МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра інформаційно-обчислювальних систем і управління

ІВАНЮШ Адам Романович

Мобільний застосунок для покращення людських взаємовідносин з використанням штучного інтелекту/ Mobile application for improving human relationships using artificial intelligence

Спеціальність 122 – Комп'ютерні науки
Освітньо-професійна програма – Комп'ютерні науки

Кваліфікаційна робота

Виконав студент групи КН-43
А. Р. Іванюш

Науковий керівник:
к.т.н., доцент П.Є. Биковий

Кваліфікаційну роботу допущено до захисту
«___»_____ 2025 р.

В.о. завідувача кафедри
_____ Н.М. Васильків

Тернопіль – 2025

Факультет комп'ютерних інформаційних технологій
Кафедра інформаційно-обчислювальних систем і управління
Освітній ступінь «бакалавр»
спеціальність: 122 «Комп'ютерні науки»
освітньо-професійна програма – «Комп'ютерні науки»

«ЗАТВЕРДЖУЮ»
В. о. завідувача кафедри ІОСУ
_____ Надія ВАСИЛЬКІВ
«____» _____20___р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
ІВАНЮШУ Адаму Романовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи: Мобільний застосунок для покращення людських взаємовідносин з використанням штучного інтелекту/ Mobile application for improving human relationships using artificial intelligence
керівник роботи Биковий П.Є., к.т.н., доц.,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від 20 грудня 2024 року № 938.
2. Строк подання студентом закінченої кваліфікаційної роботи 25 травня 2025р.
3. Вихідні дані до кваліфікаційної роботи: завдання на кваліфікаційну роботу студента, наукові статті, технічна література.
4. Основні питання, які потрібно розробити
— проаналізувати сучасні рішення взаємодії з моделями штучного інтелекту через API;
— розробити архітектуру системи з врахуванням взаємодії клієнтської частити з серверною частиною та її з моделями штучного інтелекту;
— покращити надання відповіді від моделей штучного інтелекту шляхом прописування глибших промптів;
— провести тестування інтерфейсу.
5. Перелік графічного матеріалу у роботі:
— діаграма потоків даних;
— архітектура мобільного застосунку;
— основний алгоритм роботи системи генерування питання;
— глобальна інфологічна модель;
— локальна інфологічна модель;
— модель даних у вигляді ERD;
— структурна схема програмної систем;
— UML-діаграма станів.

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 20 грудня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів кваліфікаційної роботи	Примітка
1	Затвердження теми кваліфікаційної роботи, ознайомлення з літературними джерелами та складання плану роботи	до 01.01.2025 р.	
2	Написання 1 розділу кваліфікаційної роботи	до 01.02.2025 р.	
3	Написання 2 розділу кваліфікаційної роботи	до 01.04.2025 р.	
4	Написання 3 розділу кваліфікаційної роботи	до 01.05.2025 р.	
5	Представлення попереднього варіанту кваліфікаційної роботи, перевірка та внесення змін керівником	до 15.05.2025 р.	
6	Опрацювання зауважень та представлення завершеного варіанту кваліфікаційної роботи. Підготовка супроводжуючих документів	до 25.05.2025 р.	
7	Перевірка кваліфікаційної роботи на оригінальність тексту	до 30.05.2025 р.	
8	Оформлення кваліфікаційної роботи та отримання допуску до захисту	до 10.06.2025 р.	
9	Подання кваліфікаційної роботи до захисту на засіданні атестаційної комісії	до 10.06.2025 р.	

Студент _____ А. Р. Іванюш
(підпис) (прізвище та ініціали)

Керівник роботи _____ П. Є. Биковий
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Кваліфікаційна робота на тему «Мобільний застосунок для покращення людських взаємовідносин з використанням штучного інтелекту» на здобуття освітнього ступеня «бакалавр» зі спеціальності 122 «Комп'ютерні науки» освітньої програми «Комп'ютерні науки» написана обсягом в 82 сторінки і містить 26 ілюстрацій, 1 таблицю, 4 додатки та 29 використаних джерел.

Метою роботи є розробка мобільного застосунку «Gather Together», який використовує сучасні технології штучного інтелекту для генерації питань за допомогою, яких люди зможуть покращувати взаємовідносини.

Методи дослідження: математична статистика, теорія штучних нейронних мереж.

Внаслідок виконання роботи створено інструмент, котрий генерує питання, які можна використати для покращення взаємовідносин людей, шляхом їхньої комунікації про це питання. Також досліджено ефективність різних моделей штучного інтелекту для створення подібних питань.

Результати роботи можуть бути використані в буденності у колі близьких людей для спілкування між собою.

Ключові слова: ШТУЧНИЙ ІНТЕЛЕКТ, МОБІЛЬНИЙ ЗАСТОСУНОК, APPWRITE, FLUTTER, CHATGPT, GEMINI, DEEPSEEK.

ANNOTATION

Qualification work on the topic «Mobile application for improving human relationships using artificial intelligence» for Bachelor’s degree on speciality 122 «Computer Science» educational and professional program «Computer Science» is written on 82 pages and it contains 26 figures, 1 table, 4 annexes and 29 sources.

The aim of the work is to develop a mobile application called "Gather Together", which uses modern artificial intelligence technologies to generate questions that help people improve their relationships through communication.

Research methods: mathematical statistics, theory of artificial neural networks.

As a result of the work, a tool was developed that generates questions aimed at enhancing human relationships by encouraging meaningful communication. Additionally, the effectiveness of various AI models for generating such questions was explored.

The results of the thesis can be applied in everyday life among close people to facilitate better interpersonal communication.

Keywords: ARTIFICIAL INTELLIGENCE, MOBILE APPLICATION, APPWRITE, FLUTTER, CHATGPT, GEMINI, DEEPSEEK.

ЗМІСТ

ВСТУП

На сьогоднішній день, життя дуже тісно переплітається з цифровими технологіями і штучний інтелект активно впроваджується в різноманітні сфери діяльності. Стрес, ізоляція, цифрове перевантаження та недостатня емоційна взаємодія часто стають причинами погіршення міжособистісного спілкування. Штучний інтелект, який активно впроваджується у повсякденне життя, має потенціал не лише автоматизувати процеси, а й виступати як інструмент підтримки та розвитку емоційного інтелекту, комунікації та гармонізації стосунків між людьми. Проблемою є недостатнє вивчення та усвідомлення потенціалу ШІ як засобу покращення якості міжособистісних взаємин, а також відсутність простих інструментів з якими люди зможуть взаємодіяти, аби покращити міжособистісні відносини.

Дана робота є актуальною на сьогоднішній день у зв’язку з активним розвитком штучного інтелекту, а також потребою покращення взаємовідносин людей шляхом пізнання одне одного.

Метою даної роботи є розробка мобільного застосунку, котрий використовуватиме широко розповсюджені моделі штучного інтелекту від ChatGPT, Gemini та DeepSeek для генерування питань, котрі користувачі зможуть використовувати для обговорень тим самим покращувати міжособистісні взаємовідносини. Для того аби досягти дану мету необхідно вирішити наступні завдання:

— Розробка функціоналу для генерування питань, що сприяють покращення комунікації співрозмовників. А саме інтеграція ChatGPT, Gemini та DeepSeek для генерування питань.

— Оптимізація налаштувань (як-от параметр «tempreture») для досягнення балансу між якістю та творчістю відповідей.

— Створення ефективних промптів, які враховують рекомендації психологів та забезпечують використання отриманих результатів безпосередньо у коді застосунку.

Об’єктом дослідження є процес розробки мобільного застосунку «Gather Together» з інтеграцією взаємодії з моделями штучного інтелекту від ChatGPT, Gemini та DeepSeek.

Методи дослідження:

— Аналіз наукової літератури та публікацій з області штучного інтелекту та його використання або взаємодії у області психології.

— Розробка та оптимізація промпту для генерування питань штучним інтелектом.

— Тестування розробленого застосунку та оцінка якості генерованих питань різних моделей штучних інтелектів.

Дана робота має практичне значення, так як безпосереднього розроблений мобільний застосунок «Gather Together» зможе використати будь-хто, хто має смартфон, а також співбесідника.

Результати роботи були апробовані на XXV Всеукраїнській науково-технічній конференції молодих вчених, аспірантів та студентів «Стан, досягнення і перспективи інформаційних систем і технологій» 17.04.2025-18.04.2025 Одеса, Україна та подані до друку в «Стан, досягнення і перспективи інформаційних систем і технологій / Матеріали XXV Всеукраїнської науково-технічної конференції молодих вчених, аспірантів та студентів. Одеса, 17-18 квітня 2025 р. - Одеса, Видавництво ОНТУ, 2025 р. – 287 с.» (Додаток Г).

1 АНАЛІЗ СИСТЕМ З ІНТЕГРАЦІЄЮ ШТУЧНОГО ІНТЕЛЕКТУ

1.1 Опис предметної області «Системи з інтеграцією штучного інтелекту»

Штучний інтелект (ШІ) стрімко розвивається та проникає в усі сфери людського життя. Сьогодні ШІ вже не є лише теоретичною концепцією, а став невід'ємною частиною нашого повсякдення, від цифрових помічників та пошукових систем до алгоритмів соціальних мереж.

На цьому тлі зростає інтерес до вивчення потенціалу ШІ у вирішенні складних людських потреб, включаючи фундаментальне прагнення до зв'язку та значущих відносин. Дедалі більше усвідомлення «епідемії самотності», що посилюється такими факторами, як урбанізація, соціальні мережі та глобальні події, підкреслює нагальність пошуку інноваційних рішень . Саме цей контекст створює сприятливе підґрунтя для дослідження того, як ШІ може бути використаний для підтримки та покращення людських відносин.

Розробка ШІ-застосунків для покращення людських відносин є складним завданням, яке вимагає врахування багатьох аспектів. Насамперед, критично важливо забезпечити довіру користувачів до таких систем. Люди повинні бути впевнені, що ШІ працює в їхніх інтересах, зберігає неупередженість і не загрожує їхньому емоційному добробуту. Окрім цього, необхідно ретельно підходити до конфіденційності та безпеки даних, оскільки дані якими користувачі діляться зі штучним інтелектом, є особливо чутливими. Чіткі політики обробки інформації та прозорі механізми захисту мають бути пріоритетом.

