системи на Google Cloud.

Практична цінність роботи полягає у створенні повністю функціонального застосунку, опублікованого в Google Play Store, що підтверджує промислову готовність розроблених рішень.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

У цьому розділі проводиться аналіз предметної області мобільних платформ для обміну порадами та формулюється задача дослідження. Здійснюється огляд наявних конкурентних рішень і визначаються їх ключові недоліки, що обґрунтовують актуальність розробки. На основі проведеного аналізу обирається технологічний стек та формулюються функціональні й нефункціональні вимоги до системи.

1.1 Характеристика предметної області та постановка задачі

Обмін досвідом та порадами є однією з фундаментальних потреб людини. Упродовж усієї історії люди зверталися до оточення за підтримкою в складних ситуаціях - стосовно стосунків, здоров'я, сімейних питань, навчання або особистого розвитку. Попри доступність традиційних каналів (сім'я, друзі, спеціалісти), вони мають суттєві обмеження: вузьке коло досвіду, соціальний тиск, брак анонімності та необхідність особистої взаємодії.

Поява інтернету та мобільних технологій відкрила нові можливості для анонімного спілкування між незнайомими людьми. Форуми, соціальні мережі та спеціалізовані платформи частково вирішили проблему, однак кожна з них має власні недоліки: відсутність мобільно-орієнтованого інтерфейсу, ручна або повільна модерація контенту, відсутність персоналізованих рекомендацій та незручний досвід взаємодії між автором ситуації та радником.

Поряд із цим спостерігається стрімке зростання популярності «випадкових» коротких стрічок контенту, таких як TikTok та Instagram Reels. Ключова особливість таких платформ - алгоритмічна непередбачуваність: користувач не обирає, що побачить наступним, а натомість отримує контент, підібраний системою. Такий підхід забезпечує високий рівень залученості та формує звичку регулярного використання.

У роботі ідентифіковано проблему, що полягає у відсутності мобільного застосунку, який би поєднував наративний формат публікацій (характерний для форумів типу Reddit) із захоплюючою стрічкою випадкового контенту (характерною для TikTok/Reels) та при цьому забезпечував автоматичну модерацію й категоризацію контенту за допомогою штучного інтелекту.

Метою роботи є розробка крос-платформного мобільного застосунку обміну порадами, що реалізує наступні функції:

1) публікацію анонімних «історій» — текстових описів життєвих ситуацій, що потребують поради;
2) надання порад іншими користувачами у форматі випадкової стрічки;
3) автоматичну AI-модерацію контенту для виявлення токсичних матеріалів;
4) автоматичну класифікацію тематики публікацій на основі fine-tuned моделі DistilBERT;
5) персоналізовані рекомендації на основі векторної подібності публікацій;
6) систему підписок для розширення можливостей користувача.

1.2 Огляд існуючих рішень та аналіз конкурентів

Reddit - одна з найбільших у світі платформ для обговорень, що нараховує понад 1,5 млрд відвідувань на місяць. Платформа організована у тематичні спільноти (subreddits), серед яких особливо популярні r/relationship_advice, r/AITA (Am I The Asshole), r/personalfinance та інші, де користувачі публікують описи своїх ситуацій і отримують поради від спільноти.

Переваги Reddit: велика та активна аудиторія, можливість анонімних публікацій, структурований формат дописів із заголовком та описом, розгалужені коментарі.

Разом із тим Reddit має ряд суттєвих недоліків з погляду задачі, розв'язуваної у цій роботі. Платформа розроблена з акцентом на десктопне використання; мобільний застосунок поступається за зручністю. Модерація контенту є переважно ручною - здійснюється волонтерами-модераторами, що не гарантує оперативності

та однорідності. Відсутній спеціалізований UX-потік для надання порад: користувач переглядає стрічку дописів у хаотичному порядку і не отримує конкретного стимулу залишити пораду. Алгоритм ранжування побудований на голосуванні (upvotes), а не на векторній подібності, що ускладнює побудову персоналізованих рекомендацій.

Quora — платформа у форматі «запитання–відповідь», де користувачі публікують запитання, а інші дають розгорнуті відповіді. Сервіс популярний серед аудиторії, що шукає експертні поради у професійних та академічних темах.

Недоліки Quora для розв'язуваної задачі: формальний і, як правило, неанонімний характер взаємодії (система заохочує прив'язку реального імені та фотографії); відсутність «стрічкового» UX для перегляду чужих ситуацій; контент зорієнтований на фактологічні відповіді, а не на особистий емоційний досвід; відсутня AI-модерація токсичності.

TikTok та Instagram Reels демонструють найвищий рівень залученості серед мобільних застосунків завдяки алгоритмічній стрічці «нескінченного прокручування». Механізм випадкового підбору контенту є основою їхнього успіху та моделлю для натхнення при проєктуванні стрічки у даній роботі.

Однак ці платформи є суто відеоформатними і не передбачають текстового обміну порадами. Структурована взаємодія «хтось описує ситуацію — хтось дає пораду» в них відсутня. Крім того, короткий формат (до 60–90 секунд) не підходить для детального опису життєвих ситуацій.

Wisdo та 7 Cups - застосунки взаємної підтримки, зосереджені переважно на ментальному здоров'ї. Вони передбачають більш глибокі, структуровані розмови та підтримку з боку навчених волонтерів або спеціалістів.

Недоліки: вузька тематична спрямованість, відсутність загального формату обміну порадами, обов'язкова реєстрація та профілювання, відсутність «ігрового» та випадкового характеру взаємодії.

1.3 Обґрунтування вибору технологічного стеку

Для розробки мобільного клієнта у роботі обрано фреймворк Flutter компанії Google. Flutter дозволяє компілювати єдину кодову базу мовою Dart у нативні застосунки для Android та iOS, забезпечуючи однаковий зовнішній вигляд та поведінку на обох платформах без використання WebView.

Порівняно з React Native, Flutter забезпечує вищу продуктивність завдяки власному графічному рушію Skia/Impeller та відсутності JavaScript-мосту. На відміну від нативної розробки (Kotlin/Swift), підхід Flutter скорочує витрати на розробку вдвічі, оскільки підтримується єдина кодова база.

Серверну частину розроблено на фреймворку FastAPI (Python). Вибір обґрунтований нативною асинхронністю на базі Python asyncio, що критично важливо для паралельного виконання запитів до NLP-сервісу та бази даних. FastAPI автоматично генерує OpenAPI-документацію та використовує Pydantic для валідації вхідних даних, що зменшує кількість помилок на рівні API.

Порівняно з Django, FastAPI є значно легшим і швидшим для побудови REST API без потреби в ORM та адмін-панелі. Порівняно з Flask, FastAPI надає вбудовані засоби валідації, типізацію та асинхронність без додаткових бібліотек.

У роботі використовується Firebase Authentication для управління обліковими записами (реєстрація через email, верифікація email, JWT-токени) та Cloud Firestore як основна база даних. Firestore є масштабованою NoSQL базою даних із підтримкою реального часу та автоматичним горизонтальним масштабуванням без необхідності адміністрування інфраструктури.

Порівняно з реляційними базами даних (PostgreSQL), Firestore не вимагає управління схемою міграцій, а Firebase Authentication усуває потребу у власній реалізації автентифікації, зберіганні паролів та обробці JWT.

Для задач класифікації тематики та генерації ембедингів у роботі обрано DistilBERT - спрощену версію моделі BERT,. DistilBERT містить 66 млн параметрів проти 110 млн у базовому BERT, при цьому досягає 97% точності BERT при швидкості виведення, вищій на 60%. Ця характеристика є критичною для

застосунку, де модерація контенту має відбуватися в режимі близькому до реального часу.

Модель fine-tuned на власному наборі даних для класифікації текстів за п'ятьма тематичними категоріями: General, relationships, family, mentalHealth, education.

Для розгортання NLP-моделей у роботі використовується фреймворк BentoML. BentoML спеціалізований для обслуговування ML-моделей і надає вбудовані засоби пакетної обробки запитів (batching), версіонування моделей та автоматичне формування HTTP API. Порівняно зі стандартним FastAPI, BentoML оптимізований для навантажень із паралельними запитами до ML-моделей.

Для реалізації системи рекомендацій у роботі використовується Qdrant - векторна база даних із підтримкою пошуку за косинусною подібністю. Кожна публікація після публікації отримує 384-вимірний вектор-ембединг (модель all-MiniLM-L6-v2), що зберігається у Qdrant. Рекомендації будуються на основі ембедингів публікацій, які сподобалися користувачу.

Qdrant обрано як open-source рішення з можливістю самостійного хостингу та ефективним Python SDK. Порівняно з Pinecone, Qdrant не вимагає платного хмарного провайдера.

Для реалізації системи підписок у роботі використовується SDK RevenueCat. RevenueCat спеціалізований для роботи з In-App Purchases та Google Play Billing / App Store billing і надає єдиний API для управління підписками на обох платформах. На відміну від прямої інтеграції з платіжними системами, RevenueCat автоматично обробляє оновлення, скасування та відновлення підписок через вебхуки.

1.4 Вимоги до системи

Функціональні вимоги:

1) Автентифікація та управління обліковим записом.

2) Система повинна забезпечувати реєстрацію, вхід та вихід користувача через email і пароль, а також підтвердження email-адреси. Доступ до функцій застосунку надається лише після верифікації email.
3) Користувач повинен мати змогу публікувати текстові описи життєвих ситуацій («історії») із заголовком та описом. Обсяг опису не повинен перевищувати 2500 символів.
4) Кожна опублікована історія повинна автоматично класифікуватися системою за тематичними категоріями: General, relationships, family, mentalHealth, education. Користувачу також надається можливість самостійно обрати або створити власну категорію.
5) Користувачу повинна відображатися стрічка чужих історій у випадковому порядку. До кожної історії користувач може залишити пораду або перейти до наступної. Поради, що не пройшли AI-модерацію токсичності, не зберігаються.
6) На основі вподобаних користувачем історій система повинна формувати персоналізовану добірку схожих публікацій засобами векторного пошуку.
7) Застосунок повинен підтримувати три рівні підписки: free, pro, premium, що відрізняються місячними лімітами на кількість порад, перегляд рекомендацій та кількість зірок для оцінювання авторів порад.
8) Користувачі можуть присвоювати зірки авторам порад, що формує публічний рейтинг найкращих радників.

Нефункціональні вимоги:

1) Застосунок повинен коректно працювати на пристроях під управлінням Android 8.0 і вище та iOS 13 і вище.
2) Застосунок розрахований на осіб віком від 13 років.
3) Час відповіді API при типовому навантаженні не повинен перевищувати 2 секунди. Модерація контенту відбувається у фоновому режимі засобами Google Cloud Tasks і не блокує UX.

4) Усі API-запити передаються каналом HTTPS. Автентифікація здійснюється через Firebase JWT-токени, що перевіряються на сервері при кожному запиті. Вебхуки RevenueCat захищені секретним токеном.
5) Серверна частина розгорнута на Google Cloud Run і масштабується автоматично залежно від навантаження. Firebase Firestore та Qdrant Cloud забезпечують горизонтальне масштабування сховища даних.
6) Усі списки публікацій та порад реалізовані з курсорною пагінацією з розміром сторінки 10 записів для обмеження навантаження на базу даних.

2 ПРОЕКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ

У цьому розділі описується проектування архітектури розробленої системи. Визначається загальна трирівнева структура клієнт-серверної взаємодії, будуються діаграми варіантів використання та послідовності для ключових сценаріїв, а також формалізується модель даних системи.

2.1 Загальна архітектура клієнт-серверної системи

У роботі розроблено трирівневу розподілену систему, що складається з мобільного клієнта, серверного рівня та рівня даних і зовнішніх сервісів. Такий підхід забезпечує чітке розмежування відповідальності між компонентами, незалежне масштабування кожного рівня та можливість заміни окремих компонентів без зміни інших.

Мобільний клієнт (Flutter) є єдиною точкою взаємодії кінцевого користувача з системою. Він відповідає за відображення інтерфейсу, управління локальним станом, кешування даних та виконання HTTP-запитів до серверного рівня. Усі запити до сервера супроводжуються Firebase JWT-токеном, що автоматично додається до заголовка Authorization.

Серверний рівень (FastAPI) — центральний компонент системи, що реалізує бізнес-логіку застосунку. Він виконує верифікацію JWT-токенів, оброблення запитів, координацію між сервісами та запис даних у Firestore. Серверна частина розгорнута на Google Cloud Run і масштабується автоматично.

Рівень даних і зовнішніх сервісів включає Firebase Firestore як основну базу даних, дві окремі BentoML-служби для NLP-обробки, Qdrant як векторну базу даних для рекомендацій, RevenueCat для управління підписками та Google Cloud Tasks для асинхронної обробки фонових задач.

Загальна схема взаємодії компонентів системи наведена нижче:

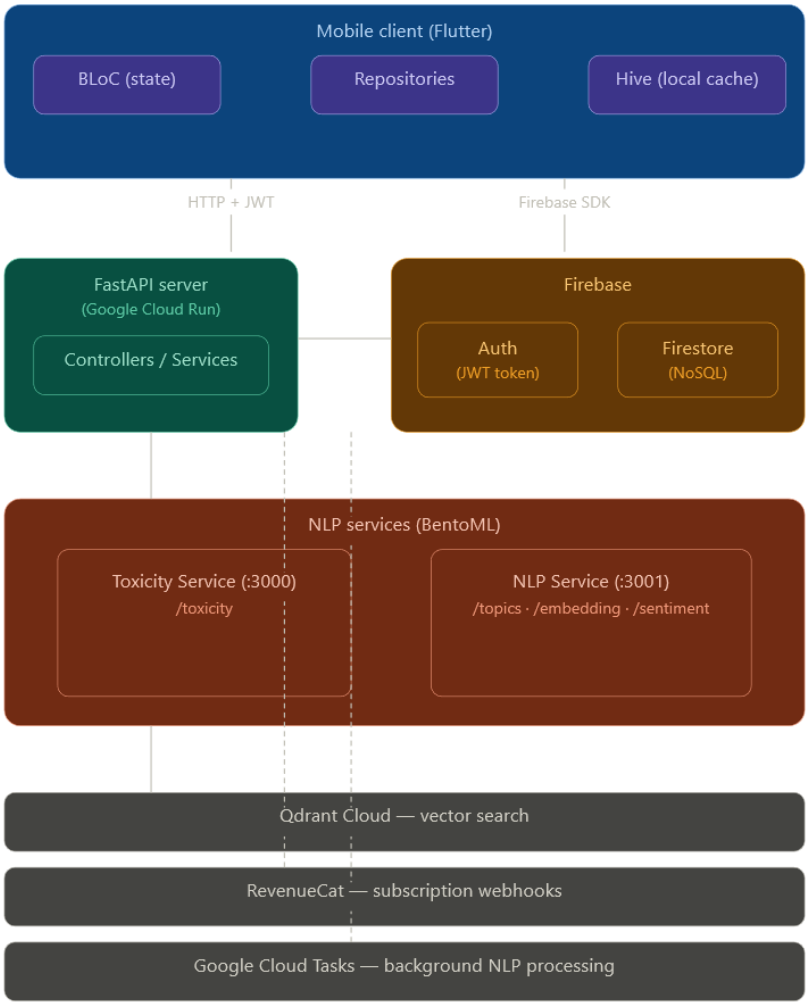

Рисунок 2.1 – Схема взаємодії компонентів системи

Потік даних при публікації історії включає наступні кроки: клієнт надсилає запит на сервер → сервер зберігає історію у Firestore зі статусом «pending» → сервер ставить задачу в чергу Google Cloud Tasks → у фоновому режимі Cloud Tasks викликає внутрішній endpoint сервера → сервер звертається до NLP-сервісу для класифікації тематики та генерації ембедингу → результати зберігаються у Firestore (поле category) та Qdrant (вектор для рекомендацій).