Ще одним викликом є уникнення алгоритмічних упереджень, які можуть спричинити соціальну нерівність. ШІ має навчатися на збалансованих даних, щоб гарантувати справедливість у взаємодії. Важливо також зберігати автентичність, аби користувачі усвідомлювали, що спілкуються саме з машиною.

У сучасному цифровому світі зростає потреба у створенні рішень, що сприяють зміцненню людських зв’язків. Штучний інтелект має значний потенціал у цій сфері, пропонуючи доступну, персоналізовану та своєчасну підтримку людям, які прагнуть покращити свої соціальні взаємодії. Однак його впровадження

супроводжується низкою викликів, зокрема ризиком емоційної залежності, погіршення соціальних навичок і виникненням етичних проблем. Саме тому важливо розробляти такі технології відповідально, орієнтуючись на потреби людини та зберігаючи безпечні межі у використанні технологій.

На рисунку 1.1 зображено мобільний застосунок як головний об’єкт котрий об’єднює в собі функціонал пов’язаний з інтерфейсом, а також серверною (backend) частиною. Ліва гілка присвячена інтерфейсу і складається з наступних функціональних елементів:

— Інтерфейс - відповідає за візуальну частину та взаємодію з користувачем.

— Авторизація / Реєстрація - користувач має можливість увійти у систему або створити новий акаунт.

— Вибір категорії питання - користувач обирає тему або категорію, до якої має належати згенероване питання.

— Відображення питання – елемент інтерфейсу, де згенероване питання показується користувачу.

— Оцінка згенерованих питань - користувач має можливість оцінити, наскільки питання є корисним або якісним.

Права гілка в свій час присвячена серверній частині, де відбуваються запити на API одного з провайдерів моделей штучного інтелекту.

— Backend - серверна частина, яка обробляє логіку, зберігає дані та комунікує з AI-моделями.

— Запити до ChatGPT - бекенд може надсилати запити до API ChatGPT для генерації питань.

— Запити до Gemini - альтернативна модель від Google, яку можна використовувати для генерації.

Запити до DeepSeek - ще одна модель, до якої можна звернутись для отримання питання.

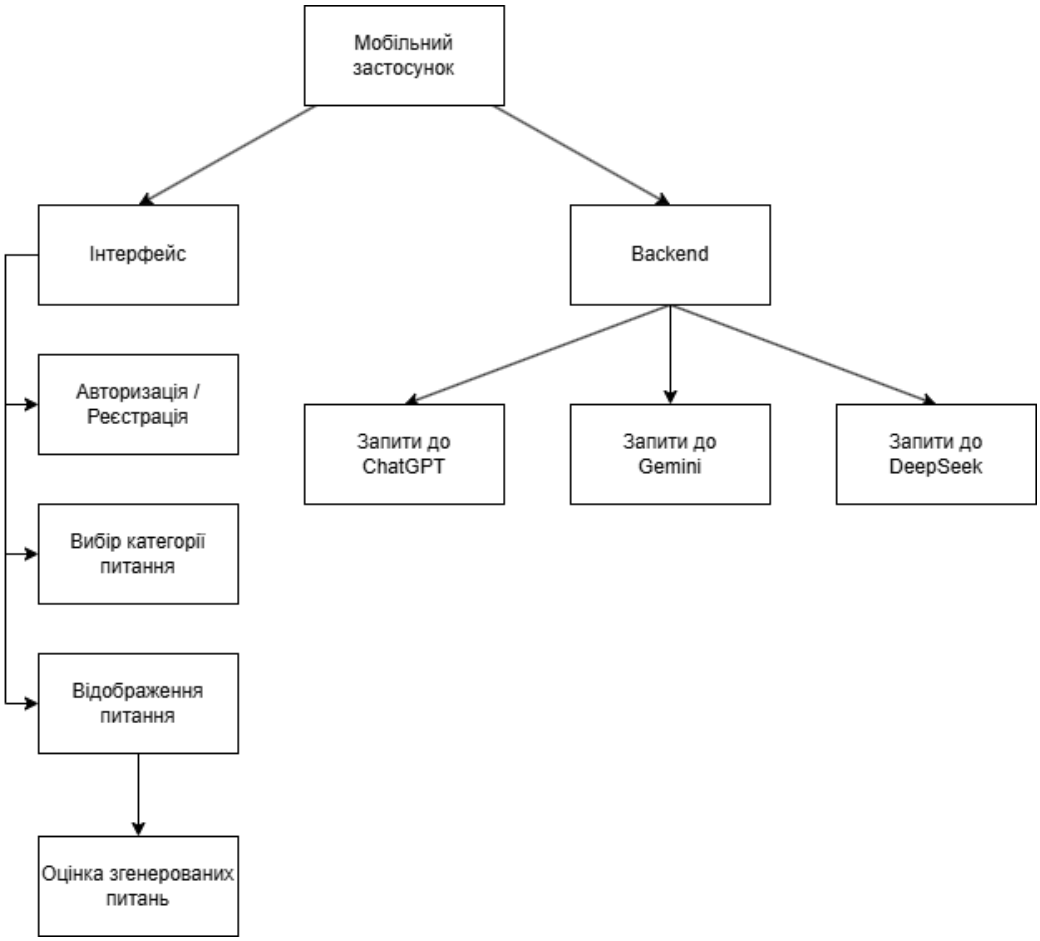


Рисунок 1.1 – Діаграма дерева функцій мобільного застосунку

1.2 Огляд і аналіз існуючих аналогів

Для вивчення інтеграції штучного інтелекту з метою генерування питань направлених на покращення людських взаємовідносин та надати можливість людям глибше пізнавати одне одного були досліджені існуючі статті, наукові дослідження на подібну тему та існуючі ШІ рішення.

Party Qs — це мобільний застосунок, створений для того, щоб допомагати людям легко розпочинати цікаві, веселі та глибокі розмови під час соціальних зустрічей. Головний екран, якого зображений на рисунку 1.2. «Party Qs» містить понад 2000 ретельно підібраних запитань, розділених на категорії, такі як: побачення, вечірки, філософські теми, гумористичні запитання, особистісний розвиток тощо.

Застосунок має простий та зручний інтерфейс для гортання запитань, збереження улюблених і навіть можливість ділитися ними в соціальних мережах (рисунок 1.3). Party Qs також дозволяє користувачам надсилати власні запитання.

Головна мета застосунку — покращити спілкування, особливо для людей із соціальною тривожністю, створити атмосферу цікавого діалогу та зменшити час, який зазвичай витрачається на перегляд телефону під час зустрічей.

Рисунок 1.2 – Головний екран застосунку «Party Qs»

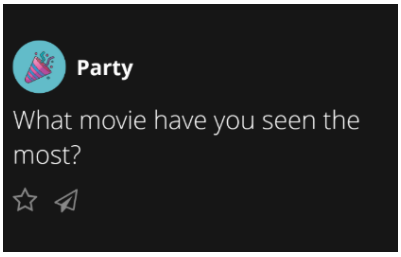


Рисунок 1.3 – Відображення питання в застосунку «Party Qs»

TableTopics — це мобільний застосунок, створений на основі популярної гри-запитальника, що допомагає розпочати цікаві розмови в будь-якому колі: з родиною, друзями чи навіть незнайомцями (рисунок 1.4). Його головна мета — сприяти щирому, живому спілкуванню та створювати незабутні моменти завдяки оригінальним питанням.

Рисунок 1.4 – Головний екран застосунку «TableTopics»

Застосунок «TableTopics» відкриває безліч можливостей для живого й захопливого спілкування. Уже з першого запуску надається понад п’ятдесят безкоштовних запитань, які охоплюють найрізноманітніші теми й підходять для будь-якої компанії. За бажанням, у додатку можна отримати доступ до понад чотирьохсот додаткових запитань через внутрішні покупки — цього достатньо, щоб жодна розмова не повторювалася.

Усі запитання в «TableTopics» розподілено за тематичними категоріями. Швидко знаходяться розділи про мистецтво й музику, вечірки й подорожі,

кулінарію, екологію та попкультуру. Передбачено окремі блоки для батьків і дітей, підлітків і студентів — тож кожну бесіду можна оживити у форматі, що найкраще відповідає конкретній компанії.

Коли знаходиться запитання, що особливо захоплює, його можна зберегти до вибраного — для повторного використання або створення власних добірок. Поділитися улюбленим запитанням можна будь-де: у Facebook, Instagram чи месенджерах — одним натисканням кнопки.

Справжня цінність «TableTopics» полягає в тому, що застосунок надихає до глибоких, веселих та іноді зовсім несподіваних розмов (рисунок 1.5). Він допомагає краще пізнати близьких, знайти спільну мову з тими, з ким важко заговорити (наприклад, підлітками), і легко перетворює навіть офіційні збори на невимушену бесіду. Завдяки «TableTopics» кожна віртуальна зустріч, сімейна вечеря, поїздка чи масштабна вечірка наповнюється новими темами й непередбачуваними відкриттями.

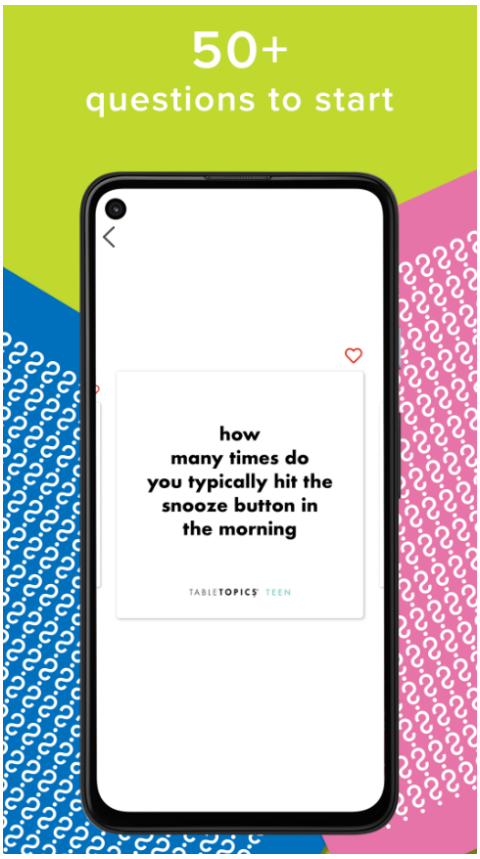


Рисунок 1.5 – Екран із запитанням в застосунку «TableTopics»

Daily Hellos — це мобільний застосунок, створений для того, щоб допомогти людям спілкуватися глибше, щиріше й усвідомленіше. Його головна мета — зробити розмови з близькими не поверхневими, а наповненими змістом, емоціями та справжнім інтересом один до одного (рисунок 1.6).

Рисунок 1.6 – Головний екран застосунку «Daily Hellos»

У додатку зібрано понад 500 ретельно підібраних запитань на понад 30 різних тем. Це запитання не лише про розваги, а й про стосунки, внутрішній світ, цінності, почуття, виховання, гроші та духовність. Вони однаково добре підходять як для першого побачення, коли хочеться краще пізнати людину, так і для глибоких розмов із партнером, вечірніх бесід за сімейною вечерею чи просто хвилин самоусвідомлення наодинці з собою.

«Daily Hellos» допомагає уникнути порожніх балачок і ніякових пауз. Це інструмент, який дозволяє зручно, легко й водночас змістовно говорити з тими, хто поруч. Тут знайдеться все — і легкі жартівливі теми для вечора з друзями, і серйозні роздуми про життя для душевної розмови на двох.

Завдяки простому інтерфейсу користувачі можуть обирати теми, переглядати запитання, зберігати улюблені та повертатися до них пізніше. Ці розмови не лише створюють приємні моменти, а й сприяють глибшому розумінню себе й інших. Іноді саме одне добре поставлене запитання може змінити якість ваших стосунків або надихнути на новий рівень близькості.

Застосунок підійде тим, хто хоче навчитися краще слухати, говорити щиро й будувати стосунки на основі довіри та взаємоповаги. Застосунок «Daily Hellos» створений для світу, якому не вистачає справжніх розмов — і може стати маленьким, але важливим кроком до більш людяного спілкування (рисунок 1.7). Проте він доступний тільки на операційній системі IOS.

Рисунок 1.7 – Екран із запитанням в застосунку «Daily Hellos»

Усі три застосунки мають одну ключову ціль — допомогти людям встановити якісний контакт через розмови. Вони спрямовані на покращення міжособистісного спілкування, розвиток емпатії та довіри між партнерами, друзями, родиною чи навіть у саморефлексії. Основою є спеціально підібрані запитання, що виступають каталізатором відкритих, щирих розмов.

Основна механіка кожного застосунку — подача запитань у структурованому вигляді. Тематичні блоки дозволяють користувачу обрати контекст: від романтичних бесід і питань про виховання до легких жартів для вечірок.

Отже, з існуючих рішень варто наслідувати велику різноманітність до категорій питань, наявність інтуїтивно зрозумілого інтерфейсу та легкий перегляд і збереження питань. Також цінним є поєднання легких і глибоких питань, що дозволяє варіювати настрій і рівень спілкування.

З іншої сторони не варто наслідувати надто агресивну монетизацію, наприклад, покупку питань поштучно, що може викликати роздратування. А також варто уникати нецікавих і поверхневих запитань.

1.3 Постановка задачі дослідження

Основною метою даного дослідження є розробка інноваційного підходу до покращення міжособистісних стосунків за допомогою технологій штучного інтелекту. На сьогоднішній день на ринку спостерігається явна нестача рішень, які б системно допомагали користувачам знаходити способи глибшого спілкування та відкриття особистості співрозмовника. Багато людей стикаються з труднощами при формулюванні питань, здатних викликати цікавість, бути достатньо глибокими для обговорення та здатними створити атмосферу довіри, а це, своєю чергою, є ключовим чинником у встановленні та розвитку здорових соціальних зв’язків.

Для подолання цієї проблеми в рамках дослідження буде використано API декількох сучасних моделей штучного інтелекту. Застосування різних ШІ дозволить не лише порівняти їх здатність генерувати релевантні та творчі

запитання, але й визначити, яка з моделей найбільш ефективно адаптується до завдань створення комунікаційно спрямованих питань. Отримання оцінок від користувачів, дозволять визначити якісні показники генерації питань та ступінь їх відповідності реальним потребам у спілкуванні.

Особливу увагу буде приділено параметру «Температура» (Temperature) у роботі з API кожного з обраних ШІ. Цей параметр визначає рівень випадковості та творчості у відповідях, що генеруються моделлю. Наприклад, при генерації коду або при завданнях, що вимагають точності, оптимальним значенням буде нульове, тоді як для завдань, які потребують творчого підходу або аналізу тексту, значення температури може варіюватися від 0,2 до 1,5. Такий підхід дозволить гнучко налаштовувати роботу моделі відповідно до специфіки завдання, забезпечуючи при цьому максимально релевантні результати.

Таким чином, дане дослідження має на меті вирішення декількох важливих завдань:

1. Завдяки комплексному підходу, що об’єднує технічні та психологічні аспекти, дослідження сприятиме розробці інструменту, який не тільки автоматизує процес генерації питань, але й допомагає користувачам будувати більш глибокі та змістовні міжособистісні стосунки шляхом кращого пізнання одне одного обговорюючи згенеровані питання.

2. Розробити інтуїтивно зрозумілий інтерфейс користувача та аби застосунок був оптимізований тим саме комфортний у користувані.

3. Реалізувати функціонал генерації питань для обговорень та впровадити механізм їх оцінювання користувачами для підвищення якості контенту.

4. Інтеграції кількох моделей шштучного інтелекту для збільшення різноманітності генерованих питань, так як кожна з моделей у роботі має свої відмінності.

2 АЛГОРИТМІЧНЕ ТА ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

2.1 Загальна структура проєктованої системи

Для реалізації проєктного рішення був обраний структурно-орієнтований підхід комп’ютеризованого проєктування систем (рисунок 2.1).

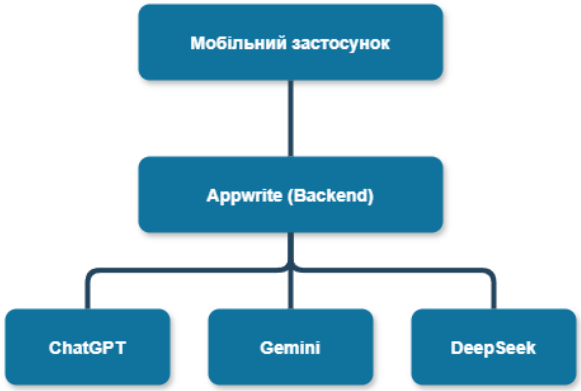


Рисунок 2.1 – Трирівнева архітектура системи

Для розробки мобільного застосунку, спрямованого на покращення міжособистісних стосунків шляхом використання штучного інтелекту для генерації питань пропонується трирівнева архітектура системи, що включає мобільний застосунок, бекенд на базі Appwrite та зовнішні сервісі штучного інтелекту такі як-от ChatGPT, Gemini та DeepSeek.

Мобільний застосунок виконуватиме функції відображення питань кінцевим користувачам, збору їхніх оцінок щодо якості питань та забезпечення базової взаємодії між користувачами. Також він буде місцем можливої монетизації в кооперації з бекендом. Сам бекенд, розгорнутий на платформі Appwrite, відіграє роль централізованого сховища даних, включаючи згенеровані питання, отримані оцінки та потенційну інформацію про користувачів. Крім того, він відповідатиме за встановлення зв'язку із зовнішнім сервісом штучного інтелекту з метою отримання нових питань на основі заданих промтів.

Зовнішніми сервісами штучного інтелекту буде використовуватись ChatGPT від OpenAI, Gemini від Google та DeepSeek від одноіменної компанії. У ChatGPT та

Gemini можливе використання файлів, що дозволить будувати базу з інформацією від досліджень та статей психологів, які будуть покращувати питання. Натомість, у DeepSeek на разі такий функціонал відсутній. Проте його перевага у тому що він дуже дешевий в порівнянні з іншими конкурентами.

На рисунку 2.2 та рисунку 2.3 можна побачити як основні дані передаються між модулями.