Потік даних при наданні поради такий: клієнт надсилає текст поради → сервер синхронно звертається до Toxicity Service → якщо текст є нетоксичним, порада зберігається у Firestore; інакше повертається помилка.

Такий поділ відповідальностей між синхронними (модерація поради) і асинхронними (обробка історії) операціями дозволяє не блокувати користувача під час ресурсоємної NLP-обробки.

2.2 Варіанти використання

На рисунку 2.1 зображено діаграму варіантів використання системи, що відображає взаємодію між трьома визначеними акторами - Неавтентифікованим користувачем, Авторизованим користувачем та Системою і доступними їм функціями.

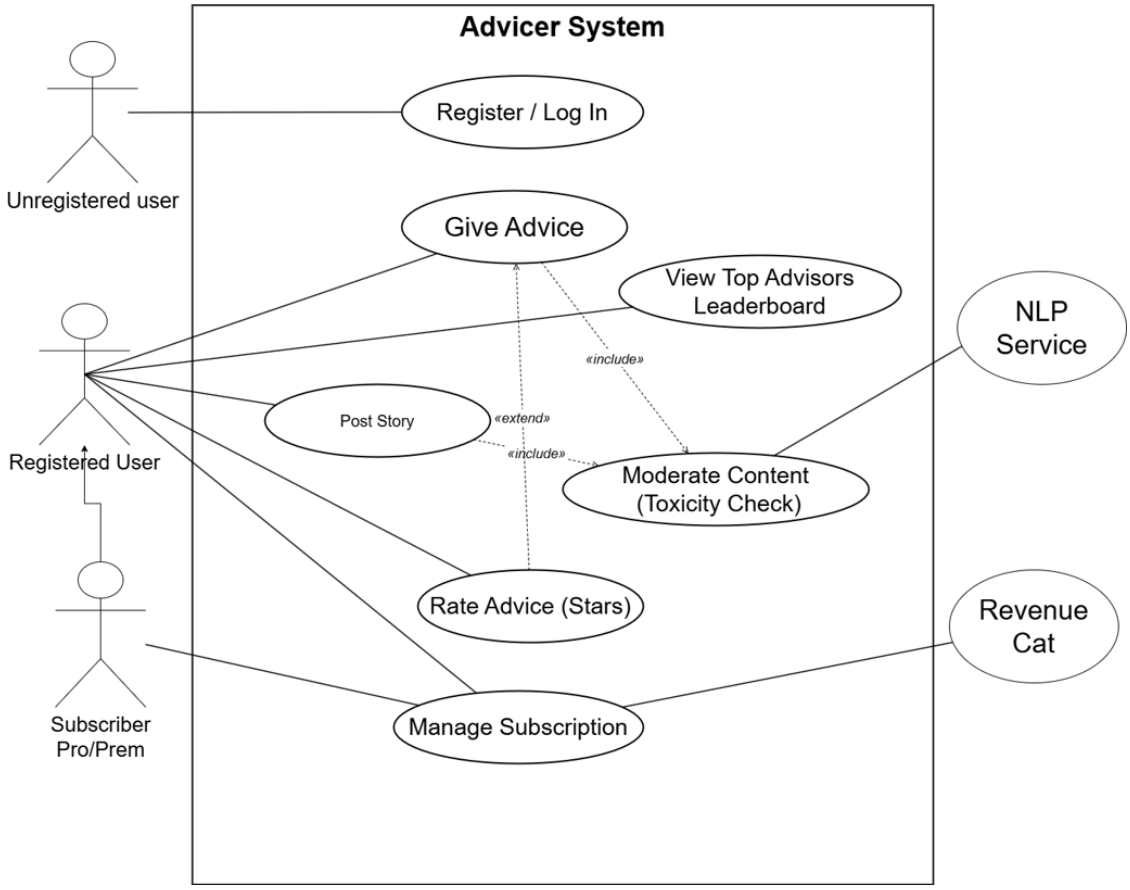

Рисунок 2.1 – Діаграма варіантів використання системи

Неавтентифікований користувач має доступ лише до двох варіантів використання: реєстрації (введення email та пароля з подальшим підтвердженням email-адреси) та входу до системи. Жоден інший функціонал застосунку не доступний до верифікації email.

Авторизований користувач отримує доступ до повного набору функцій: перегляду стрічки порад у випадковому порядку, надання порад до чужих історій, публікації власних історій, перегляду персоналізованих рекомендацій, оцінювання авторів порад зірками, перегляду профілю та управління підпискою. Кількість доступних щомісячних дій залежить від рівня підписки.

Таблиця 2.1 – Ліміти різних рівнів підписок

Дія	**Free**	**Pro**	**Premium**
Надання порад (на місяць)	25	100	300
Зірки для оцінювання (на місяць)	25	100	300

2.3 Ключові сценарії

У роботі реалізовано асинхронну обробку нових публікацій з метою мінімізації часу відповіді, що відчуває користувач.

Послідовність взаємодії:

Користувач заповнює форму (заголовок, опис, опціональна категорія) і натискає кнопку «Опублікувати».

Клієнт надсилає POST /api/v1/story з Bearer JWT-токеном.

Сервер верифікує токен через Firebase Auth.

Сервер перевіряє наявний ліміт публікацій у Firestore. Якщо ліміт вичерпано, то повертає помилку.

Якщо ліміт не вичерпано, сервер зберігає запис у колекції stories зі статусом active і зменшує лічильник availableStories.

Сервер ставить задачу до черги Google Cloud Tasks з ідентифікатором щойно створеної публікації.

Клієнт отримує відповідь із success: true і storyId — оновлений стан одразу відображається в UI.

Асинхронно (у фоні): Cloud Tasks викликає POST /internal/process-story на сервері.

Сервер звертається до NLP-сервісу через POST /process-story — отримує категорію, confidence та ембединг.

Сервер оновлює документ у Firestore (поля category, categoryConfidence).

Сервер зберігає ембединг у Qdrant з метаданими для подальшого пошуку за подібністю.

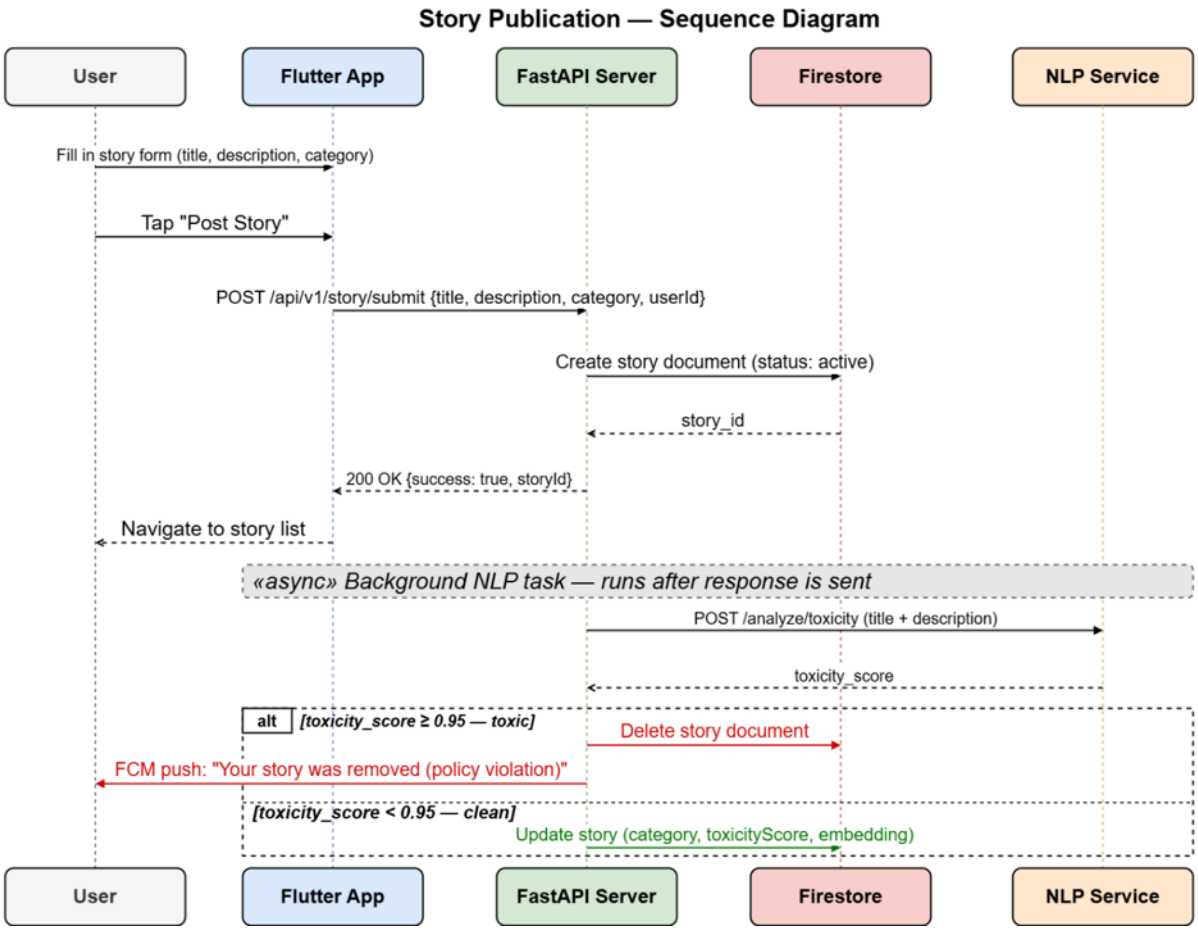

Рисунок 2.2 – Діаграма послідовності сценарію публікації історії

У роботі реалізовано асинхронну модерацію порад: порада зберігається негайно, а NLP-аналіз (перевірка токсичності та тональності) виконується у фоновому режимі і лише позначає запис відповідними мітками, не видаляючи його.

Послідовність взаємодії:

Користувач переглядає чужу історію і вводить текст поради в модальному bottom sheet.

Клієнт надсилає POST /api/v1/advice/submit з текстом поради та storyId.

Сервер верифікує JWT-токен та перевіряє ліміт availableAdvices. Якщо ліміт вичерпано - повертає помилку.

Сервер зберігає пораду у колекції advices і зменшує лічильник availableAdvices.

Клієнт отримує success: true з оновленим залишком availableAdvices і відображає SnackBar про успішну публікацію.

Асинхронно (у фоні, через FastAPI BackgroundTasks): сервер звертається до NLP-сервісу - виконується аналіз токсичності (POST /analyze/toxicity) та тональності (POST /analyze/sentiment).

Результати аналізу записуються у поле nlpAnalysis документа поради: isToxic, toxicityScore, sentimentLabel, sentimentScore.

Якщо isToxic: true - порада залишається у Firestore, але позначена відповідним прапорцем, що дозволяє фільтрувати або приховувати її на рівні відображення.

Рекомендації формуються на основі ембедингів публікацій, що сподобалися користувачу, з урахуванням вже переглянутих ним публікацій.

Послідовність взаємодії:

Клієнт надсилає POST /api/v1/story/recommendations з переліком ідентифікаторів вже переглянутих публікацій (seen_stories_id) та параметрами запиту (total, random_ratio).

Сервер отримує зі сховища ідентифікатори публікацій, що сподобалися користувачу (поле likedStories у документі Firestore).

Сервер звертається до Qdrant і отримує вектори відповідних публікацій.

Qdrant виконує батчевий пошук подібних публікацій (query_batch_points) за вилученими векторами, виключаючи вже переглянуті публікації.

Результати пошуку агрегуються та дедублікуються. Відповідно до параметра random_ratio до вибірки додається частка випадкових публікацій (для забезпечення різноманіття стрічки).

Сервер отримує повні дані публікацій із Firestore батчевим запитом.

Клієнт отримує список публікацій і відображає рекомендаційну стрічку.

2.4 Модель даних

У роботі використовується комбінація двох сховищ: Firebase Firestore для зберігання основних документів та Qdrant для зберігання векторних представлень. На рисунку 2.2 зображено діаграму класів моделі даних системи, де синім кольором виділено Firestore-колекції, зеленим — вбудовані об'єкти (embedded), жовтим — Qdrant-колекцію.

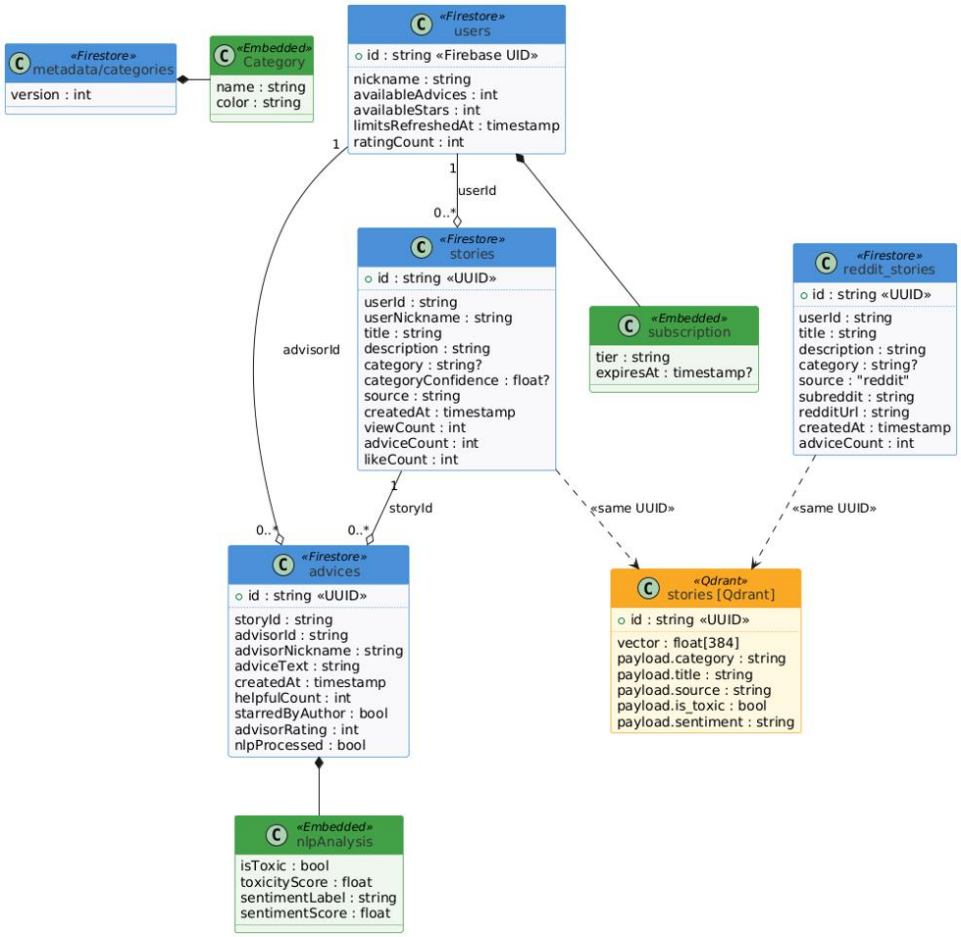

Рисунок 2.3 – Взаємодія між користувачем системи та інтерфейсом прогнозування

Центральною сутністю є колекція stories, що зберігає публікації користувачів. Кожен документ містить текстовий опис ситуації, заголовок, поля для лічильників активності (viewCount, adviceCount, likeCount), а також поля category та categoryConfidence, що заповнюються автоматично NLP-сервісом після

публікації. Колекція reddit_stories має аналогічну структуру і додатково містить поля subreddit та redditUrl — вона зберігає публікації, імпортовані з Reddit для наповнення бази контентом на початковому етапі.

Колекція advices зберігає поради, прив'язані до публікацій через поле storyId. Вбудований об'єкт nlpAnalysis містить результати фонового аналізу тексту поради: прапорець токсичності (isToxic), числовий показник токсичності (toxicityScore), тональність (sentimentLabel, sentimentScore). Поле nlpProcessed дозволяє відстежувати, чи вже оброблено конкретний запис.

Колекція users зберігає профіль кожного користувача. Ідентифікатор документа збігається з Firebase UID. Місячні ліміти (availableAdvices, availableStars) та дата їх оновлення (limitsRefreshedAt) зберігаються безпосередньо в документі. Інформація про підписку винесена у вбудований об'єкт subscription з полями tier (free, pro, premium) та expiresAt.

Колекція metadata/categories містить конфігурацію тематичних категорій системи. Кожна категорія представлена вбудованим об'єктом Category з полями name та color. Поле version на рівні документа дозволяє клієнту інвалідувати локальний Hive-кеш при оновленні переліку категорій.

Колекція stories [Qdrant] є дзеркальним представленням публікацій у векторній базі даних. Ідентифікатор точки збігається з Firestore document ID як для stories, так і для reddit_stories, що дозволяє ефективно поєднувати результати векторного пошуку з повними даними документів. Кожна точка зберігає 384-вимірний ембединг тексту (модель all-MiniLM-L6-v2) та метадані у полі payload, що дозволяє поєднувати семантичний пошук із фільтрацією за категорією, джерелом або токсичністю в межах одного запиту.

3 РОЗРОБКА МОБІЛЬНОГО КЛІЄНТУ

У цьому розділі описується реалізація мобільного клієнта застосунку на фреймворку Flutter. Розглядаються рішення щодо управління станом на основі патерну BLoC, організації навігації, ін'єкції залежностей та локального кешування даних. Окремо описується інтеграція з Firebase Authentication та реалізація системи підписок через RevenueCat.

3.1 Архітектура Flutter-додатку та патерн BLoC

У роботі мобільний клієнт розроблено на фреймворку Flutter з використанням патерну BLoC (Business Logic Component) для управління станом. Патерн BLoC забезпечує чітке розмежування між рівнем представлення (UI-віджети) та бізнес-логікою, що спрощує тестування та підтримку коду.

Кожна функціональна сторінка застосунку організована у власну директорію під lib/flow/<feature>/ і містить два підкаталоги: bloc/ для логіки та view/ для UI. Файли BLoC організовані за патерном part/part of: головний файл <feature>_bloc.dart є єдиною точкою входу, а файли подій (<feature>_event.dart) і станів (<feature>_state.dart) підключаються до нього через директиву part of. Такий підхід дозволяє логічно групувати пов'язані класи в одну одиницю компіляції.

Взаємодія між компонентами BLoC відбувається таким чином: UI-віджет додає події (Event) до BLoC через метод add(), BLoC обробляє їх за допомогою зареєстрованих обробників і випускає нові стани (State), на які реагують BlocBuilder або BlocConsumer у дереві віджетів. Всі класи станів розширюють Equatable для коректного порівняння об'єктів — це дозволяє уникнути зайвих перебудов UI, коли стан фактично не змінився.

BLoC отримує репозиторії через конструктор — ін'єкція залежностей відбувається на рівні get_it і описана у підрозділі 3.3.

На рисунку 3.1 зображено структуру патерну BLoC на прикладі AuthBloc — центрального компонента автентифікації застосунку.

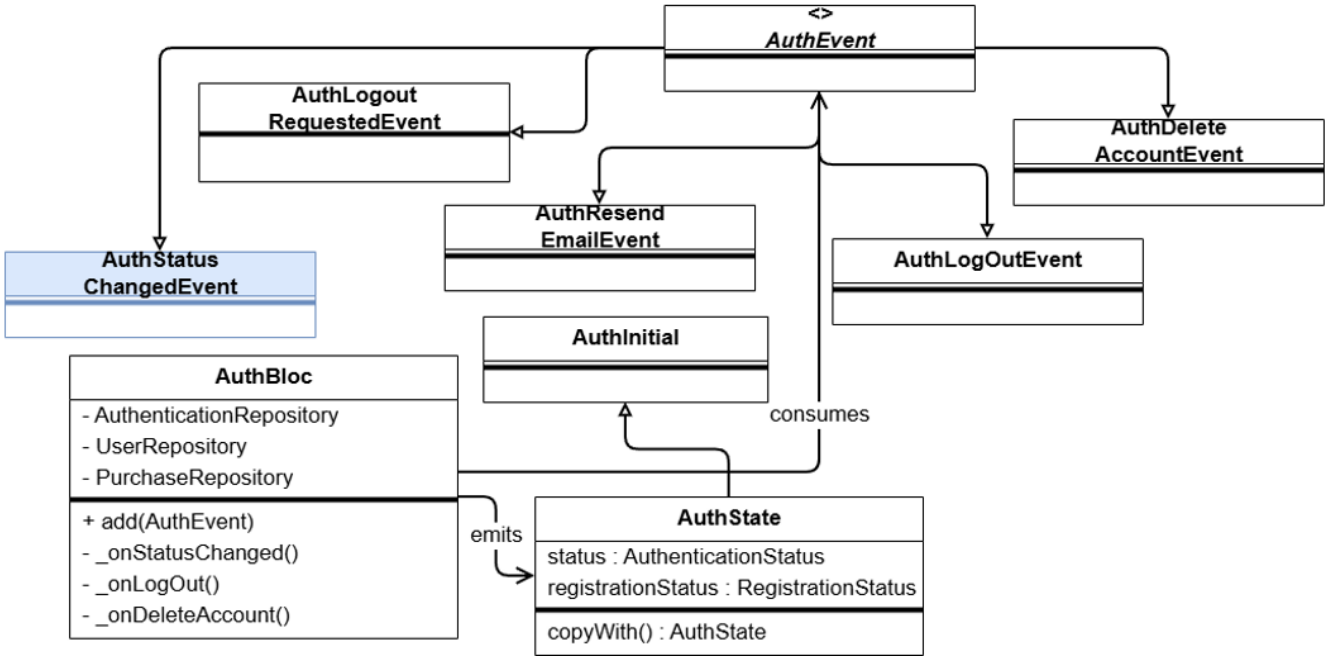

Рисунок 3.1 – Структура патерну BLoC на прикладі AuthBloc

3.2 Система навігації та маршрутизації

У роботі навігацію реалізовано за допомогою пакету go_router. Центральний клас AppRouter отримує екземпляр AuthBloc і конфігурує GoRouter із функцією перенаправлення (redirect), яка автоматично реагує на зміни стану автентифікації.

Зв'язок між AuthBloc та GoRouter забезпечується класом GoRouterRefreshStream: він підписується на потік станів BLoC і сповіщає роутер (через ChangeNotifier.notifyListeners()) про необхідність перерахувати перенаправлення щоразу, коли BLoC випускає новий стан.

Логіка перенаправлення побудована на основі чотирьох значень перелічення AuthenticationStatus:

1) unknown. Початковий стан при запуску; роутер направляє на сторінку-заставку (/splash).
2) unauthenticated. Користувач не увійшов; дозволений доступ лише до публічних сторінок (/login, /register, /confirm-email, /password-recover), решта маршрутів перенаправляють на /login.

3) emailVerificationNeeded. Користувач зареєстрований, але email не підтверджено; роутер направляє на /confirm-email.

На рисунку 3.2 зображено діаграму потоку навігації.

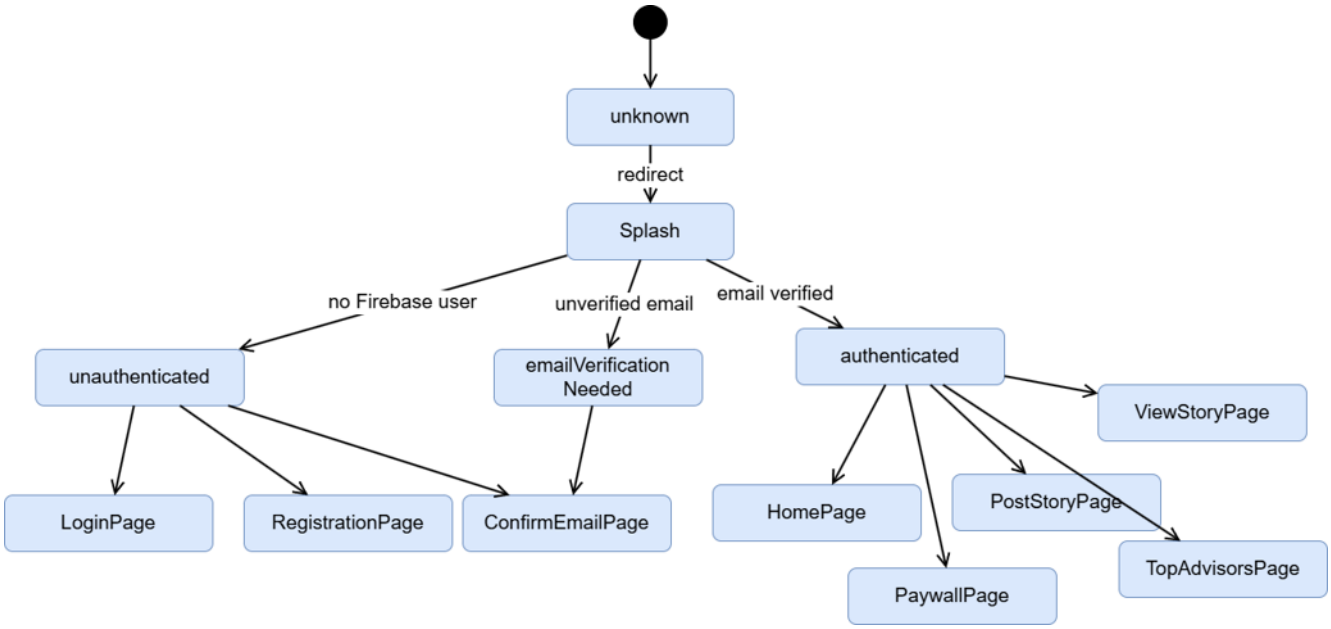

Рисунок 3.2 – Діаграма потоку навігації застосунку

Для переходів між сторінками у роботі реалізовано кілька типів анімацій (PageTransitionType): slideFromRight — для сторінок деталей публікації, slideFromBottom — для модальних сторінок (публікація нової історії, paywall), fadeIn — за замовчуванням. Тривалість переходу становить 300 мс.

Маршрути у коді зберігаються як константи в окремому файлі page_pathes.dart, що виключає використання рядкових літералів безпосередньо в коді навігації.

3.3 Ін'єкція залежностей та репозиторний патерн

У роботі для управління залежностями застосовано пакет get_it - service locator, що реєструє всі сервіси та репозиторії як синглтони при запуску застосунку. Уся ініціалізація зосереджена в єдиній функції setupDi() у файлі lib/di/setup_di.dart, що забезпечує єдину точку реєстрації залежностей.

Зареєстровані компоненти: ApiClient, UserStorage, SeenStoriesStorage, CategoryStorage, CategoryService, CategoryRepository, AuthenticationRepository, StoryRepository, AdviceRepository, UserRepository, DrawerService, RefreshService, PurchaseRepository.

Використання сервісу у BLoC або сервісі виконується через getIt<T>(), однак виключно поза методом build() віджетів — ін'єкція відбувається при конструюванні BLoC.

Кожен репозиторій у застосунку реалізується через абстрактний інтерфейс (наприклад, StoryRepository) та конкретну реалізацію (StoryRepositoryImpl). Такий підхід дозволяє підміняти реалізацію без зміни споживчого коду. Зокрема, для StoryRepository передбачено клас StoryRepositoryMock, що повертає фіктивні дані з налаштованою затримкою - він активується прапорцем _useMockStoryRepository у setup_di.dart без змін у решті коду.

На рисунку 3.3 зображено структуру репозиторного патерну та залежностей між компонентами.

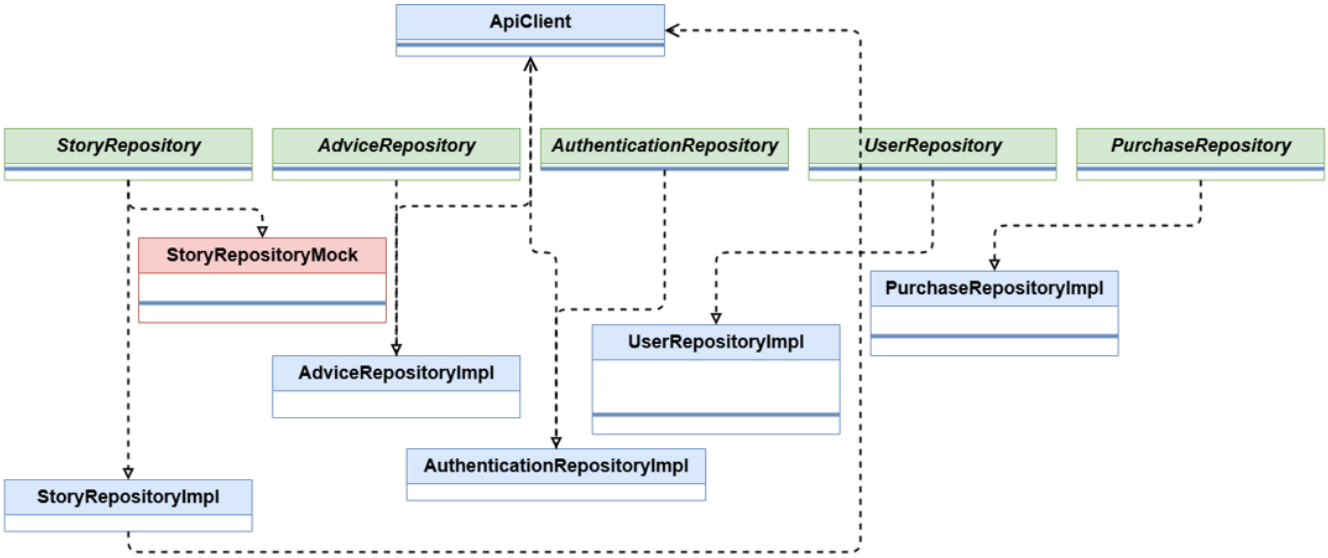

Рисунок 3.3 – Репозиторний патерн та структура залежностей

3.4 Локальне кешування даних

У роботі для локального зберігання даних застосовано пакет Hive - легковагову NoSQL базу даних для Flutter, що зберігає дані у бінарному форматі безпосередньо на пристрої. Hive не потребує окремого процесу і забезпечує синхронний доступ до даних після ініціалізації боксу.