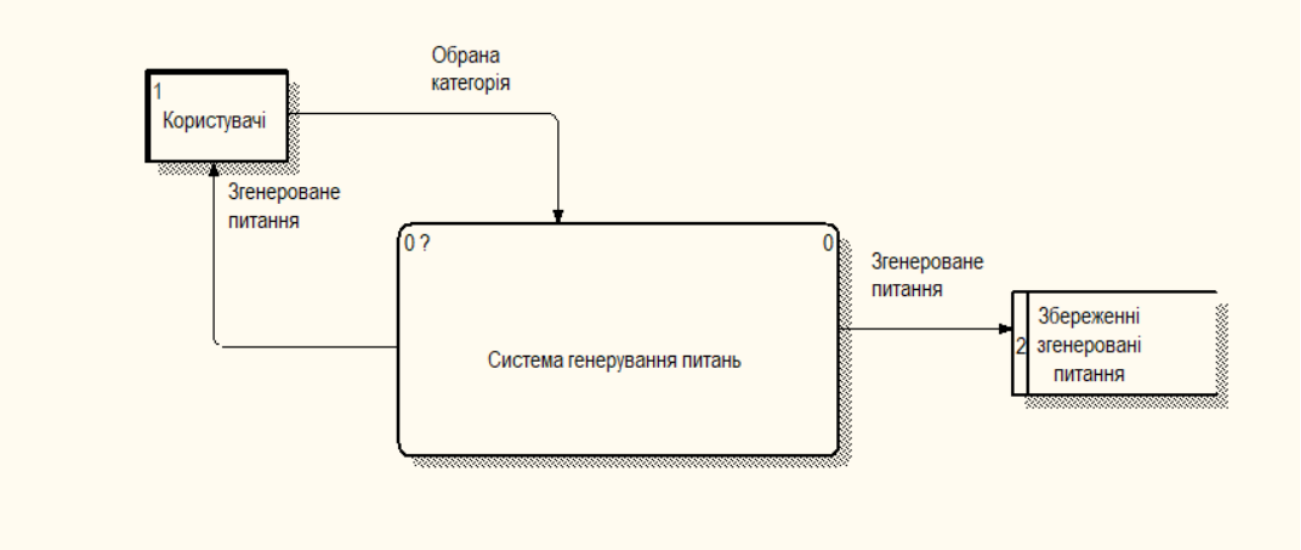


Рисунок 2.2 – Діаграма потоків даних

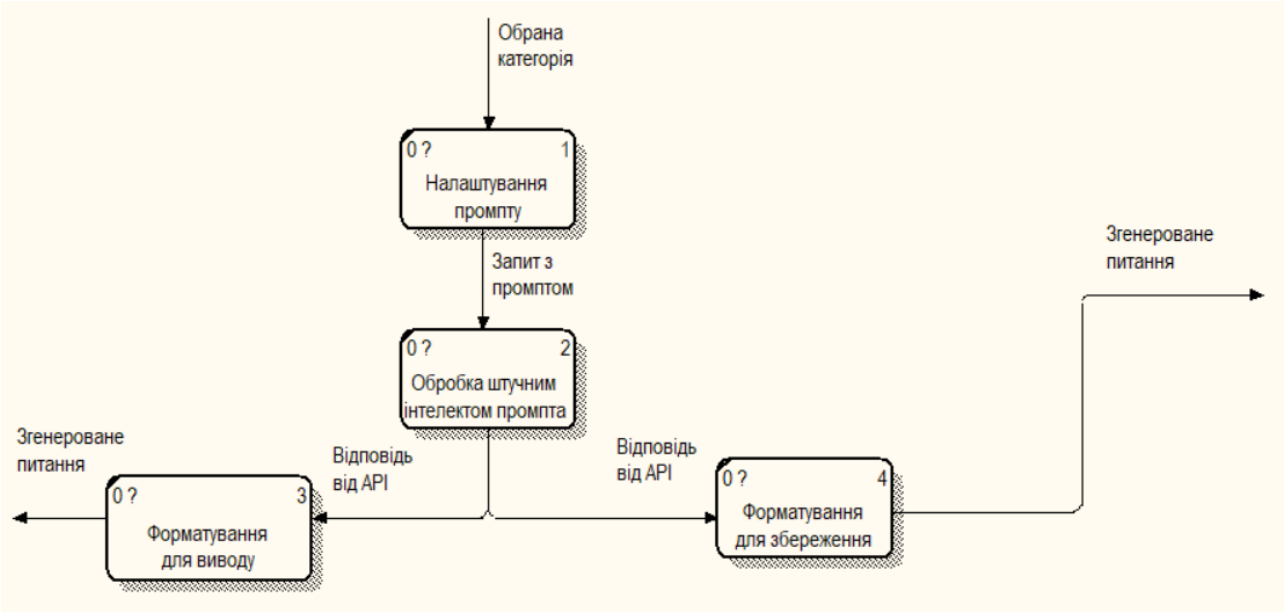


Рисунок 2.3 –Декомпозиція елементу «Система генерування питань»

На даних рисунках можна побачити, що користувач здійснює запит для генерування питання у певній категорії. Наступним кроком на бекенді буде здійснюватись налаштування промпта до обраної категорії. Після чого здійснюється запит на одну з модель штучного інтелекту для подальшого отримання відповіді у певному вигляді здатному до форматування.

Після генерації відповіді з питаннями від ШІ наступним кроком ці дані якраз відправляються на форматування, аби можна було показати юзеру та для БД, де зберігаються питання шляхом генерування.

2.1.1 Мобільна архітектура

Для розробки мобільного застосунку будуть використовуватись технології Flutter та Dart (рисунок 2.4). За допомогою Flutter є можливість розробляти застосунки одразу на усі існуючі платформи.

Отже, пишучи один код можна отримати застосунок для Android, IOS, Windows, MacOS, Linux, Web. Це значно пришвидшує розробку завдяки можливій компіляції проєкту на усі платформи. Також дана технологія дозволяє робити адаптивний дизайн, що означатиме що на всіх платформах застосунок буде виглядати задовільно. Також Flutter дозволяє робити чудові анімації, що буде добре позначатись на користувацькому досвіді.

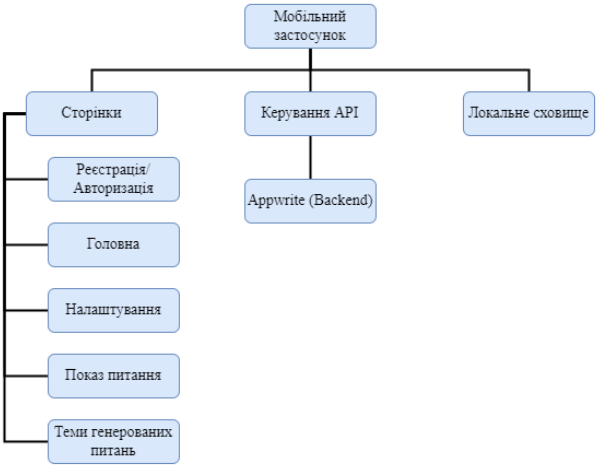


Рисунок 2.4 – Архітектура мобільного застосунку

2.1.2 Backend архітектура

Для реалізації бекенду був вибраний клауд сервіс Appwrite так як він має ряд переваг і достатню кількість інструментів для роботи з мобільним застосунком (рисунок 2.5).

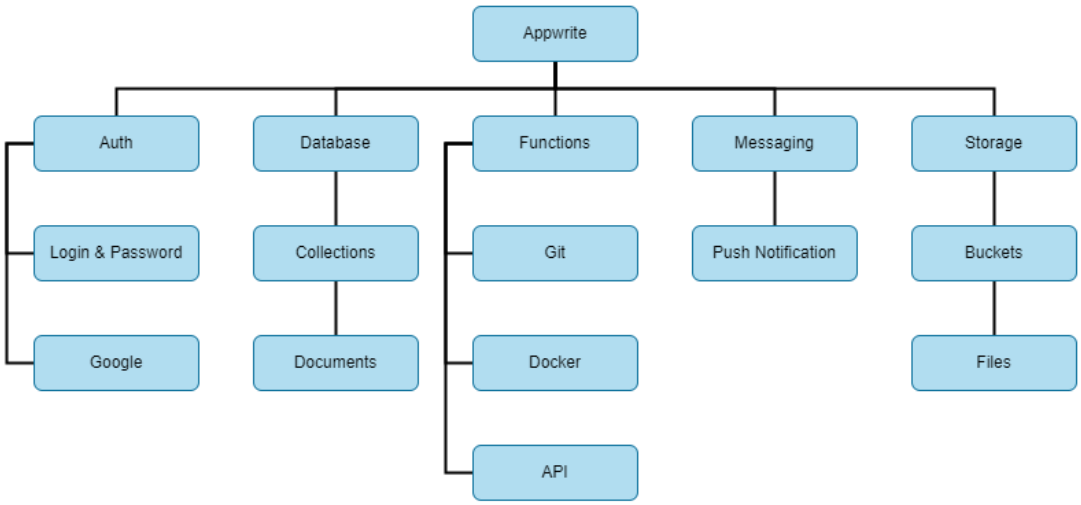


Рисунок 2.5 – Архітектура бекенду

В першу чергу наявність Auth модулю, котрий буде відповідати за авторизацію та ідентифікацію користувачів. Що дозволить керувати монетизацією для користувачів, так як планується надавати обмежену кількість питань безкоштовних, а безліміт на цю генерацію шляхом через підписку користувачами.

Також є важливий модуль Database – що відповідає за базу даних де будуть зберігатись параметри, промпти, згенеровані питання та статистика по цих ж питаннях, тобто як людям є дане питання чи воно було дійсно корисним чи ні.

Є модуль Functions – який буде використовуватись для генерування питань від штучного інтелекту і безпосередньо записувати в базу даних. Цей момент буде виокремлений, аби мати змогу вручну збільшувати кількість питань, щоб ті не повторювались чи не було якісь з ним проблем, а також за потреби перегенеровувати, якщо ті чи інші питання не підійшли. Тим самим покращуючи вміст наявних питань які будуть видаватись користувачам. А також це покращить взаємодію з застосунком так як не прийдеться тривалий час очікувати, поки

штучний інтелект згенерує запитання. Це буде перевагою, так як даний функціонал надасть можливість забезпечити безперебійну роботу застосунку і не буде проблем, якщо якийсь сервіс API перестане працювати належним чином.

За допомогою цих інструментів можливо реалізувати MVP, який буде мати в собі необхідну кількість функціоналу для того аби відображати основні функції. У майбутньому буде можливість створити функціонал, аби людям генерувалось дане питання і записуючи на нього відповіді, наприклад партнери чи друзі таким шляхом могли пізнавати одне одного.

Завдяки Appwrite є можливість швидко розгорнути процес розробки і пришвидшити основні моменти завдяки його функціоналу.

Інтеграція бекенду Appwrite із зовнішніми сервісами штучного інтелекту вимагає налаштування надійного каналу зв'язку між системами, де Appwrite виступає посередником, що спрямовує запити від мобільного застосунку до API сервісів ШІ. Це включає створення RESTful або GraphQL API, за допомогою яких бекенд відправлятиме промти для генерації питань та отримуватиме відповіді, що забезпечує асинхронну взаємодію між компонентами.

Ключовим технічним аспектом є налаштування аутентифікації та авторизації запитів. Appwrite може використовувати токени доступу або ключі API для забезпечення безпеки при передачі даних між сервером і зовнішніми ШІ сервісами, таким чином запобігаючи несанкціонованому доступу до сервісів та захищаючи конфіденційну інформацію користувачів.

Також важливою є обробка помилок та логування запитів. Бекенд має моніторити відповіді від ШІ сервісів, фіксувати помилки, пов'язані з недоступністю або некоректною роботою сервісів, і відповідним чином інформувати мобільний застосунок про статус виконання операцій. Цей підхід дозволяє оперативно виявляти і усувати технічні збої. Для оптимізації продуктивності реалізуються механізми кешування та балансування навантаження. Appwrite може зберігати кешовані відповіді або результати попередніх запитів, що зменшує затримки при повторних зверненнях до ШІ сервісів. Крім того,

використання черг повідомлень та обробки запитів у фоновому режимі дозволяє розподіляти навантаження та підвищувати ефективність системи.