В застосунку визначено три незалежні Hive-бокси з чітко розмежованою відповідальністю:

`UserStorage` (бокс `user`) зберігає дані облікового запису поточного користувача: залишки місячних лімітів (availableAdvices, availableStars), рівень підписки (subscriptionTier) та інформацію про лідера рейтингу для відображення у шторці. Дані оновлюються після кожної дії, що змінює ліміти — після надання поради або присвоєння зірки — без повторного звернення до сервера.

`SeenStoriesStorage` (бокс `seenStories`) зберігає список ідентифікаторів публікацій, які користувач вже переглянув у стрічці порад. Список передається на сервер з кожним запитом нових публікацій, щоб виключити вже бачені записи. Реалізовано FIFO-витіснення: при перевищенні ліміту у 30 записів найстаріші ідентифікатори видаляються автоматично.

`CategoryStorage` (бокс `categories`) кешує список тематичних категорій, завантажених із Firestore. Разом із категоріями зберігається поле version — при кожному запуску CategoryService порівнює локальну версію із серверною і оновлює кеш лише у разі розбіжності, скорочуючи кількість зайвих звернень до Firestore.

Усі три бокси ініціалізуються асинхронно на старті застосунку у функції setupDi() до реєстрації залежностей у get_it.

3.5 Інтеграція з Firebase Authentication

У роботі автентифікацію реалізовано на основі Firebase Authentication з підтримкою реєстрації та входу через email і пароль. Обов'язковою умовою отримання доступу до функцій застосунку є підтвердження email-адреси.

Клас AuthenticationRepositoryImpl інкапсулює Firebase Auth SDK і транслює зміни стану автентифікації у потік (Stream<AuthenticationStatus>). AuthBloc підписується на цей потік при ініціалізації і додає подію AuthStatusChangedEvent при кожній зміні, що і запускає перерахунок маршруту в GoRouter.

При успішній автентифікації AuthBloc виконує послідовність ініціалізаційних дій: ідентифікація користувача у RevenueCat (loginUser(uid)) для коректної синхронізації підписок, ініціалізація або оновлення документа користувача у Firestore з одночасним наповненням Hive-кешу (initUser()), завантаження актуального переліку категорій (categoryService.init()).

Усі HTTP-запити до FastAPI-сервера автоматично підписуються JWT-токеном Firebase. Клас ApiClient перед кожним запитом отримує актуальний токен через firebaseAuth.currentUser?.getIdToken() і додає його до заголовка Authorization: Bearer <token>. Термін дії токена становить одну годину;

Firebase SDK автоматично оновлює його у фоновому режимі. Окрім email/пароля, у роботі реалізовано вхід через Google OAuth за допомогою пакету google_sign_in. Метод signInViaGoogle() виконує таку послідовність: відкриває нативний вибір Google-акаунта отримує accessToken та idToken від Google формує GoogleAuthProvider.credential() передає його у _firebaseAuth.signInWithCredential(). Firebase автоматично створює новий обліковий запис, якщо користувач входить вперше, або прив'язує вхід до наявного, якщо акаунт із цим email вже існує. Оскільки Google гарантує верифікацію email-адреси, крок підтвердження пошти для Google-акаунтів не потрібен - AuthenticationStatus одразу переходить у authenticated. Інтерфейс AuthenticationRepository також визначає метод signInViaApple(), однак його реалізацію заплановано на наступний етап розробки.

3.6 Система підписок RevenueCat

У роботі монетизацію застосунку реалізовано через систему підписок із трьома рівнями: free, pro та premium. Управління підписками побудовано на SDK RevenueCat (purchases_flutter), що інкапсулює взаємодію з Google Play Billing та App Store.

Абстрактний інтерфейс PurchaseRepository визначає контракт взаємодії зі службою підписок і надає такі можливості: отримання доступних пакетів (getOfferings()), придбання підписки (purchasePackage()), відновлення раніше придбаних підписок (restorePurchases()), а також відкриття нативного екрану управління підпискою платформи (showManageSubscriptions()).

Ідентифікація користувача в RevenueCat прив'язана до Firebase UID: при вході викликається loginUser(uid), при виході — logoutUser(). Це забезпечує коректне зіставлення події покупки з документом користувача у Firestore на серверному боці.

Синхронізація стану підписки з Firestore відбувається через серверний вебхук: при будь-якій зміні підписки (покупка, поновлення, скасування, закінчення) RevenueCat надсилає подію на endpoint POST /webhooks/revenuecat FastAPI-сервера. Сервер обробляє подію в subscription_service.py, визначає новий рівень підписки та оновлює поле subscription.tier у документі користувача в Firestore, а також перераховує місячні ліміти відповідно до нового рівня. Клієнт отримує актуальний стан при наступному запиті до сервера або при ініціалізації після перезапуску застосунку.

4 РОЗРОБКА СЕРВЕРНОЇ ЧАСТИНИ

У цьому розділі описується реалізація серверної частини системи на базі фреймворку FastAPI. Розглядаються структура REST API, взаємодія з базою даних Firebase Firestore, інтеграція NLP-сервісу для модерації контенту та векторна база даних Qdrant для формування рекомендацій. Окремо розглянуто механізм обробки вебхуків RevenueCat для синхронізації стану підписок.

4.1 REST API на основі FastAPI

У роботі серверну частину розроблено на фреймворку FastAPI. Точкою входу є файл src/app.py, де визначається екземпляр FastAPI, реєструються middleware та підключаються роутери контролерів.

Усі маршрути API поділено на чотири групи з різними механізмами автентифікації:

Таблиця 4.1 - Структура контролерів

Префікс	**Контролер**	**Автентифікація**
`/api/v1/advice`	`advice.py`	Firebase JWT
`/api/v1/story`	`story.py`	Firebase JWT
`/api/v1/user`	`user.py`	Firebase JWT
`/webhooks/revenuecat`	`webhook.py`	Shared secret
`/internal`	`internal.py`	Cloud Tasks secret
`/health`	`app.py`	Без автентифікації

Маршрути під префіксом /api/v1 захищені залежністю get_current_user FastAPI Dependency Injection, що верифікує Firebase JWT-токен із заголовка Authorization Bearer і повертає об'єкт AuthenticatedUser(uid). При невалідному токені або невірифікованому email повертається HTTP 401.

У роботі підключено два middleware: CORSMiddleware для підтримки cross-origin запитів у режимі розробки та slowapi для rate limiting. Ліміти налаштовані на рівні окремих endpoint-ів і прив'язані до Firebase UID (а не IP-адреси).

При запуску застосунок виконує два кроки ініціалізації: init_firebase() - підключення Firebase Admin SDK та створення асинхронного Firestore-клієнта; init_http_client() - створення singleton HTTP-клієнтів для двох BentoML-сервісів із налаштованими пулами з'єднань. При завершенні викликається close_http_client() для коректного закриття з'єднань.

Контролери у роботі реалізовано як тонкий шар: вони відповідають лише за десеріалізацію запиту, виклик відповідного сервісу та формування HTTP-відповіді. Вся бізнес-логіка зосереджена у сервісах (advice_service.py, story_service.py, user_service.py). Прапорець USE_MOCK_SERVICES=true вимикає Firebase та Qdrant і підміняє їх in-memory заглушками для навантажувального тестування без реальної інфраструктури. На рисунку 4.1 зображено компонентну діаграму серверної частини.

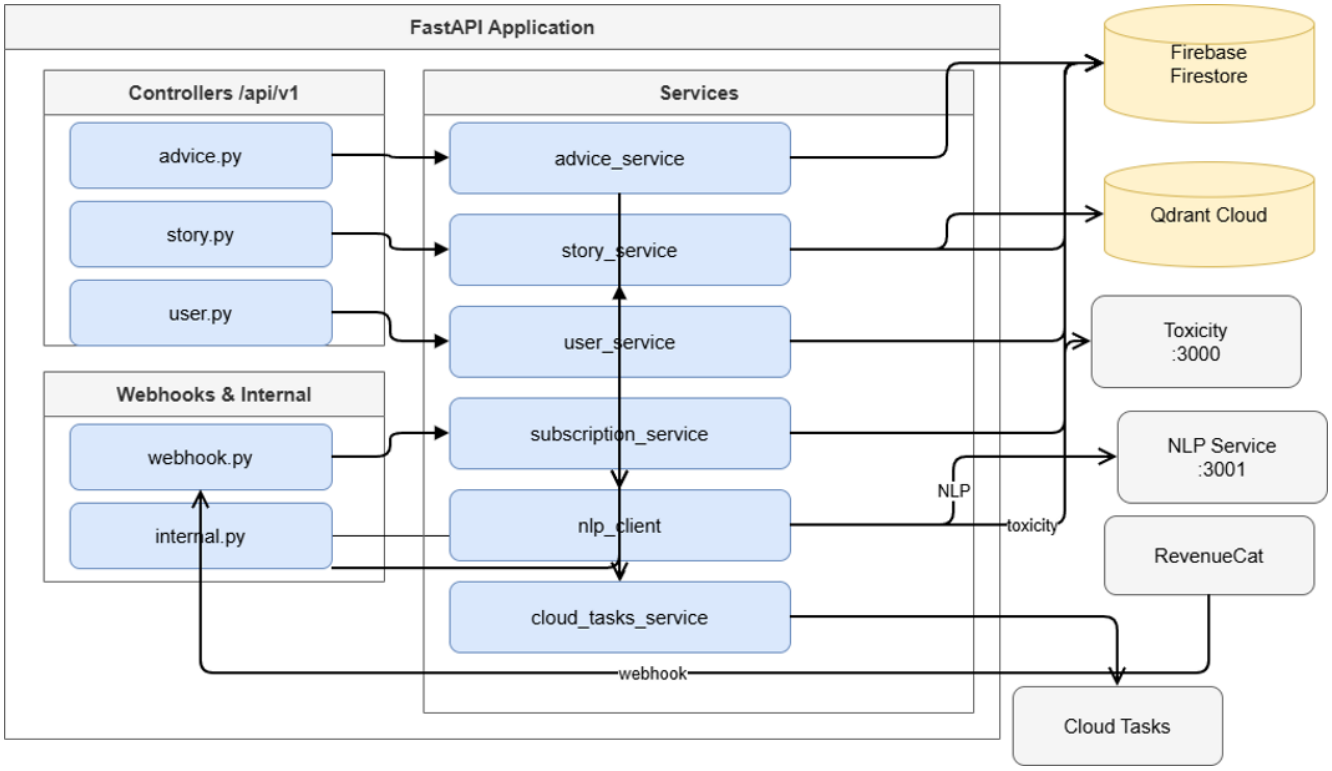

Рисунок 4.1 – Компонентна діаграма серверної частини

4.2 Робота з Firebase Firestore

У роботі всі операції з Firestore зосереджено в єдиному модулі firebase_service.py. Жоден контролер або сервіс не звертається до Firestore напряму - виключно через функції цього модуля. Такий підхід спрощує підміну реальних функцій mock-версіями.

Для роботи з Firestore у роботі використовується AsyncClient з бібліотеки google-cloud-firestore. Це забезпечує неблокуючий ввід-вивід у межах event loop FastAPI, що дозволяє одночасно обслуговувати велику кількість запитів без виділення окремих потоків.

Запит випадкових публікацій для стрічки порад реалізовано без сканування всієї колекції. Генерується випадковий UUID, після чого виконується запит із start_at([random_uuid]) за полем __name__ (document ID). Оскільки UUID-ідентифікатори рівномірно розподілені по простору значень, такий підхід ефективно вибирає документи з різних частин колекції.

Паралельно виконується другий запит «discovery query»: відбирається публікація з найменшою кількістю переглядів серед найновіших. Один слот у стрічці завжди резервується для такої публікації, що гарантує нещодавно доданим публікаціям шанс потрапити до користувачів незалежно від алгоритму.

Для зменшення кількості Firestore-читань при відображенні рейтингу авторів порад у роботі використовується in-memory TTL-кеш (cachetools.TTLCache) з вікном 5 хвилин та ємністю 1000 записів. Запис у кеші інвалідується при кожному присвоєнні зірки відповідному автору.

Для отримання повних даних публікацій після векторного пошуку використовується метод db.get_all(refs), що виконує один мережевий запит замість N послідовних читань. Результати фільтруються на наявність (doc.exists) і перетворюються у словники з доданим полем id.

4.3 Інтеграція NLP-сервісу модерації контенту

У роботі NLP-функціональність розподілена між двома окремими BentoML-сервісами, що дозволяє масштабувати та оновлювати їх незалежно. Toxicity Service (порт 3000) відповідає за перевірку тексту на токсичність через endpoint POST /analyze/toxicity. NLP Service (порт 3001) виконує класифікацію тематики (POST /analyze/topics), аналіз тональності (POST /analyze/sentiment), генерацію ембедингу (POST /embedding) та комбінований endpoint для обробки публікацій (POST /process-story).

Модуль nlp_client.py ініціалізує по одному singleton httpx.AsyncClient на кожен сервіс із налаштованими пулами keep-alive з'єднань (до 100 з'єднань, 20 keep-alive). Кількість одночасних звернень до NLP-сервісів обмежується через asyncio.Semaphore: окремий семафор для NLP-запитів (NLP_SEMAPHORE) та для генерації ембедингів (EMBEDDING_SEMAPHORE), що запобігає перевантаженню BentoML при пікових навантаженнях.

Функція analyze_text() виконує запити до обох сервісів паралельно через asyncio.gather(). Залежно від переданих прапорців (include_toxicity, include_sentiment, include_topics) формується список задач, що виконуються одночасно. Таким чином токсичність і тональність визначаються за час одного мережевого round-trip, а не послідовно.

При публікації нової історії NLP-обробка не блокує відповідь користувачу. В продакшн-середовищі (Cloud Run) задача ставиться у чергу Google Cloud Tasks через cloud_tasks_service.py: сервіс формує HTTP-задачу із захисним заголовком X-Cloud-Tasks-Secret і передає її в чергу GCP. Cloud Tasks викликає внутрішній endpoint POST /internal/process-story та виконує до трьох повторних спроб у разі невдачі. У режимі розробки, де Cloud Tasks недоступний, використовується FastAPI BackgroundTasks для аналогічного ефекту.

4.5 Обробка вебхуків RevenueCat

У роботі синхронізацію стану підписок між RevenueCat та Firestore реалізовано через вебхук-endpoint. Цей маршрут не використовує Firebase JWT і захищений спільним секретним ключем.

Верифікація запиту здійснюється через залежність _verify_revenuecat_secret, що порівнює значення заголовка Authorization із налаштованим значенням REVENUECAT_WEBHOOK_SECRET. Використання константно-часового порівняння запобігає timing-атакам. Якщо секрет не налаштований (режим розробки), перевірка пропускається з попередженням у лог.

Логіка обробки зосереджена у subscription_service.py. Сервіс класифікує події за двома групами.

Активаційні події (INITIAL_PURCHASE, RENEWAL, PRODUCT_CHANGE, UNCANCELLATION) - визначають новий рівень підписки через _resolve_tier(), що зіставляє ідентифікатори entitlement RevenueCat із внутрішніми значеннями free, pro, premium. Скидання місячних лімітів відбувається лише при INITIAL_PURCHASE та PRODUCT_CHANGE, при поновленні (RENEWAL) ліміти не скидаються, щоб не надавати незаслужену квоту посеред розрахункового циклу.

Деактиваційні події (EXPIRATION) - знижують рівень до free та скидають ліміти до квоти безкоштовного рівня. Ігноровані події (CANCELLATION, BILLING_ISSUE тощо) - логуються, але не спричиняють жодних змін: підписка залишається активною до настання EXPIRATION.

Endpoint завжди повертає HTTP 200, навіть для нерозпізнаних подій RevenueCat повторює доставку при будь-якому не-2xx статусі, тому повернення помилки призвело б до нескінченних повторів.

4.6 Векторна база даних Qdrant

У роботі для реалізації семантичного пошуку та персоналізованих рекомендацій використовується Qdrant Cloud. Усі операції з Qdrant зосереджені в модулі qdrant_service.py і виконуються через AsyncQdrantClient.

Колекція stories налаштована на косинусну відстань між 384-вимірними векторами. При запуску застосунок перевіряє існування колекції і створює її за потреби через ensure_collection_exists().

Кожна публікація зберігається у Qdrant як PointStruct з UUID-ідентифікатором, який береться з Firestore, вектором та метаданими у полі payload: категорія, заголовок, джерело (user або reddit), прапорець токсичності та тональність. Ідентичність ідентифікаторів між Firestore та Qdrant дозволяє за результатами векторного пошуку одразу виконувати батчеве читання повних документів із Firestore.

Для формування рекомендацій використовується метод search_similar_stories_batch(), що виконує кілька векторних запитів в одному мережевому зверненні через query_batch_points(). Кожен запит відповідає одній із вподобаних публікацій користувача. Пошук підтримує фільтрацію за exclude_ids (вже переглянуті публікації) та за полем payload.source для розмежування користувацького та Reddit-контенту.

Щоб рекомендаційна стрічка не ставала надто передбачуваною, story_service.py змішує результати векторного пошуку з випадковими публікаціями відповідно до параметра random_ratio. При значенні 0.5 половина стрічки формується на основі семантичної подібності, інша половина - випадково, що забезпечує баланс між персоналізацією та різноманіттям контенту.

5 ШІ КОМПОНЕНТИ СИСТЕМИ

У цьому розділі описуються компоненти ШІ модулю системи, що забезпечують автоматичну обробку контенту. Розглядається архітектура двох незалежних NLP-мікросервісів на базі BentoML, процес дообробки (fine-tuning) моделі DistilBERT для класифікації тематики публікацій, а також алгоритм побудови персоналізованих рекомендацій на основі векторної подібності.

5.1 Архітектура NLP-сервісу на BentoML

У роботі NLP-функціональність виокремлена в два незалежні мікросервіси, розгорнуті на фреймворку BentoML 1.4. Рішення про розподіл на два сервіси замість одного монолітного обумовлено різними характеристиками навантаження: сервіс токсичності обслуговує синхронні запити модерації порад із низькою частотою (один виклик на нову пораду), тоді як NLP-сервіс опрацьовує теми та ембединги у фонових задачах із вищою пропускною здатністю. Незалежне розгортання дозволяє масштабувати та оновлювати кожен сервіс без зупинки іншого.

Toxicity Service (порт 3000) побудований навколо бібліотеки Detoxify із моделлю "original" на базі BERT. Сервіс приймає масив текстів (list[str]) і повертає для кожного елемента набір числових оцінок: загальна токсичність, атака на ідентичність, образа, нецензурна лексика, погроза, сексуальний контент. Поріг isToxic = toxicityScore > 0.7 визначає бінарне рішення про блокування публікації.

Endpoint POST /analyze/toxicity оголошено бачним (batchable=True) із параметрами max_batch_size=32 та max_latency_ms=300. BentoML накопичує запити, що надходять одночасно від різних клієнтів, і передає їх у модель єдиним батчем - модель Detoxify нативно приймає список рядків, тому додаткового розпаралелювання не потрібно. Параметр concurrency: 1 у конфігурації трафіку обмежує кількість паралельних потоків виведення до одного: це усуває конкуренцію за GPU/CPU без втрати пропускної здатності завдяки батчингу.

Автентифікація реалізована через ASGI Middleware ApiKeyMiddleware: кожен HTTP-запит перевіряється на наявність заголовка x-api-key до того, як дістанеться до обробника. Якщо ключ відсутній або невірний, middleware повертає HTTP 403 і запит не доходить до моделі.

NLP Service (порт 3001) завантажує три моделі при запуску:

Таблиця 5.1 – Призначення моделей

Модель	**Призначення**	**Endpoint**
Fine-tuned DistilBERT (`story-classifier/`)	Класифікація теми публікації	`POST /analyze/topics`
`sentencetransformers/paraphrase-multilingual-mpnet-base-v2`	Генерація 768-вимірних ембедингів	`POST /embedding`
`cardiffnlp/twitter-roberta-base-sentiment-latest`	Визначення тональності тексту	`POST /analyze/sentiment`

Усі три endpoint-и оголошені батчованими з max_batch_size=32. Додатково реалізований комбінований endpoint POST /process-story, що виконує класифікацію тематики та генерацію ембедингу в одному запиті: це усуває зайвий мережевий round-trip при обробці нових публікацій (детально описано у підрозділі 4.3).

Для генерації ембедингів застосовується mean pooling із маскуванням padding-токенів: значення токен-ембедингів усереднюються з урахуванням attention_mask, після чого вектор нормалізується до одиничної L2-норми. Завдяки нормалізації косинусна відстань еквівалентна скалярному добутку, що безпосередньо відповідає метриці колекції Qdrant (Distance.COSINE).

Автентифікація виконується функцією _check_api_key(ctx) в тілі кожного обробника: функція порівнює значення заголовка x-api-key з очікуваним секретом і при розбіжності встановлює ctx.response.status_code = 403 та повертає порожній результат, не піднімаючи виключення. До сервісу підключено MonitoringMiddleware для збору метрик запитів.

Усі моделі Hugging Face завантажуються під час збірки Docker-образу через download_models.py, що гарантує їх наявність у кеші та унеможливлює затримки першого запиту на продакшн.

На рисунку 5.1 зображено компонентну діаграму ШІ-підсистеми.

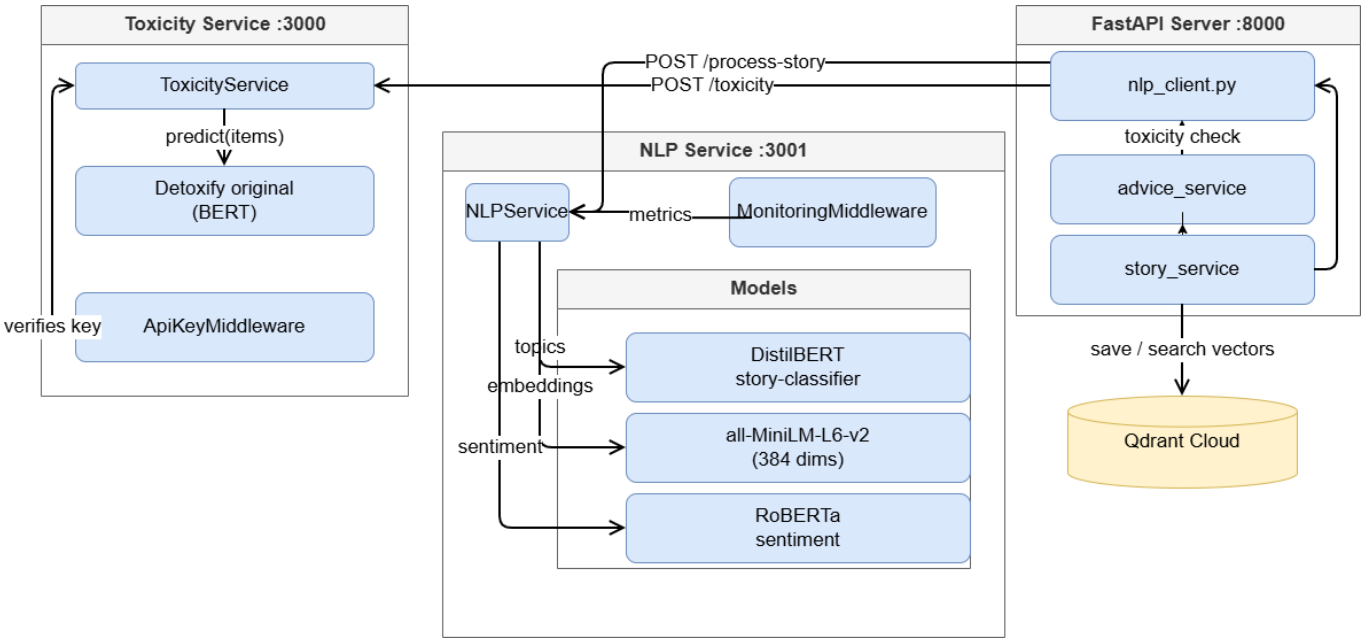

Рисунок 5.1 – Компонентна діаграма ШІ-підсистеми

5.2 Fine-tuning моделі DistilBERT для визначення тематики порад

Задача класифікації тематики публікацій вирішується як багатокласова класифікація з однією правильною відповіддю. Набір класів включає 5 тематичних категорій: relationships, family, mentalHealth, education— а також клас general для публікацій, що не вкладаються в жодну тематичну категорію.

На етапі прототипування у роботі була протестована модель нульового шоту MoritzLaurer/deberta-v3-base-zeroshot-v2.0. Підхід zero-shot classification не вимагає маркованих даних — модель отримує список текстових міток і визначає найбільш підходящу через механізм Natural Language Inference. Однак вимірювання на тестових прикладах показали затримку близько 14 000 мс на CPU, що є неприйнятним для фонової обробки сотень публікацій.

Для порівняння були обрані три архітектурно легші моделі: distilbert-base-uncased, google/mobilebert-uncased та albert-base-v2. Ці моделі придатні для дообробки (fine-tuning) під конкретну задачу, що дозволяє значно скоротити час виведення. За результатами бенчмарку на CPU обрано distilbert-base-uncased як базову архітектуру: DistilBERT є дистильованою (скороченою) версією BERT, що зберігає 97 % якості при скороченні кількості шарів удвічі (6 трансформерних блоків замість 12). Це забезпечує баланс між швидкістю (~91 мс на CPU після fine-tuning) та якістю класифікації.

Навчальний датасет сформовано з текстів публікацій із Reddit та призначених для тематичних категорій порталу. Кожен приклад являє собою пару (текст публікації, мітка категорії). Тексти попередньо очищались: переводились до нижнього регістру, видалялись URL-адреси, HTML-теги, емодзі та спеціальні символи. Ця функція _clean_text() використовується і в продакшн-сервісі — однаковий пайплайн гарантує відсутність train-serve skew (розбіжності між середовищем тренування та виведення).

Дообробка виконувалась через Hugging Face Trainer API із DistilBertForSequenceClassification. Базові ваги DistilBERT заморожені не були — повне fine-tuning на власних даних дозволяє адаптувати всі шари до специфіки тематичних текстів. Класифікаційна голівка (nn.Linear) додається поверх [CLS]-токена — стандартний підхід для задач класифікації послідовностей.

Результуюча модель збережена локально у директорії story-classifier/ і завантажується при запуску NLP-сервісу через Hugging Face pipeline("text-classification", top_k=None). Параметр top_k=None повертає оцінки впевненості для всіх категорій одночасно — це необхідно для endpoint POST /process-story, що передає поле allScores (словник усіх оцінок) разом із первинною категорією та ембедингом.

Після fine-tuning час виведення на CPU складає ~91 мс для одного тексту при max_length=512 токенів. Первинна категорія визначається як клас із найвищим показником впевненості (confidence). Значення confidence зберігається у Firestore у

полі categoryConfidence разом із самою категорією та використовується для аналітики якості моделі.

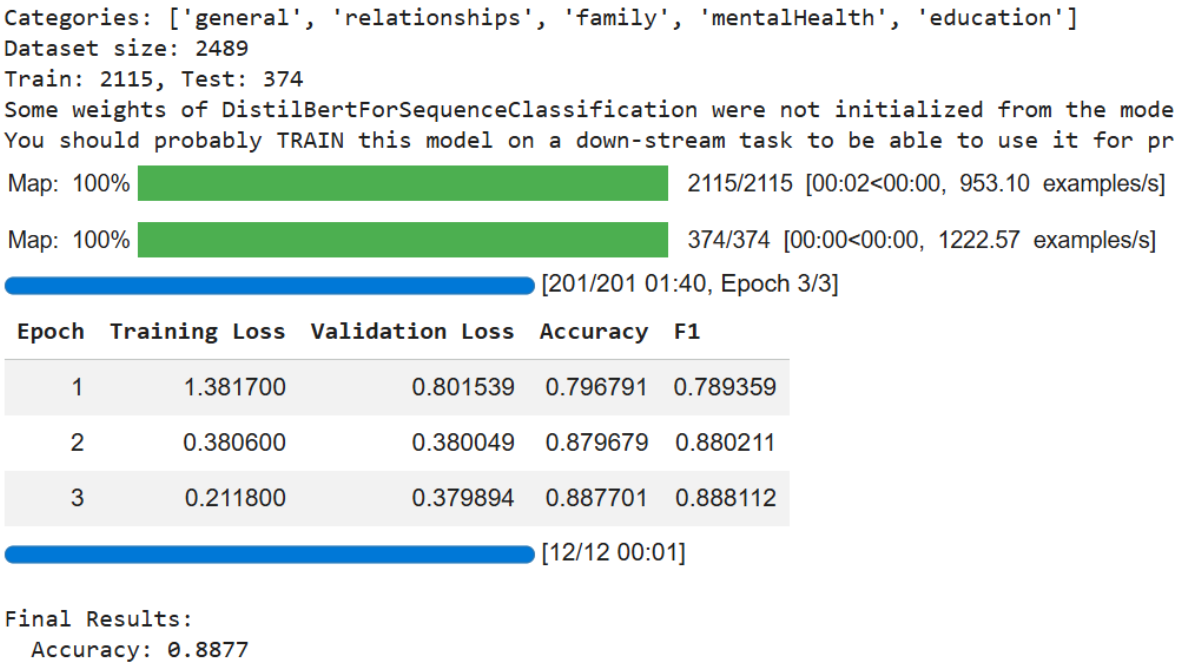

Рисунок 5.2 – FineTuning text Classification моделі

5.3 Алгоритм векторних рекомендацій

Система рекомендацій побудована на основі семантичної подібності між ембедингами публікацій. Кожна публікація при додаванні до системи проходить NLP-обробку: генерується 768-вимірний ембединг і зберігається у Qdrant разом із метаданими (категорія, заголовок, джерело). При запиті рекомендацій ембединги вподобаних публікацій користувача використовуються як запитні вектори.