Нарешті, інтеграція повинна враховувати можливість масштабування, що передбачає горизонтальне розширення ресурсів Appwrite для обробки зростаючої кількості запитів. Це забезпечує стабільну роботу платформи навіть при різких коливаннях навантаження, що є критично важливим для забезпечення високої доступності та швидкодії сервісу.

2.1.3 Архітектура взаємодії з API штучних інтелектів

Загалом усі бібліотеки здебільшого оптимізовані під ChatGPT так як через його першучергову появу на ринку він тепер вважається еталоном і часто береться як основа. По можливості у проєкті використовуються безкоштовні варіанти моделей. З ChatGPT буде використовуватись gpt-4o-mini, для Gemini - gemini-2.5-pro-exp-03-25, а для DeepSeek – deepseek-chat (рисунок 2.6).

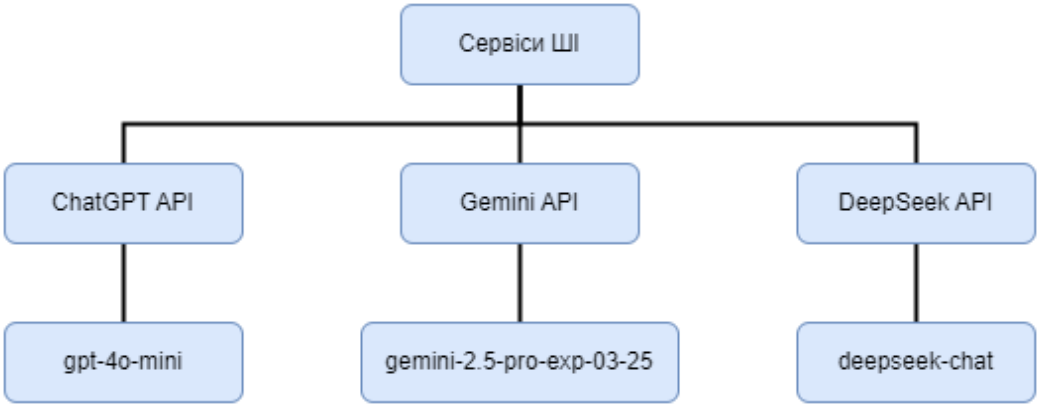


Рисунок 2.6 – Архітектура сервісів ШІ

Вони навчаються та тренуються за різними підходами, що зумовлено їхньою архітектурою, метою використання. Проте, попри ці відмінності, існують потенційно схожі методи навчання, оптимізації параметрів і попередньої обробки або очистки даних, які можуть застосовуватись у роботі з такими моделями для покращення результатів і стабільності. Наприклад використовують стандартний

підхід трансформерів із механізмом самоуваги (self-attention). Основна формула для обчислення уваги виглядає так:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (2.1)$$

де Q – матриця запитів;

K – матриця ключів;

V – матриця значень;

d_k – розмірність ключів.

Також використовується багатоголова увага (Multi-Head Attention), яка дозволяє моделі одночасно враховувати різні аспекти контексту.

Після шару уваги дані проходять через позиційно-незалежну нейронну мережу з нелінійною активацією. Типова формула для такого шару:

$$FFN(x) = \max(0, xW_1 + b_1)\,W_2 + b_2 \quad (2.2)$$

де x – вхідний вектор;

W_1, W_2 – матриці ваг першого і другого лінійного шару;

b_1, b_2 – відповідні вектори зсуву.

Модель навчається за принципом автогресивного прогнозування, тобто обчислює ймовірність наступного токена $P(w_{t+1}|w_1, \dots, w_t)$ та мінімізує крос-ентропію між передбаченням і реальним текстом.

При підготовці даних для тренування використовують алгоритми, які аналізують кожен зразок за ознаками «шуму» чи небажаного контенту. Наприклад, одна з ідей полягає у використанні класифікатора, який за допомогою softmax-шару визначає ймовірність того, що даний текст містить небажаний вміст:

$$P(noise \mid x) = softmax(Wx + b) \quad (2.3)$$

де:

де x — вектор ознак (наприклад, ембедінг вхідного тексту);

W — матриця ваг, яка трансформує вхідний вектор у простір класів;

b — вектор зміщення.

Також важливою складовою дослідження є розробка та оптимізація промптів, які використовуються для взаємодії з штучним інтелектом. З метою досягнення максимальної ефективності генерації питань, промпти будуть конструюватися з урахуванням рекомендацій, отриманих з практики професійних психологів. Це забезпечить не лише точне визначення тематики та контексту, але й сприятиме генеруванню питань, що відображають глибину людських емоцій та інтересів. Крім того, особливу увагу буде приділено формулюванню промптів таким чином, щоб отримані відповіді могли бути безпосередньо використані у подальшій розробці бекенд-систем, що забезпечить оперативну інтеграцію результатів у програмний код.

Для інтеграції моделей ШІ в систему буде використовуватись функціонал Appwrite з модуля Functions, де за допомогою скрипта буде виконуватись запит на генерування питань і вони відповідно будуть додаватись у базу даних звідки потім мобільний застосунок спокійно зможе їх підтягувати. У випадку якщо всі питання уже були переглянуті в автоматичному порядку буде запускатись скрипт на генерування питань, аби не було повторень, що дозволить отримувати унікальний досвід від згенерованих питань.

Також у сховищі будуть зберігатись промпти, які будуть безпосередньо надсилатись на API ШІ сервісів. Що дозволить, наприклад, промт інженеру займатись цими інструкціями, які можна буде потім використовувати. Також завдяки такому способу реалізації можливо буде аналізувати окрему базу даних, яка буде стосуватись виключно рекомендацій психологів для покращення людських відносин, що буде використовувати лише ШІ. У випадку використовуваних сервісів це буде стосуватись ChatGPT та Gemini, так як тільки вони мають такий функціонал.

Оскільки ChatGPT вже тривалий час присутній на ринку, для нього створено чимало бібліотек, які дозволяють спростити роботу з кодом без необхідності прописувати кожен API-запит окремо. Для налаштування взаємодії з ШІ слід визначити його роль за допомогою параметра “Developer”, в якому зазначається,

яку функцію виконує ШІ, яка його мета та яких принципів слід дотримуватись у відповідях.

За допомогою іншого параметра “User” — задаються очікування, тематика, рівень складності запитання тощо. Це сприяє покращенню якості відповідей і формулювання запитань. Якість запитань можна оцінити на основі зворотного зв’язку від користувачів.

Технічна інтеграція API ChatGPT дозволяє розробникам використовувати потужні мовні моделі для створення інтелектуальних сервісів, таких як чат-боти, автоматизовані відповіді та аналітичні інструменти. API надає можливість взаємодіяти з моделлю GPT через програмні запити, що дозволяє впроваджувати генерацію природної мови у різні додатки.

Завдяки простоті використання та гнучкості API ChatGPT можна легко інтегрувати у вебсайти, мобільні додатки, системи автоматизації клієнтської підтримки та багато інших рішень. Це відкриває широкі можливості для автоматизації завдань, покращення взаємодії з користувачами та розширення функціоналу програмних продуктів.

Для тестування та інтегрування з API Gemini – Google надає можливість користуватись Google AI Studio. У якому можна протестувати різноманітні речі котрі стосуються Gemini. Різноманітні моделі, сценарії роботи, потестувати інтеграції та як параметри впливають на відповідь тої чи іншої моделі штучного інтелекту.

Однією з ключових переваг Google AI Studio є її інтеграція з Google Cloud. Це дозволяє розробникам легко використовувати інші сервіси Google Cloud, такі як Cloud Storage та Vertex AI, для зберігання та обробки даних. Крім того, платформа надає інструменти для моніторингу та керування моделями, що полегшує їх розгортання та підтримку.

Google AI Studio підтримує різні фреймворки машинного навчання, такі як TensorFlow та PyTorch, що дає розробникам можливість використовувати звичні

інструменти. Платформа також надає інструменти для візуалізації даних та результатів навчання моделей, що допомагає краще розуміти їх роботу.

Інтеграція з API Gemini дозволяє створювати різноманітні додатки, такі як чат-боти та віртуальні помічники, інструменти для аналізу та обробки тексту, додатки для розпізнавання та аналізу зображень, а також системи для генерації контенту.

API DeepSeek пропонує широкий спектр можливостей для інтеграції AI-моделей у різні проекти. Його ключова сила полягає в універсальності: від обробки природної мови (NLP) до роботи з кодом.

Наприклад, розробники можуть використовувати API для генерації тексту, перекладу, стиснення інформації або аналізу тональності. Для програмістів особливо корисними будуть функції автодоповнення коду, налагодження або навіть конвертації між мовами програмування.

Окрім стандартних задач, DeepSeek підтримує спеціалізовані моделі для галузей на кшталт фінансів чи медицини, що дозволяє адаптувати рішення під конкретні бізнес-потреби, такі як семантичний пошук або автоматизація звітів.

Інтеграція з API реалізується через зручні та сучасні технології. Основною точкою входу є RESTful API з використанням HTTP-запитів (наприклад, POST для відправки даних), що підходить для більшості сценаріїв. Для додатків, які потребують миттєвої взаємодії, як чат-боти або інструменти реального часу, доступна інтеграція через WebSockets.

Розробникам також пропонуються SDK для популярних мов, таких як Python, JavaScript або Java, що спрощує роботу з API без необхідності писати низькорівневий код. Безпека забезпечується через API-ключі або OAuth 2.0, а всі дані передаються з шифруванням TLS, що критично для роботи з конфіденційною інформацією.

Prompt Engineering – це спеціалізація, що зосереджена на розробці та оптимізації текстових запитів (prompt'ів) для взаємодії з системами штучного інтелекту, зокрема з великими мовними моделями.