Алгоритм реалізований у функції get_recommendations() у story_service.py та виконується у п'ять основних кроків.

Крок 1. Cold start. Якщо користувач ще не вподобав жодної публікації, система не може сформувати персоналізований профіль. У цьому випадку стрічка повністю формується з випадкових публікацій: спочатку з реального контенту (stories), а при нестачі — доповнюється Reddit-публікаціями.

Крок 2. Отримання ембедингів вподобаних публікацій. Для активного користувача з бази Qdrant завантажуються ембединги останніх N=15 вподобаних публікацій разом із їх метаданими (категорія). Одне звернення до Qdrant із методом retrieve(ids=[...], with_vectors=True) повертає всі ембединги в одному мережевому запиті.

Крок 3. Групування за категоріями та обчислення центроїдів. Ембединги розбиваються на групи за тематичними категоріями. Для кожної категорії обчислюється центроїд - покомпонентне середнє значення всіх ембедингів групи. Центроїд відображає усереднений «профіль» користувача в межах конкретної теми. Кількість рекомендацій, що виділяється кожній категорії, пропорційна її частці у вподобаних публікаціях. За рахунок семантичної близькості, центроїди кожної категорії будуть близькі у векторному просторі до семантично подібних історій див рис 5.3.

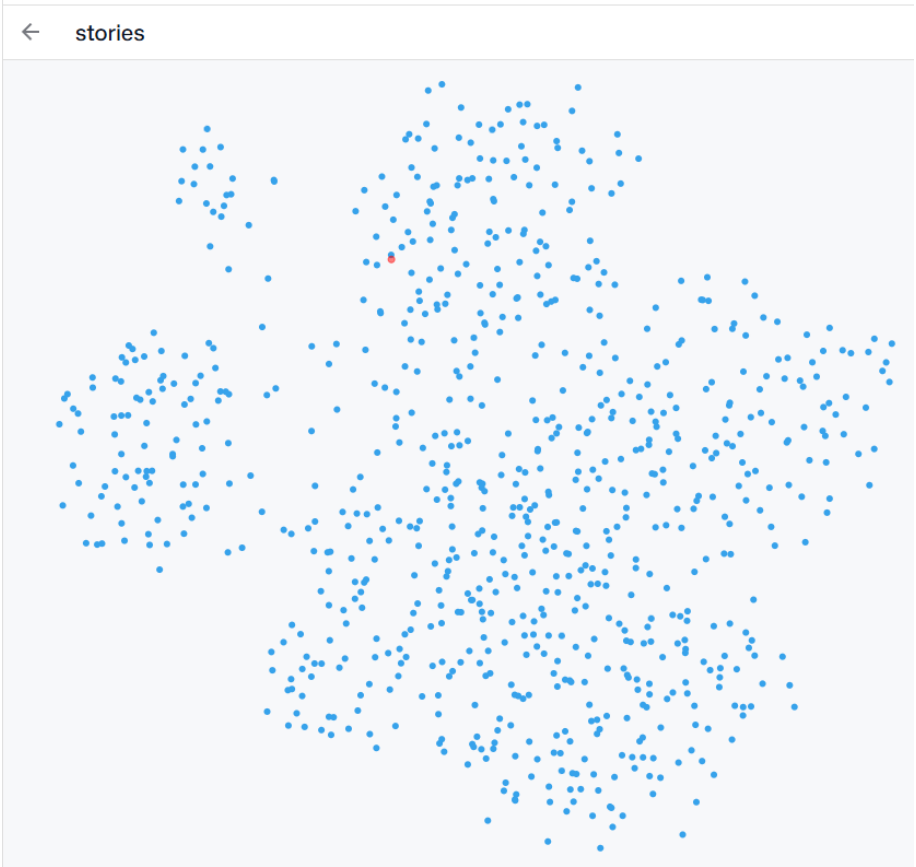

Рисунок 5.3 – Розміщення векторів історій у 768 вимірному просторі

Крок 4. Батчевий пошук у Qdrant. Центроїди всіх категорій передаються в єдиний виклик query_batch_points() через метод search_similar_stories_batch(). BentoML виконує всі пошукові запити в одному мережевому round-trip. При першому проходженні пошук обмежується реальними публікаціями користувачів (source_filter="exclude_reddit"). Якщо результатів недостатньо, виконується другий прохід лише серед Reddit-публікацій (source_filter="reddit").

Крок 5. Змішування з випадковими публікаціями. Параметр random_ratio (за замовчуванням 0.4) визначає частку випадкових публікацій у стрічці. При 40-відсотковому значенні з 10 запрошених публікацій 6 є персоналізованими, а 4 — випадковими. Завантаження випадкових публікацій запускається паралельно з персоналізованим пайплайном через asyncio.create_task() ще до отримання ембедингів із Qdrant, що скорочує загальний час відповіді.

Кешування рекомендацій

Для початкового завантаження стрічки (виклик без exclude_ids) результати кешуються в TTL-кеші _recommendations_cache з вікном 2 хвилини та ємністю 500 записів. Повторний запит одного й того ж користувача протягом вікна повертає кешовані дані без жодних читань із Firestore або Qdrant. Запити з непорожнім exclude_ids (користувач гортає стрічку і хоче нові публікації) завжди обходять кеш і виконуються повністю.

Ідентифікатори вже переглянутих публікацій та вподобань об'єднуються у множину all_excluded_ids і передаються у фільтр HasIdCondition(must_not=[...]) в Qdrant: це гарантує, що жодна раніше показана публікація не потрапить у стрічку повторно.

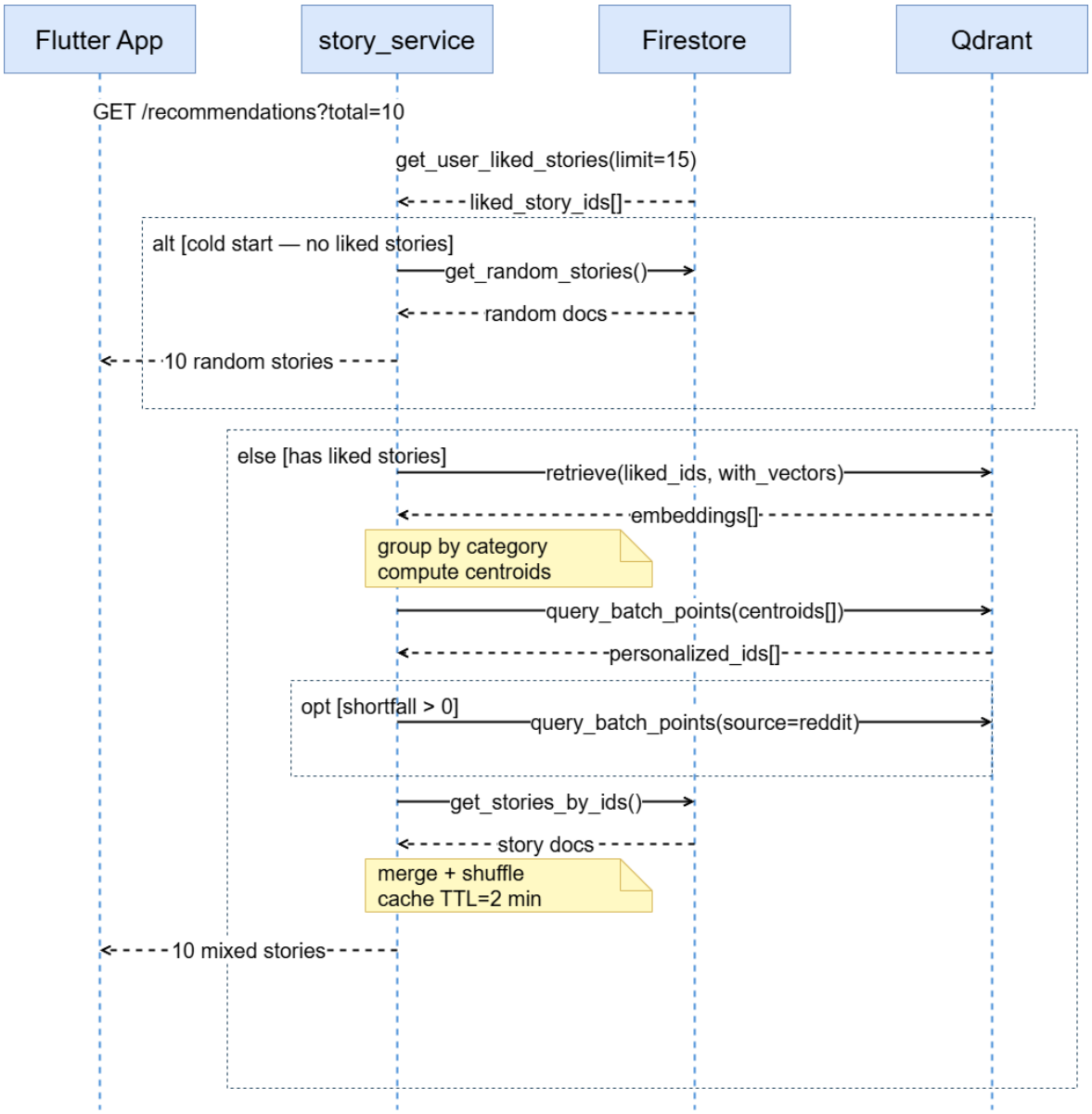

Рисунок 5.2 – Діаграма послідовності алгоритму векторних рекомендацій

6 РОЗГОРТАННЯ СИСТЕМИ

У цьому розділі описується процес розгортання всіх компонентів системи у хмарному середовищі. Розглядаються параметри конфігурації Google Cloud Run для серверної та NLP-частин, налаштування Firebase і допоміжної хмарної інфраструктури, а також процедура підготовки та публікації мобільного застосунку у Google Play Store.

6.1 Розгортання серверної частини на Google Cloud

У роботі для розгортання серверної частини обрано Google Cloud Run — керований безсерверний сервіс, що автоматично масштабує контейнери залежно від навантаження. Перевагами Cloud Run для даного проєкту є: оплата лише за фактичний час виконання запитів, автоматичне масштабування до нуля при відсутності трафіку (для FastAPI-сервера), вбудована підтримка HTTPS та кастомних доменів, а також нативна інтеграція з іншими сервісами GCP (Secret Manager, Cloud Tasks, Artifact Registry). Всі ресурси розгорнуто в регіоні europe-west1, що забезпечує оптимальну затримку для цільової аудиторії в Центральній та Східній Європі.

Кожен із трьох серверних компонентів пакується у власний Docker-образ і зберігається в Artifact Registry.

FastAPI-сервер (server/Dockerfile) будується на базі python:3.11-slim. Залежності встановлюються до копіювання коду застосунку, що дозволяє Docker кешувати шар із залежностями при повторних збірках і значно скорочує час CI/CD. Команда запуску передає порт із змінної середовища PORT, яку Cloud Run встановлює автоматично

Toxicity Service (toxicity-service/Dockerfile) завантажує ваги моделі Detoxify під час збірки образу, а не при першому запиті. Це забезпечує передбачувану затримку холодного старту контейнера.

NLP Service (nlp-service/Dockerfile) виконує окремий скрипт під час збірки, що завантажує ваги моделей all-MiniLM-L6-v2 та cardiffnlp/twitter-roberta-base-sentiment-latest із Hugging Face Hub у кеш Docker-шару. Ваги fine-tuned DistilBERT (story-classifier/) копіюються безпосередньо разом із кодом сервісу.

У роботі всі чутливі параметри зберігаються виключно у Google Cloud Secret Manager. Секрети не передаються як звичайні змінні середовища при розгортанні, оскільки вони могли б потрапити до логів або бути витягнутими через metadata API. Перелік секретів із їх призначенням наведено в таблиці 6.1.

Таблиця 6.1 — Секрети в Google Cloud Secret Manager

Назва секрету	**Призначення**
`firebase-credentials`	Сервісний акаунт Firebase Admin SDK (JSON)
`nlp-service-api-key`	Спільний ключ для автентифікації між FastAPI та BentoML-сервісами
`qdrant-api-key`	API-ключ кластера Qdrant Cloud
`revenuecat-webhook-secret`	Секрет для верифікації вебхуків RevenueCat
`cloud-tasks-secret`	Секрет для захисту внутрішнього endpoint `/internal/process-story`

Credentials Firebase монтуються безпосередньо у файлову систему контейнера через параметр --set-secrets "/secrets/firebase-credentials.json=firebase-credentials:latest", а шлях до файлу передається у змінну середовища FIREBASE_CREDENTIALS_PATH. Решта секретів підставляються як змінні середовища.

Сервісному акаунту за замовчуванням Compute Engine надається роль roles/secretmanager.secretAccessor для кожного із зазначених секретів — це дотримується принципу мінімальних привілеїв.

NLP Service розгортається із виділеними ресурсами, оскільки ML-моделі вимагають значного обсягу оперативної пам'яті. Конфігурація: 2 CPU, 6 ГБ пам'яті, concurrency 10, мінімум 1 і максимум 5 екземплярів, таймаут 180 секунд, увімкнені cpu-boost та execution-environment gen2. Опція cpu-boost надає додаткові CPU-ресурси при запуску контейнера. Таймаут 180 секунд перевищує внутрішній traffic.timeout=120 BentoML, щоб Cloud Run не обривав з'єднання раніше, ніж BentoML встигне повернути відповідь.

FastAPI-сервер масштабується до нуля при відсутності трафіку (min-instances 0) - це скорочує витрати у нічний час. Значно менший обсяг пам'яті (512 МБ) відповідає вимогам чистого API-сервісу. Черга story-processing налаштована з обмеженням до 100 одночасних завдань та максимумом 3 повторних спроб. Кожне завдання вказує на внутрішній endpoint POST /internal/process-story на FastAPI-сервері, захищений заголовком X-Cloud-Tasks-Secret. Такий підхід гарантує, що жодна публікація не залишиться без NLP-обробки навіть при тимчасовій недоступності NLP-сервісу. На рисунку 6.1 зображено інфраструктурну діаграму розгорнутої системи.

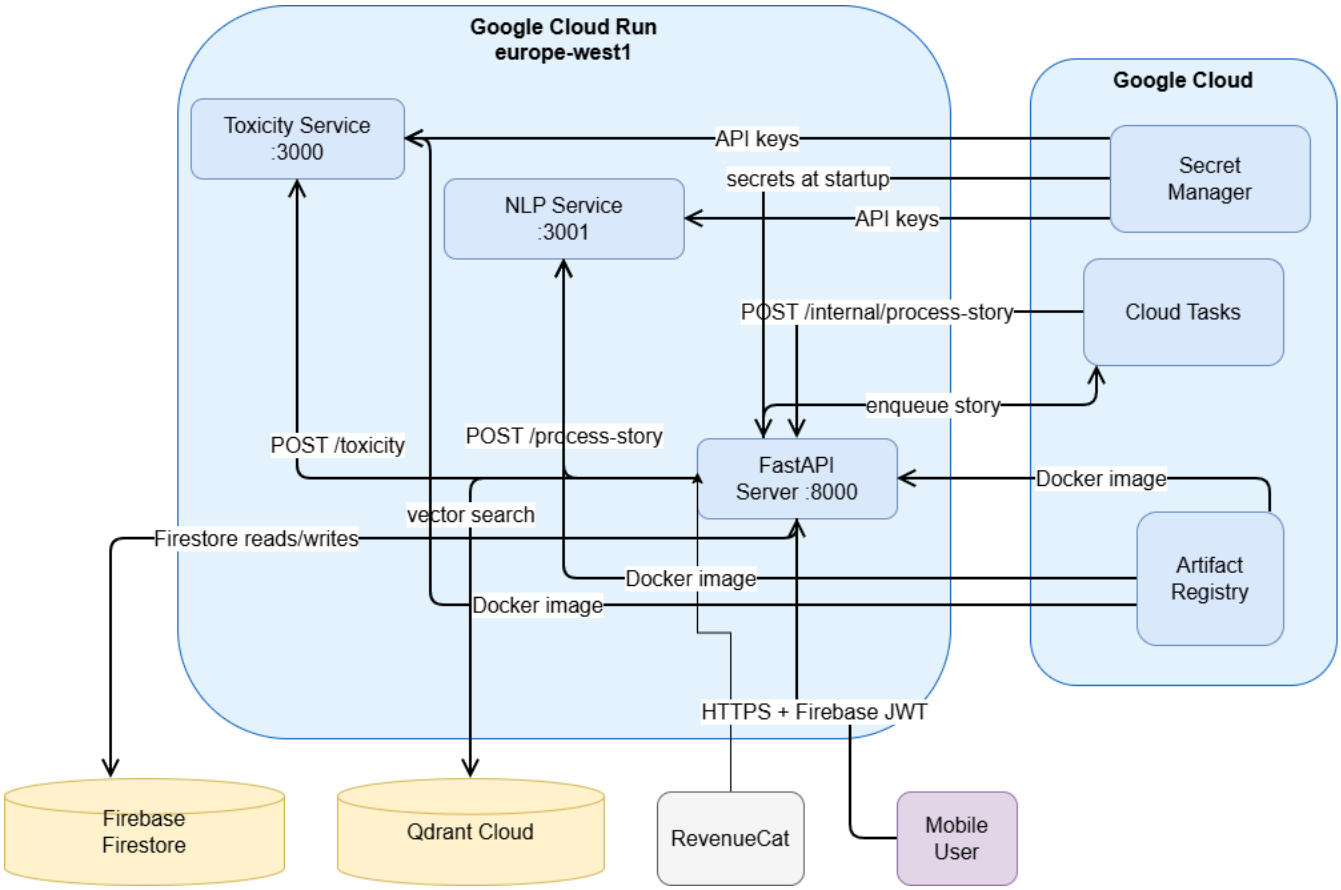

Рисунок 6.1 – Інфраструктурна діаграма системи

6.2 Налаштування Firebase та хмарної інфраструктури

У роботі Firebase Authentication використовується для реєстрації та входу користувачів через електронну пошту та пароль. Після реєстрації система надсилає лист підтвердження електронної адреси. Доступ до API-endpoints блокується до підтвердження email: залежність get_current_user на сервері перевіряє поле email_verified у декодованому Firebase JWT-токені та повертає HTTP 401 для неверифікованих акаунтів.

На стороні Flutter додаток відстежує потік AuthStateChanges із Firebase SDK: AuthBloc підписується на цей потік і перенаправляє користувача на екран підтвердження email при emailVerified == false.

Firebase-проєкт налаштований через FlutterFire CLI. Конфігурація генерується у файл lib/firebase_options.dart, де містяться ідентифікатори додатку

для Android та iOS. Файл google-services.json для Android не включається до репозиторію і зберігається в системі CI/CD.

Firestore організований у три основні колекції та один документ метаданих:

Таблиця 6.1 – Призначення firestore колекцій

Колекція / шлях	**Призначення**
`stories`	Публікації користувачів та Reddit-контент
`advices`	Поради до публікацій
`users`	Профілі користувачів, ліміти, рівень підписки
`metadata/categories`	Масив категорій із лічильниками публікацій

На рівні безпеки Firestore Rules налаштовано таким чином, що прямий клієнтський доступ до колекцій заборонений – сенсетивні операції відбуваються виключно через FastAPI-сервер, який використовує Firebase Admin SDK із сервісним акаунтом. Це запобігає несанкціонованому доступу до даних інших користувачів.

Для ефективного пошуку у роботі в Firestore налаштовано складені індекси. Зокрема, для рейтингового борду потрібен індекс за полями ratingCount (спадання) та __name__ (зростання), оскільки Firestore вимагає явного оголошення індексів для запитів із фільтрацією та сортуванням за різними полями.

Кластер Qdrant Cloud розміщений у регіоні europe-west1-gcp. Безкоштовний рівень хмарного кластера забезпечує 1 ГБ для зберігання векторних даних, що достатньо для початкового масштабу проєкту. Колекція stories налаштована з косинусною відстанню та розмірністю векторів 384 (відповідає моделі all-MiniLM-L6-v2). API-ключ зберігається у Secret Manager та передається у контейнер FastAPI через механізм монтування секретів.

RevenueCat налаштований з двома entitlement: pro та premium, кожен із яких містить відповідні продукти Google Play (одномісячні та річні підписки). Вебхук URL налаштовано у RevenueCat Dashboard з підтвердженим заголовком Authorization, значення якого збережене у Secret Manager як revenuecat-webhook-secret. Ідентифікатор RevenueCat API Key для Flutter-додатку завантажується із файлу .env через пакет flutter_dotenv. Файл .env не включається до репозиторію та передається окремо під час збірки.

6.3 Публікація та тестування додатку

Перед публікацією необхідно підписати APK/AAB цифровим підписом. У роботі налаштування підпису описано у файлі android/key.properties, який не включається до репозиторію. Файл містить шлях до keystore, alias, а також паролі. Конфігурація підпису підключається у android/app/build.gradle.kts через конфігурацію release збірки.

Завантаження нової версії у Google Play Console відбувається через розділ Internal Testing, що дозволяє тестувати збірку обмеженому колу користувачів до відкритого випуску. Після успішного тестування версія просувається до Open Testing або Production через систему поетапного розгортання (Staged Rollout) — спочатку до 10 % аудиторії, після моніторингу помилок — до 100 %.

На момент написання записки додаток проходить етап закритого альфа тестування.

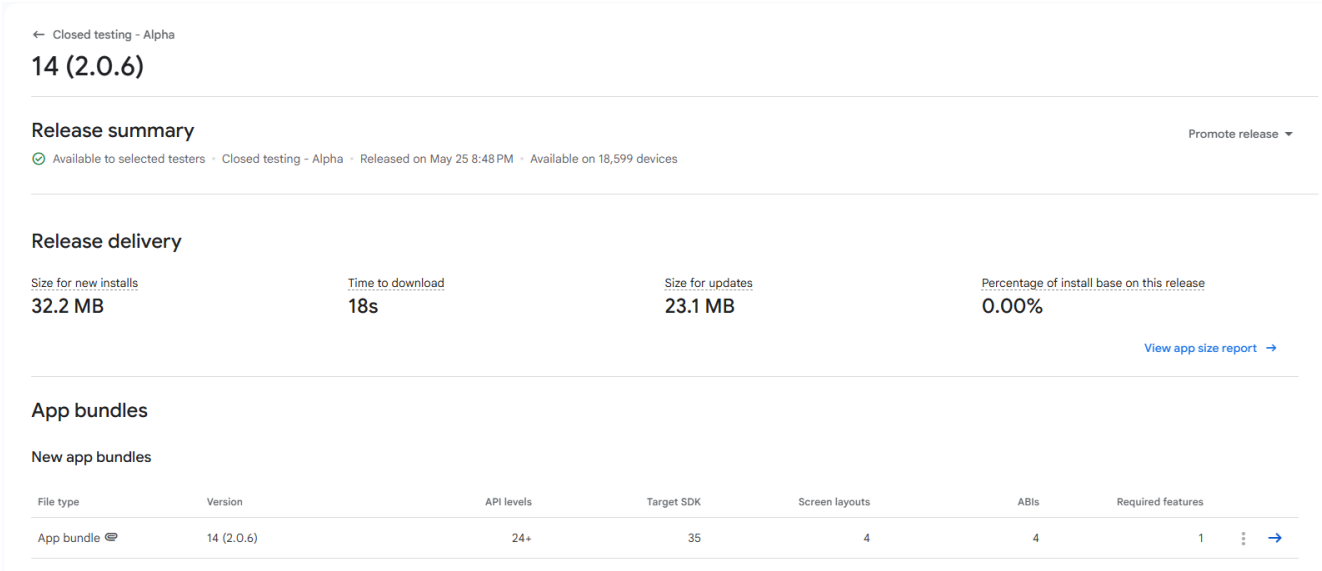

Рисунок 6.2 – Сторінка підсумків закритого тестування у Google Play Console

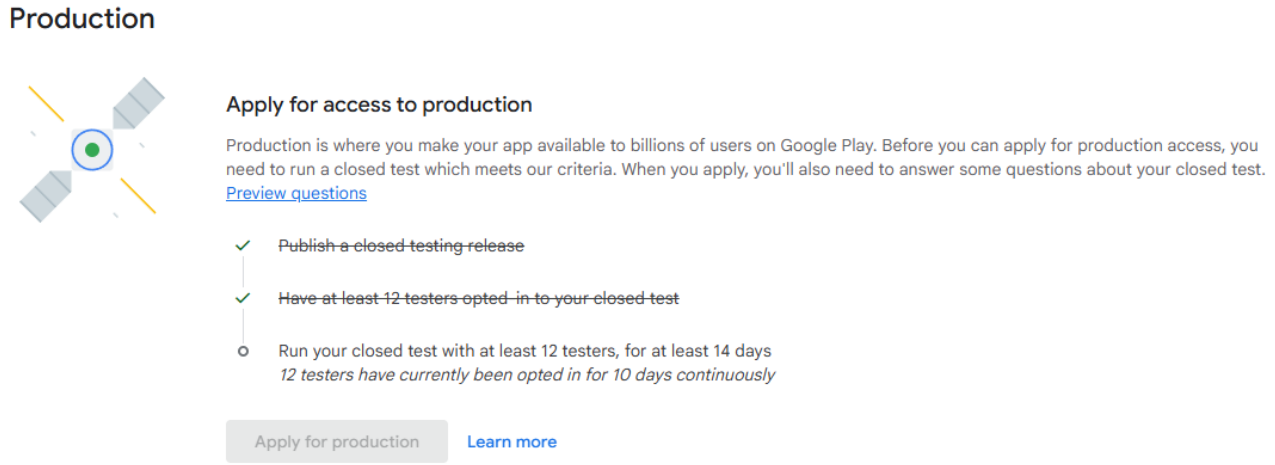

Рисунок 6.3 – Сторінка прогресу доступу до Production етапу у Google Play Console

7 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

У цьому розділі розглядаються питання безпеки життєдіяльності, пов'язані з розробкою програмного забезпечення та тривалою роботою за персональним комп'ютером. Аналізуються психологічні чинники, що становлять небезпеку для розробника в процесі професійної діяльності, а також визначаються вимоги до проходження профілактичних медичних оглядів для працівників, чия робота пов'язана з використанням ПК.

7.1 Психологічні чинники небезпеки

Психологічні чинники небезпеки — це особливості виробничого середовища та характеру діяльності, що можуть негативно впливати на психічний стан людини, спричиняти розвиток стресу, емоційного виснаження та зниження когнітивних функцій. Відповідно до НПАОП 0.00-7.15-18 та ДСанПіН 3.3.2.007-98, робота з відеодисплейними терміналами (ВДТ) відноситься до видів діяльності з підвищеним психоемоційним навантаженням, що зумовлено специфікою сприйняття інформації з екрана, необхідністю підтримувати концентрацію уваги протягом тривалого часу та алгоритмічним характером праці [24].

Психологічні чинники небезпеки при роботі з ПК поділяються на такі основні групи.

Психоемоційні перевантаження виникають унаслідок необхідності одночасної обробки великих обсягів інформації, прийняття рішень в умовах дефіциту часу та відповідальності за якість кінцевого продукту. У розробників програмного забезпечення зазначений чинник набуває особливої значущості через необхідність утримувати в оперативній пам’яті складні алгоритмічні структури та логічні залежності між компонентами системи.

Монотонність праці та знижена рухова активність є характерними для тривалої роботи за ПК. Відсутність фізичного навантаження у поєднанні з постійним зоровим та когнітивним напруженням порушує баланс між процесами

збудження та гальмування в центральній нервовій системі, що призводить до загальмованості, зниження продуктивності та збільшення кількості помилок.

Соціальна ізоляція та дефіцит міжособистісної комунікації є поширеними чинниками ризику в умовах дистанційної роботи, що набула масового поширення в ІТ-галузі. Тривала відсутність безпосереднього соціального контакту може спричиняти розвиток тривожних розладів, зниження мотивації та порушення соціальної адаптації.

Інформаційний стрес пов’язаний із постійним опрацюванням значних обсягів різнорідної інформації, відстеженням змін у технологіях, документації та вимогах до проекту. За класифікацією Міжнародної організації праці (МОП), інформаційний стрес відноситься до групи психосоціальних ризиків на робочому місці.

Розроблюваний мобільний додаток Advicer призначений для надання порад користувачам у різних сферах життєдіяльності. Враховуючи специфіку платформи, окремим і критично важливим психологічним чинником небезпеки є ризик отримання користувачем порад із деструктивним або токсичним змістом. Цей ризик набуває особливої значущості з огляду на те, що особи, які звертаються до подібних сервісів, нерідко перебувають у стані підвищеної психоемоційної вразливості, активно шукають підтримки або готуються до прийняття важливих рішень.

Психологічні наслідки зустрічі з негативними порадами можуть охоплювати широкий спектр реакцій: від короткочасного погіршення настрою та підвищення рівня тривожності до більш серйозних наслідків — загострення депресивних станів, підкріплення деструктивних поведінкових патернів, зниження самооцінки та формування негативних когнітивних установок. За даними досліджень у сфері цифрового психічного здоров’я, регулярна взаємодія з токсичним або знецінюючим контентом суттєво корелює із зростанням тривожності та депресивної симптоматики в користувачів мобільних платформ [30]. Особливо вразливими категоріями є підлітки та молодь у процесі формування ідентичності,

особи з діагностованими тривожними або депресивними розладами, а також ті, хто переживає кризові життєві ситуації.

З метою мінімізації зазначеного психологічного ризику в системній архітектурі Advicer реалізовано багаторівневий механізм автоматичної модерації контенту. NLP-сервіс на базі BentoML здійснює аналіз кожної поради на рівень токсичності з використанням моделі машинного навчання на основі трансформерної архітектури. Поради, що перевищують встановлений пороговий рівень токсичності, автоматично блокуються до публікації без участі модератора, що забезпечує безперервний захист психологічної безпеки аудиторії. На рівні серверної частини (FastAPI) запроваджено додаткову валідацію якості контенту перед збереженням у базі даних Firestore. Такий підхід відповідає принципам концепції «психологічно безпечного цифрового середовища», рекомендованої ВООЗ у документах щодо охорони психічного здоров'я в цифровому середовищі [30].