Головною метою є створення таких інструкцій, які дозволяють ШІ правильно інтерпретувати завдання та надавати точні, корисні та релевантні відповіді. Завдяки чіткому формулюванню запитів можна максимально використати потенціал моделі, враховуючи її сильні сторони і обмеження.

Основні задачі Prompt Engineering включають розробку оптимальних інструкцій, постійне тестування різних варіантів запитів і аналіз отриманих результатів.

Процес зазвичай є ітеративним: інженери експериментують з формулюванням промптів, вносять корективи на основі аналізу відповідей, щоб усунути недоліки та підвищити ефективність роботи ШІ. Такий підхід дозволяє адаптувати взаємодію з моделлю до конкретних завдань та умов використання.

Нюанси цієї діяльності пов’язані з особливостями мови, контексту та стилістики, які мають значний вплив на розуміння моделі. Правильний вибір слів, структура запиту та врахування специфічних параметрів моделі, таких як температура генерації або максимальна довжина відповіді, є критично важливими для отримання бажаних результатів. Крім того, розуміння обмежень самої моделі дозволяє уникнути небажаних наслідків і підвищити якість відповіді.

Обов’язки промпт-інженерів охоплюють широку сферу завдань, пов’язаних із формулюванням, тестуванням та вдосконаленням запитів (промптів) для ефективної взаємодії з системами штучного інтелекту. Зокрема, до їхньої роботи входить створення точних і релевантних інструкцій, які дозволяють ШІ виконувати завдання з максимальною ефективністю, а також аналіз результатів для подальшого покращення запитів.

Крім того, промпт-інженери беруть участь у дослідженні нових можливостей генеративного ШІ, експериментують із різними підходами до побудови запитів та розробляють внутрішні бібліотеки шаблонів для типових сценаріїв. Цей комплексний підхід не лише забезпечує ефективне застосування ШІ в різних сферах, а й сприяє постійному вдосконаленню технологій і розвитку ШІ-інфраструктури загалом.

2.2 Алгоритмічне забезпечення

На рисунку 2.7 представлено загальний алгоритм роботи зі системою генерації питань. Описаний алгоритм є про систему генерації запитань за допомогою штучного інтелекту. Процес починається зі старту роботи користувача в системі після авторизації чи реєстрації.

На початку алгоритму описується момент коли користувач може вийти за бажання із мобільного застосунку. Якщо так — система зупиняється. Якщо ж користувач продовжує, він переходить до вибору категорії питання, наприклад, це може бути дружба, для пар, про побут.

Після вибору категорії система формує текстовий запит (так званий «промпт») для надсилання до API штучного інтелекту. Цей запит містить інструкції, що саме має згенерувати ШІ. Коли запит готовий, він надсилається до відповідного API, і система очікує на відповідь.

Отримавши відповідь, система обробляє її — перевіряє структуру, формат, а також зміст згенерованого питання.

Далі це питання зберігається у внутрішній базі даних. Після збереження питання виводиться користувачу для ознайомлення.

Після перегляду згенерованого питання користувач має змогу надати оцінку. Якщо оцінка надана, вона зберігається в системі та може бути використана для подальшого аналізу якості генерації.

Потім система повертається до початкової перевірки — чи бажає користувач завершити роботу, чи продовжити з новим питанням.

Таким чином, цикл може повторюватися до тих пір, поки користувач не вирішить завершити роботу.

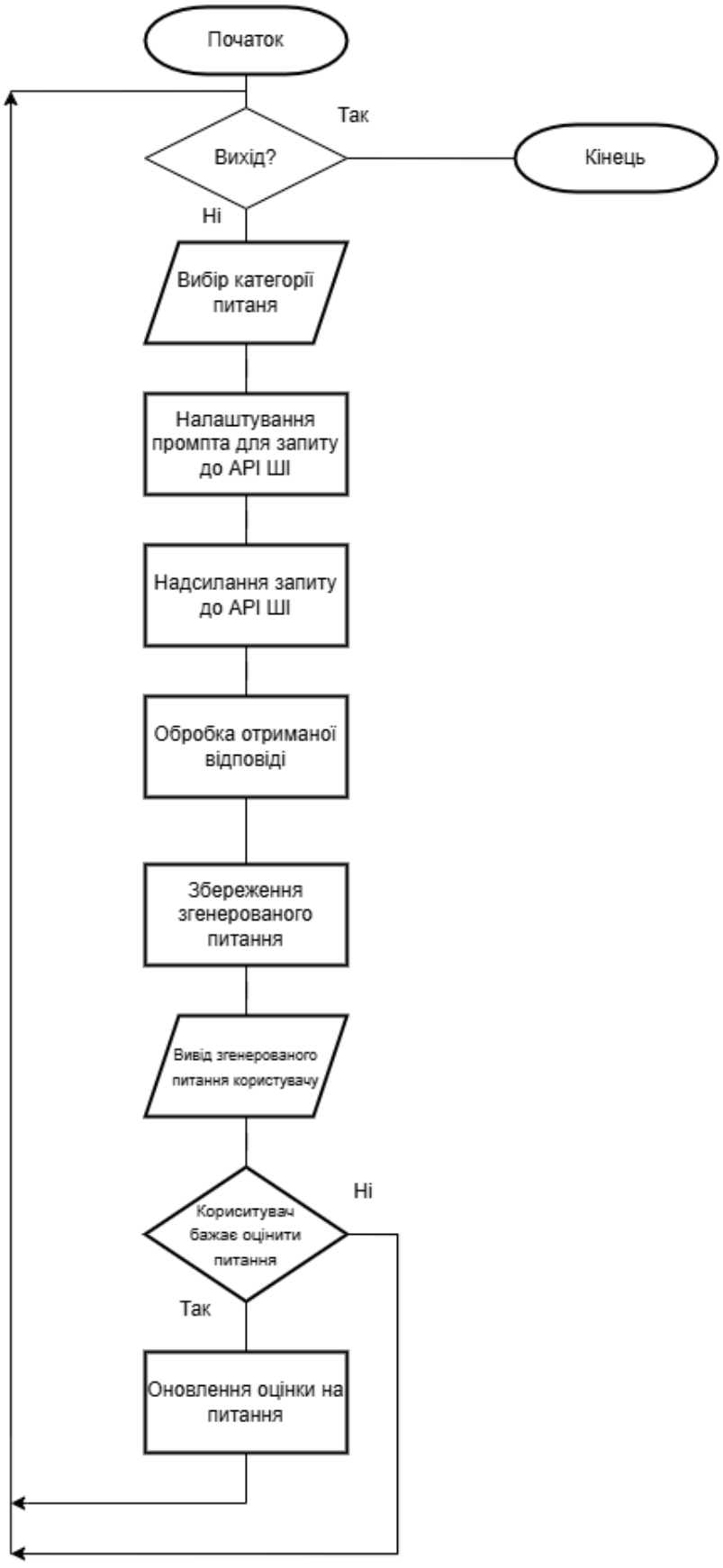


Рисунок 2.7 – Схема алгоритму роботи системи генерування питання

2.3 Інформаційне забезпечення

Під час роботи з API необхідно правильно сформувати промпт. Промпт — це набір інструкцій для моделі, простіше кажучи, те, що потрібно отримати. Такі промпти слід писати англійською мовою, оскільки це значно підвищує розуміння моделі завдання, що потрібно виконати. Це пов’язано з тим, що основними даними для навчання моделей штучного інтелекту є англійська мова.

Далі визначаються параметри та характеристики бажаного результату. Якщо потрібно отримати згенеровані питання, це слід чітко вказати. Також визначається тематика категорій і мета, для якої вони призначені, щоб модель зрозуміла, який результат очікується.

Окрім змістовної частини, важливо також прописати вимоги до форматування відповіді. Це необхідно для подальшої автоматичної обробки та парсингу даних у застосунку, забезпечуючи коректне зчитування та відображення інформації. Для більшої наочності і уникнення непорозумінь може бути додано приклад, який демонструє очікуваний формат вихідних даних. Такий підхід сприяє стандартизації результатів і полегшує їх інтеграцію у подальші етапи роботи.

Таким чином, остаточний вигляд промпта набуває вигляд: «Generate a list of questions on the topic of topic which user said that are designed to encourage people to discuss and get to know each other better. The questions should be open-ended and promote meaningful conversation. Return as json response as example {\"questions\":[\"question1\",\"question2\",\"etc\"]}».

2.3.1 Опис вхідної та вихідної інформації

Вхідна та вихідна інформація, яка обробляється в межах функцій предметної області, що автоматизується подана в таблиці 2.1. У таблиці можна побачити, яку інформацію необхідно подати на вхід та які вимоги, аби отримати у вихідній інформації необхідні дані у необхідному форматі.

Таблиця 2.1 – Вхідна та вихідна інформація

	Інформація	Подробиці
Вхідна інформація	Промпт	Інструкції про те що ми бажаємо отримати в результаті
	Формат	Формат у якому бажаємо отримати відповідь від AI через API
	Категорія (тема питання)	Категорія генерованих питань, які побажав отримати користувач.
Вихідна інформація	Відповідь від API ШІ у форматі JSON	Відповідь від штучного інтелекту на базі вхідної інформації.

2.3.2 Концептуальне інфологічне проєктування

Глобальна інфологічна модель даних (рисунок 2.8) подає загальну структуру даних для всієї системи, охоплюючи всі сутності та їхні взаємозв’язки на концептуальному рівні.

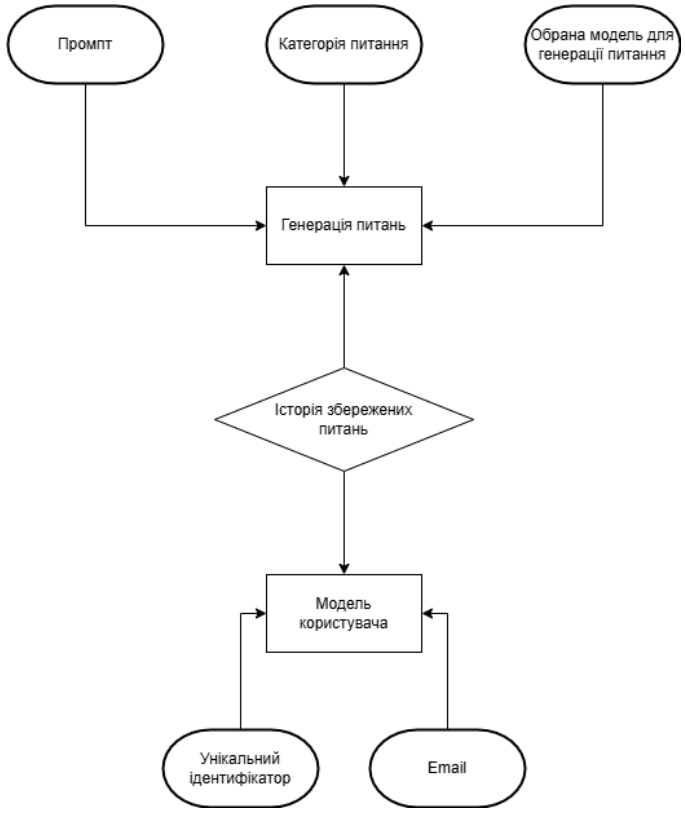


Рисунок 2.8 – Інфологічна модель бази даних

Локальна інфологічна модель даних (рисунок 2.9) поглиблено описує окремі підсистеми або компоненти системи, уточнюючи інформацію для кожної сутності та відображаючи внутрішні взаємозв’язки в межах конкретної області або підсистеми.

Рисунок 2.9 – Інфологічна модель бази даних

Концептуальне інфологічне проєктування дозволяє точно визначити структуру даних і їхні взаємозв’язки, що становить важливу основу для подальшої розробки програмного модуля.

2.3.3 Проєктування глобальної даталогічної моделі даних

ER-модель (рисунок 2.10) — це модель даних, яка відображає концептуальну схему предметної області систем з інтегрованими елементами штучного інтелекту.

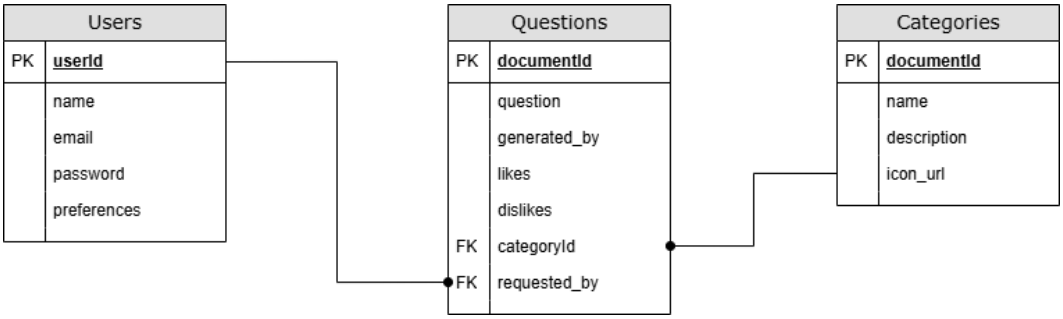


Рисунок 2.10 – Модель даних у вигляді ERD (у нотації IDEF1X)

Реляційна модель (рисунок 2.11) дає змогу визначити структуру бази даних і встановити зв’язки між таблицями, враховуючи вимоги до генерації запитань за допомогою штучного інтелекту.

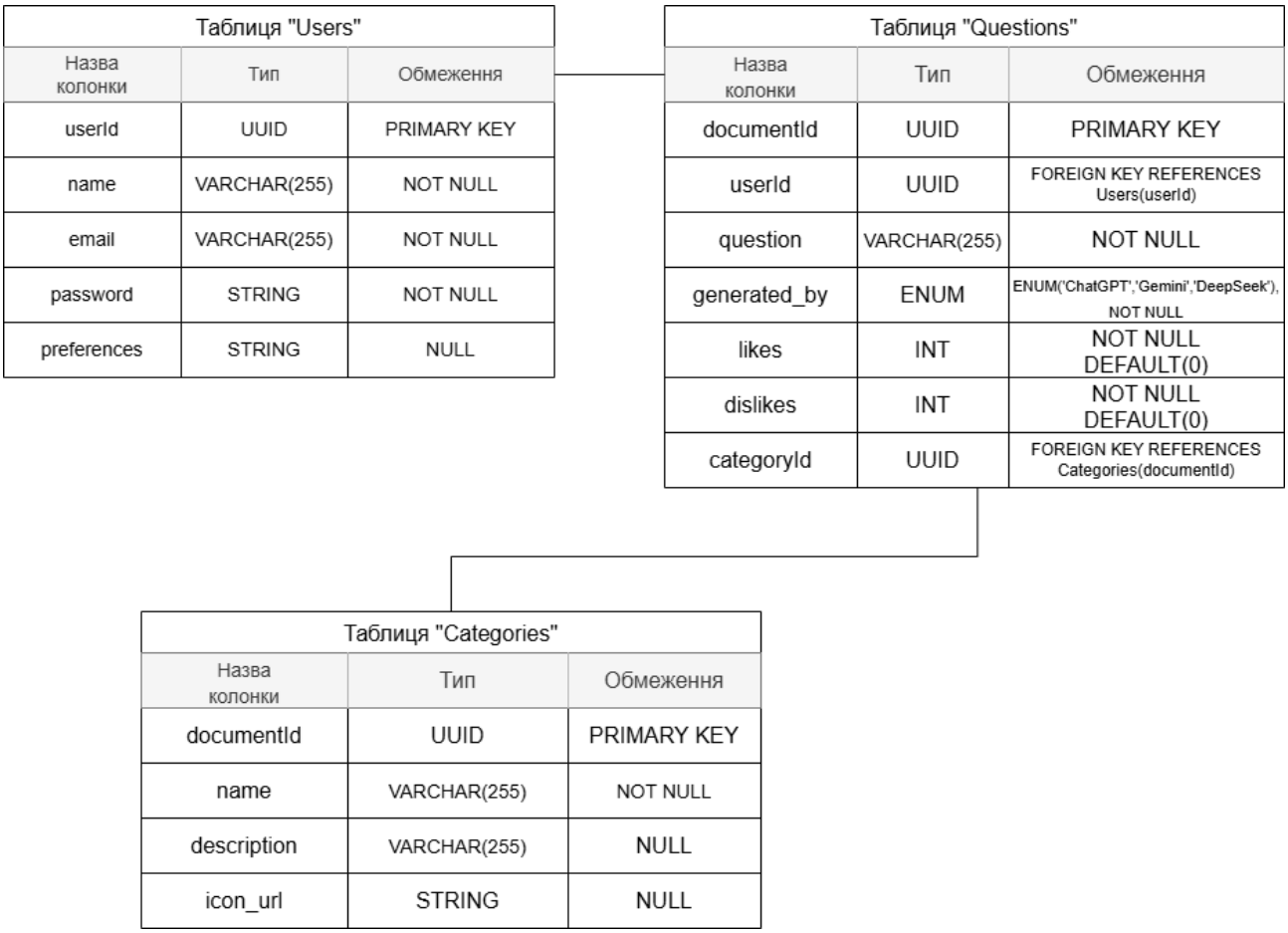


Рисунок 2.11 – Реляційна модель бази даних

Дана реляційна модель структурує дані так, аби згенеровані питання зберігали оцінку про себе, відповідали певній категорії та були зрозуміло за допомогою якої моделі штучного інтелекту були згенеровані і ким ця генерація була ініційована.

2.3.4 Проєктування фізичної моделі даних

Проєктування фізичної моделі даних (рисунок 2.12) передбачає детальне визначення структури бази даних, зокрема її таблиць та атрибутів.

У даній фізичній моделі даних використовуються наступні типи даних для відповідних полів:

— UUID (Universally Unique Identifier): 128-бітний ідентифікатор, що використовується для гарантування унікальності записів у розподілених системах.

— VARCHAR(N): Стрічковий тип даних змінної довжини, що може зберігати до N символів.

— STRING: Загальний тип для зберігання текстових даних, довжина якого може відрізнятися залежно від конкретної системи управління базами даних.

— ENUM: Тип даних, який дозволяє полю містити лише одне значення зі заздалегідь визначеного списку дозволених значень.

— INT: Ціле число, що використовується для зберігання числових значень без десяткових знаків.

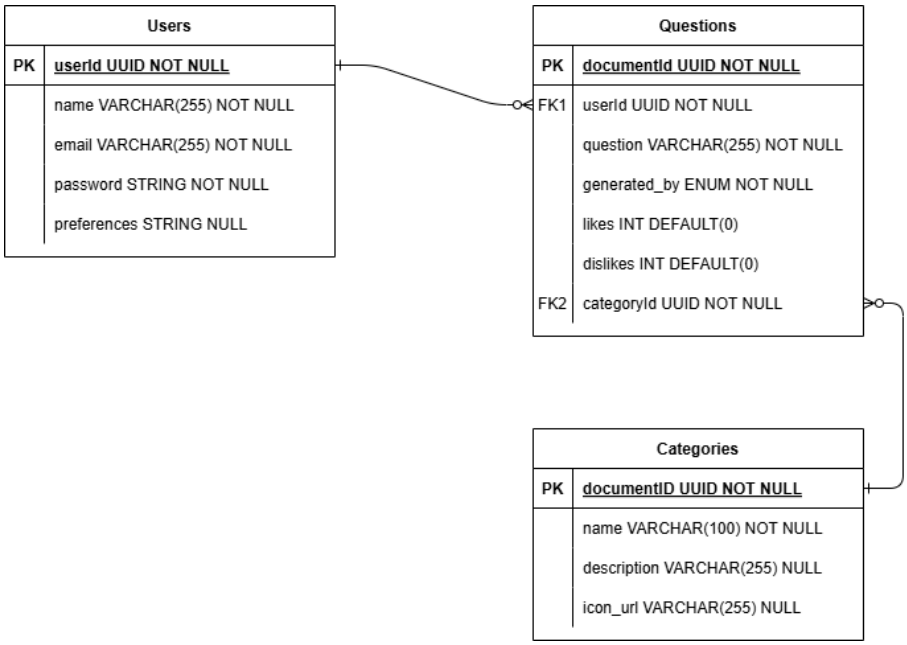


Рисунок 2.12 – Фізична модель даних

2.3.5 Програмна реалізація бази даних

Так як використовується бекенд Appwrite для збереження даних, то через візуальний редактор необхідно створити базу даних і кожна окрема таблиця там це

колекція. Для таблиці «Users», використовується їхній сервіс Auth, через який відбувається також взаємодія з зовнішніми сервісами для авторизації, таких як-от Google, Apple, Github тощо.