Для зниження впливу психологічних чинників небезпеки на розробників системи необхідно:

1) встановити регламентовані перерви тривалістю не менше 10–15 хвилин після кожної години безперервної роботи з ПК відповідно до ДСанПіН 3.3.2.007-98;
2) забезпечити раціональний розподіл навантаження з урахуванням індивідуальних психофізіологічних особливостей кожного члена команди;
3) впроваджувати елементи автоматизації рутинних операцій для зниження монотонності праці;
4) організувати систему психологічної підтримки та можливість анонімних консультацій з фахівцем для персоналу, залученого до перегляду та модерації контенту;
5) проводити регулярний моніторинг психоемоційного стану команди шляхом анонімного опитування та аналізу зворотного зв'язку.

Таким чином, психологічні чинники небезпеки при розробці та експлуатації інформаційних систем вимагають комплексного підходу як до охорони праці розробників, так і до забезпечення психологічної безпеки кінцевих користувачів. Реалізація механізмів автоматичної NLP-модерації в додатку Advicer є технічним заходом мінімізації ризику негативного психологічного впливу деструктивного контенту на аудиторію платформи.

7.2 Вимоги до профілактичних медичних оглядів для працівників ПК

Профілактичні медичні огляди є обов'язковою складовою системи охорони здоров'я працівників, діяльність яких пов'язана з постійним використанням персональних комп'ютерів та відеодисплейних терміналів. Правову основу їх проведення складають: Закон України «Про охорону праці» від 14.10.1992 № 2694-XII (стаття 17), Закон України «Основи законодавства України про охорону здоров'я» від 19.11.1992 № 2801-XII, а також Наказ Міністерства охорони здоров'я України від 21.05.2007 № 246 «Про затвердження Порядку проведення медичних оглядів працівників певних категорій» [23, 29].

Відповідно до Наказу МОЗ № 246 та НПАОП 0.00-7.15-18, обов'язковим профілактичним медичним оглядам підлягають усі категорії працівників, що у своїй основній діяльності використовують ВДТ та ПК [24]:

1) оператори комп'ютерного набору та введення даних;
2) програмісти, системні адміністратори та розробники програмного забезпечення;
3) операційні та технічні працівники з обслуговування обчислювальної техніки;
4) диспетчери та оператори систем управління, що працюють з екранними пристроями.

Умовою обов'язковості огляду є використання ВДТ протягом більше 50 % робочого часу або не менше 4 годин безперервно на добу.

Чинне законодавство передбачає три основні види медичних оглядів для зазначеної категорії працівників.

Попередній медичний огляд проводиться при прийнятті на роботу, пов'язану з використанням ПК, до початку виконання трудових обов'язків. Метою є виявлення абсолютних та відносних протипоказань до даного виду діяльності й встановлення вихідного стану здоров'я. Наявність певних захворювань - виражених порушень зору, хронічних захворювань кістково-м'язової системи, нервово-психічних розладів - може слугувати підставою для обмеження або заборони роботи з ВДТ.

Первинний (плановий) медичний огляд проводиться не рідше одного разу на два роки для всіх категорій операторів ПК, а для осіб, молодших 21 року, — щорічно. Відповідно до пункту 2.4 Наказу МОЗ № 246, роботодавець зобов'язаний забезпечити проведення медичних оглядів за власний рахунок та надати відповідні направлення до закладів охорони здоров'я.

Позаплановий медичний огляд призначається за поданням фахівця з охорони праці або профспілкового органу у разі погіршення стану здоров'я працівника, появи скарг, що можуть бути пов'язані з умовами праці, а також після тривалої (понад 2 місяці) тимчасової непрацездатності.

Під час профілактичного медичного огляду оператора ПК обов'язковими є консультації таких спеціалістів:

1) терапевт - загальний стан здоров'я, серцево-судинна та дихальна системи;
2) невролог - стан нервової системи, виявлення тунельного синдрому зап'ястного каналу, синдрому хронічного стомлення;
3) офтальмолог - гострота зору, рефракція, стан акомодаційного апарату, внутрішньоочний тиск;
4) хірург-ортопед - стан опорно-рухового апарату, виявлення патологій хребта та суглобів.
5) загальний аналіз крові та сечі;
6) визначення рівня глюкози крові;

7) ЕКГ (один раз на два роки для осіб до 40 років, щорічно - для осіб старше 40 років);
8) вимірювання артеріального тиску при кожному огляді;
9) перевірка гостроти зору та кольоросприйняття.

За результатами медичного огляду лікарська комісія видає висновок про придатність або непридатність до роботи з ВДТ, який долучається до особової справи працівника. Роботодавець зобов'язаний вести журнал обліку медичних оглядів за встановленою формою та своєчасно передавати до Фонду соціального страхування відомості про виявлені профзахворювання. Відповідно до статті 17 Закону України «Про охорону праці», відмова від проходження медичного огляду є підставою для відсторонення працівника від виконання трудових обов'язків без збереження заробітної плати [23].

Постійними медичними протипоказаннями до роботи з ВДТ відповідно до Наказу МОЗ № 246 є: виражені психічні розлади та розлади поведінки, епілепсія та судомні стани, виражені захворювання зорового нерва та сітківки, виражений тремор кінцівок. Тимчасові протипоказання встановлюються при гострих інфекційних захворюваннях, загостреннях хронічних хвороб та вагітності (після 5-го місяця).

Таким чином, система профілактичних медичних оглядів для операторів ПК є комплексним механізмом раннього виявлення та попередження захворювань, зумовлених специфікою роботи з відеодисплейними терміналами. Суворе дотримання вимог Наказу МОЗ № 246 роботодавцями ІТ-сфери є необхідною умовою збереження здоров'я та тривалої працездатності команди розробників програмного забезпечення.

ВИСНОВКИ

У дипломній роботі розроблено крос-платформний мобільний застосунок обміну порадами Advicer, що поєднує наративний текстовий формат публікацій із алгоритмічною стрічкою випадкового контенту та автоматичною обробкою тексту засобами штучного інтелекту.

Проведений аналіз предметної області та огляд існуючих рішень показав, що жодна з розглянутих платформ - Reddit, Quora, TikTok/Reels, Wisdo - не поєднує повного набору необхідних властивостей: мобільно-орієнтованого інтерфейсу, випадкової стрічки, автоматичної AI-модерації та персоналізованих рекомендацій. Визначено технологічний стек, що відповідає вимогам продуктивності, масштабованості та швидкості розробки: Flutter, FastAPI, Firebase, DistilBERT, BentoML та Qdrant. Спроектовано трирівневу розподілену архітектуру системи. Мобільний клієнт взаємодіє з FastAPI-сервером через HTTPS із Firebase JWT-автентифікацією; серверна частина координує взаємодію між Firestore, двома NLP-мікросервісами, Qdrant та Google Cloud Tasks. Асинхронна обробка нових публікацій через чергу Cloud Tasks дозволяє не блокувати UX під час ресурсоємної NLP-обробки та гарантує надійну доставку задач при тимчасовій недоступності сервісів.

Розроблено мобільний клієнт на Flutter із застосуванням патерну BLoC для управління станом, go_router для auth-захищеної навігації, репозиторного патерну для абстракції рівня даних та Hive для локального кешування.

Реалізовано три рівні підписок (free, pro, premium) із синхронізацією через RevenueCat та вебхуки на стороні сервера. Реалізовано серверну частину на FastAPI із чітким поділом контролерів та сервісного шару, rate limiting за Firebase UID, батчевим читанням із Firestore та TTL-кешуванням рейтингів. Алгоритм випадкової вибірки публікацій на основі багатовимірного простору забезпечує рівномірний розподіл без повного сканування колекції, а механізм discovery query гарантує нещодавно доданим публікаціям шанс потрапити до стрічки.

Розроблено підсистему штучного інтелекту, що включає два незалежні BentoML-мікросервіси. Сервіс токсичності на базі моделі Detoxify із батчингом запитів забезпечує модерацію порад із порогом 0.7. NLP-сервіс поєднує fine-tuned DistilBERT для класифікації тематики (20 категорій, час виведення ~91 мс на CPU), модель paraphrase-multilingual-mpnet-base-v2 для генерації 768-вимірних ембедингів та RoBERTa для аналізу тональності.

Система персоналізованих рекомендацій будує профіль користувача через центроїди ембедингів вподобаних публікацій, виконує батчевий пошук у Qdrant та змішує результати з випадковим контентом відповідно до параметра random_ratio для забезпечення різноманіття стрічки. Систему розгорнуто на Google Cloud Run у регіоні europe-west1. FastAPI-сервер та NLP-сервіс масштабується до нуля при відсутності трафіку. Усі чутливі параметри зберігаються у Google Cloud Secret Manager. Мобільний застосунок опубліковано у Google Play Store із поетапним розгортанням. Таким чином, усі поставлені функціональні та нефункціональні вимоги виконано: реалізовано анонімну публікацію і перегляд порад у випадковій стрічці, автоматичну AI-класифікацію та модерацію контенту, персоналізовані рекомендації на основі векторного пошуку, систему підписок та рейтинг авторів порад. Застосований підхід забезпечує горизонтальну масштабованість та можливість незалежного оновлення кожного компонента системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Михалик Д.М. Методичні вказівки до виконання кваліфікаційної роботи бакалавра для здобувачів першого (бакалаврського) рівня вищої освіти за освітньо-професійною програмою «Інженерія програмного забезпечення» спеціальності 121 – «Інженерія програмного забезпечення» всіх форм навчання. – Тернопіль : ТНТУ ім. І. Пулюя, 2024. – 45 с.

2. Flutter documentation. – URL: https://docs.flutter.dev (дата звернення: 10.05.2025).

3. FastAPI documentation. – URL: https://fastapi.tiangolo.com (дата звернення: 10.05.2025).

4. Firebase documentation. – URL: https://firebase.google.com/docs (дата звернення: 10.05.2025).

5. Sanh V., Debut L., Chaumond J., Wolf T. DistilBERT, a distilled version of BERT. – 2019. – URL: https://arxiv.org/abs/1910.01108.

6. BentoML documentation. – URL: https://docs.bentoml.com (дата звернення: 10.05.2025).

7. Qdrant vector database documentation. – URL: https://qdrant.tech/documentation (дата звернення: 10.05.2025).

8. RevenueCat documentation. – URL: https://docs.revenuecat.com (дата звернення: 10.05.2025).

9. Google Cloud Run documentation. – URL: https://cloud.google.com/run/docs (дата звернення: 10.05.2025).

10. Detoxify library. – URL: https://github.com/unitaryai/detoxify (дата звернення: 10.05.2025).

11. Devlin J. et al. BERT: Pre-training of Deep Bidirectional Transformers. – 2018. – URL: https://arxiv.org/abs/1810.04805.

12. Reimers N., Gurevych I. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. – 2019. – URL: https://arxiv.org/abs/1908.10084.

13. Hive database for Flutter. – URL: https://docs.hivedb.dev (дата звернення: 10.05.2025).

14. go_router package. – URL: https://pub.dev/packages/go_router (дата звернення: 10.05.2025).

15. Kocienda K. et al. Bloc Library for Flutter. – URL: https://bloclibrary.dev (дата звернення: 10.05.2025).

16. Google Cloud Secret Manager. – URL: https://cloud.google.com/secret-manager/docs (дата звернення: 10.05.2025).

17. Жидецький В.Ц. Охорона праці користувачів комп'ютерів : підручник. 2-ге вид., перероб. і доп. Львів : Афіша, 2022. 192 с.

18. НПАОП 0.00-7.15-18. Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями : затв. наказом Міністерства соціальної політики України від 14.02.2018 № 207.

19. ДСН 3.3.6.042-99. Санітарні норми мікроклімату виробничих приміщень. Київ : МОЗ України, 1999.

20. ДБН В.2.5-28 : 2018. Природне і штучне освітлення. Київ : Мінрегіон України, 2018. 133 с.

21. Гогіташвілі Г.Г., Лапін В.М. Основи охорони праці : навч. посіб. 5-те вид., випр. і доп. Київ : Знання, 2023. 318 с.

22. НАПБ А.01.001-2014. Правила пожежної безпеки в Україні. Київ : ДСНС України, 2014. 45 с.

23. Наказ Міністерства охорони здоров'я України від 21.05.2007 № 246 «Про затвердження Порядку проведення медичних оглядів працівників певних категорій». Київ : МОЗ України, 2007.

24. Бережна С.В., Хмарук О.О. Психосоціальні ризики в цифровому робочому середовищі : монографія. Київ : Університет «Україна», 2023. 228 с.

ДОДАТКИ

Додаток А – Дослідження прогнозування властивостей матеріалів за допомогою ансамблевих та базових ML алгоритмів

Advances in Materials Science and Engineering

WILEY

RESEARCH ARTICLE OPEN ACCESS

Data-Driven Design of Epoxy Composites: Heat Resistance Prediction Using Machine Learning Algorithms

Oleh Pastukh[1] | Yuriy Petrov[1] | Andrii Buketov[2] | Oleh Totosko[3] | Vladyslav Strelchenko[2] | Vitaliy Sotsenko[2]

[1]Department of Software Engineering, Ternopil Ivan Pulyu National Technical University, Ternopil, Ukraine | [2]Department of Transport Technologies and Mechanical Engineering, Kherson State Maritime Academy, Kherson, Ukraine | [3]Department of Computer-Integrated Technologies, Ternopil Ivan Pulyu National Technical University, Ternopil, Ukraine

Correspondence: Oleh Totosko (totosko@gmail.com)

Received: 17 December 2025 | **Revised:** 11 March 2026 | **Accepted:** 9 April 2026

Academic Editor: Mohammad Rezwan Habib

Keywords: composite | fillers | heat resistance | machine learning algorithms | mechanical properties | polymer | prediction | transport

ABSTRACT

The paper presents the results of studies of the mechanical properties of modified and dispersion-filled epoxy composite materials: destructive stresses and modulus of elasticity in bending, impact toughness, adhesive strength, residual stresses and heat resistance. At the initial stage, we applied the analysis of a set of properties only for training ensemble machine learning (ML) algorithms. Based on experimental data, we then considered the problem of predicting the values of the variable heat resistance depending on the values of the mechanical properties of composites. To solve the prediction problem, we used various ML algorithms: AdaBoost, bagging, decision trees, gradient boost, random forest, stacking, voting, ridge regression, support vector machine (SVM) and K-nearest neighbour (*k*-NN), which establish the dependence of the values of the variable heat resistance on the values of the above variables. To ensure the reliability of the results and minimise dependence on the specifics of individual algorithms, a comprehensive approach was used, combining ML and mathematical statistics methods. The results of solving two important tasks have been obtained: the first task is to predict the values of the heat resistance variable based on the values of the mechanical properties of materials, and the second task is to determine the importance of the influence of the values of all variables on the value of the predicted heat resistance variable. Based on the results of the first task, it can be stated that all models (ML algorithms) obtained in the work for prediction are highly effective since their MAPE has low values on the test sample and, therefore, can be used by experimenters to reduce the costs of various types of resources for conducting experiments. Based on the results of solving the second task using three different approaches, conclusions can be drawn about the significance of the results obtained, given that the models assess the importance of the influence of variable values in the context of their complex interaction with each other. The information obtained has potential value for experimenters in their optimal management of variable importance values and in organising experiments to create new materials with predicted values of the selected variable.