Після створення бази даних та таблиці(колекції) «Questions» додані наступні атрибути (рисунок 2.13):

— Question – тут буде зберігатись саме питання, приймає тип VARCHAR з обмеженням по кількості символів, а також відмічене як необхідне до заповнення.

— Generated_by – використовується тип ENUM так як треба обмежити лиш кількома значенням такими як-от ChatGPT, Gemini та DeepSeek.

— Likes та Dislikes – мають цілочисельний тип int та за замовчуванням присвоюються до нуля.

— Category – тип тут також UUID буде проте у сервісі це як Relation. Так є бо він буде під капотом утворювати зв’язок.

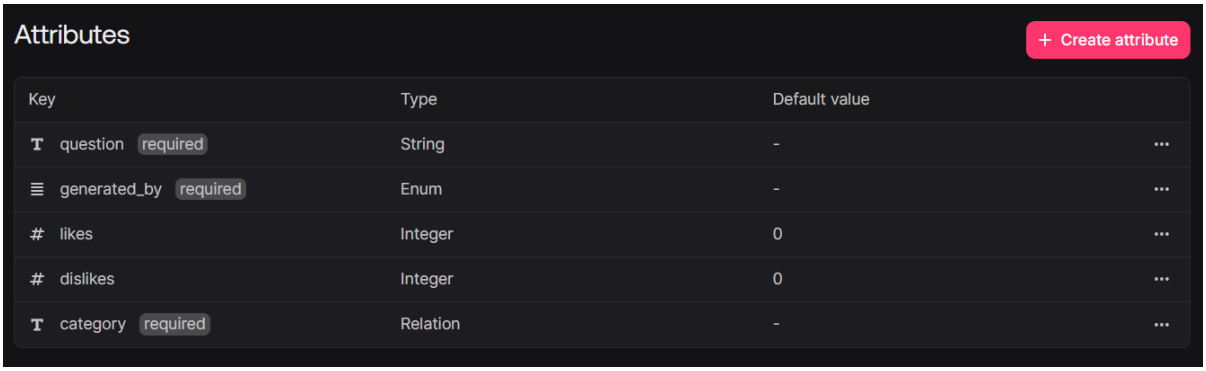


Рисунок 2.13 – Відображення заданих атрибутів таблиці «Questions»

Після створення бази даних та таблиці (колекції) «Categories» додані наступні атрибути (рисунок 2.14):

— Name – тут буде зберігатись назва категорії. Приймає тип VARCHAR з обмеженням по кількості символів, а також відмічений як необхідний до заповнення та унікальний, що забезпечує унікальність назви кожної категорії.

— Description – використовується для зберігання розширеного опису категорії, приймає тип VARCHAR з обмеженням по кількості символів і є необов'язковим для заповнення.

— Icon_url – цей атрибут призначений для зберігання URL-адреси іконки, пов'язаної з категорією. Має тип STRING і є необов'язковим для заповнення. Іконки можуть бути завантажені у сховище Appwrite (Storage), а потім посилання на них збережено тут.

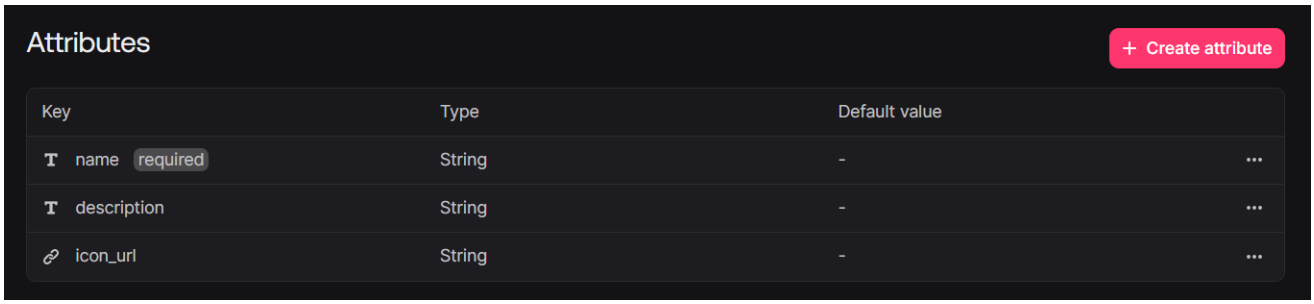


Рисунок 2.14 – Відображення заданих атрибутів таблиці «Categories»

Завдяки використанню Appwrite Database досягається надійність та ефективна роботи бази даних.

3 ПРОГРАМНО-ТЕХНОЛОГІЧНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

3.1 Реалізація програмного забезпечення

Структурна схема (рисунок 3.1) мобільного застосунку наочно демонструє складові елементи, а також зв’язки та взаємодію між ними. На схемі відображено процес, за якого користувач через інтерфейс обирає категорію для генерації запитань, після чого відповідні дані надсилаються на Backend, реалізований за допомогою платформи Appwrite.

Backend, у свою чергу, формує запити до трьох моделей штучного інтелекту — ChatGPT, Gemini і DeepSeek — для створення запитань. Отримані результати повертаються до Backend, де проходять обробку, зберігаються у базі даних (Appwrite Database) і передаються назад в інтерфейс для подальшої оцінки користувачем.

Оцінки, що надає користувач, також надсилаються до Backend і фіксуються в базі даних. Така архітектура забезпечує централізоване управління усіма етапами: від генерації запитань до їх зберігання та оцінювання, що сприяє ефективній і злагодженій роботі всієї системи.

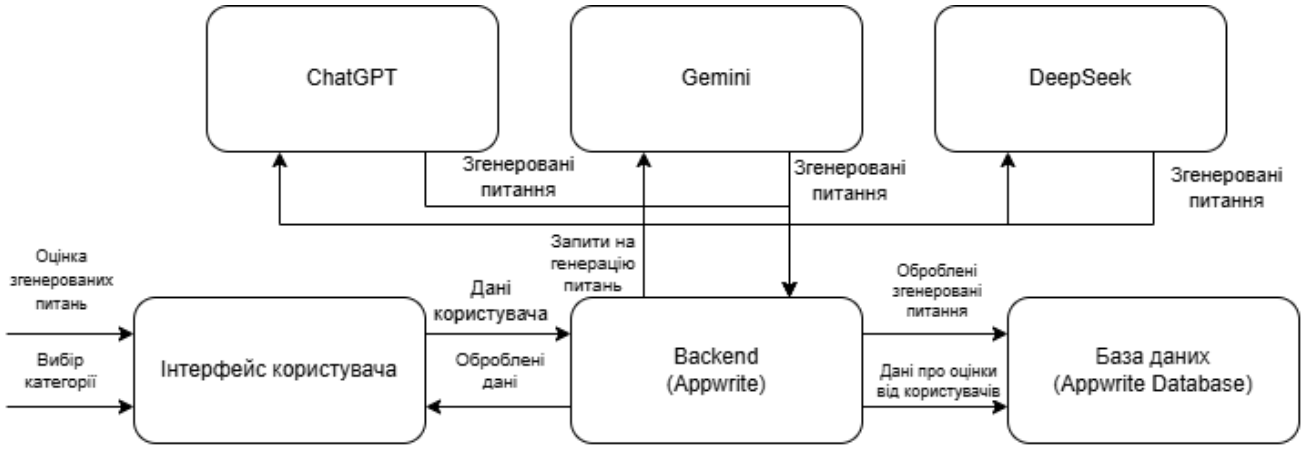


Рисунок 3.1 - Структурна схема мобільного застосунку

UML-діаграма класів (додаток Б) реалізує основну бізнес-логіку програмної системи. Клас «RegisterPage» відповідає за створення інтерфейсу сторінки реєстрації. Він використовує текстові контролери для вводу електронної пошти та пароля, а також глобальний ключ форми. Метод build формує зовнішній вигляд сторінки. Для ініціації процесу реєстрації використовується RegisterUseCase.

Клас «RegisterUseCase», який належить до доменного шару, реалізує бізнес-логіку реєстрації. Його метод call приймає об'єкт UserEntity, який містить дані користувача, і передає їх до RegisterRepository.

Клас «RegisterRepositoryImpl», розміщений у шарі даних, є конкретною реалізацією RegisterRepository. Він використовує RegisterDataSource для передавання даних користувача в зовнішні джерела.

Клас «RegisterDataSourceImpl» реалізує інтерфейс RegisterDataSource. Метод registerUser використовує екземпляр класу Account з ApiService для виконання API-запиту та реєстрації користувача.

Клас «LoginPage» надає інтерфейс для авторизації. Він містить контролери для введення email та пароля, кнопку входу, а також метод build, який виводить відповідну форму. Після авторизації користувача перенаправляє на головну сторінку.

Клас «MyHomePage» є головною сторінкою після входу. Він керує навігацією між різними сторінками, такими як QuestionPage, CategoryPage, і виводить компоненти типу CategoryCard.

Клас «QuestionPage» відповідає за відображення питань або. Він приймає параметр titleName, який визначає заголовок сторінки, і реалізує метод build, що відповідає за побудову візуального інтерфейсу з урахуванням заданих питань або форм.

Клас «CategoryCard» є віджетом, який представляє певну категорію. Він використовує дані, такі як назва, іконка та опис, і формує картку у форматі UI.

Клас «Theme» відповідає за налаштування тем оформлення додатку. Класи ExtendedColor та ColorFamily визначають палітри кольорів, які використовуються

для різних тем — світлої, темної, контрастної. Всі кольорові схеми об'єднуються у TextTheme та ThemeData, що використовуються для стилізації застосунку.

UML-діаграма станів на рисунку 3.2 відображає різні стани, в яких можуть знаходитися елементи графічного інтерфейсу користувача, і переходи між цими станами внаслідок певних дій користувача або системних подій.

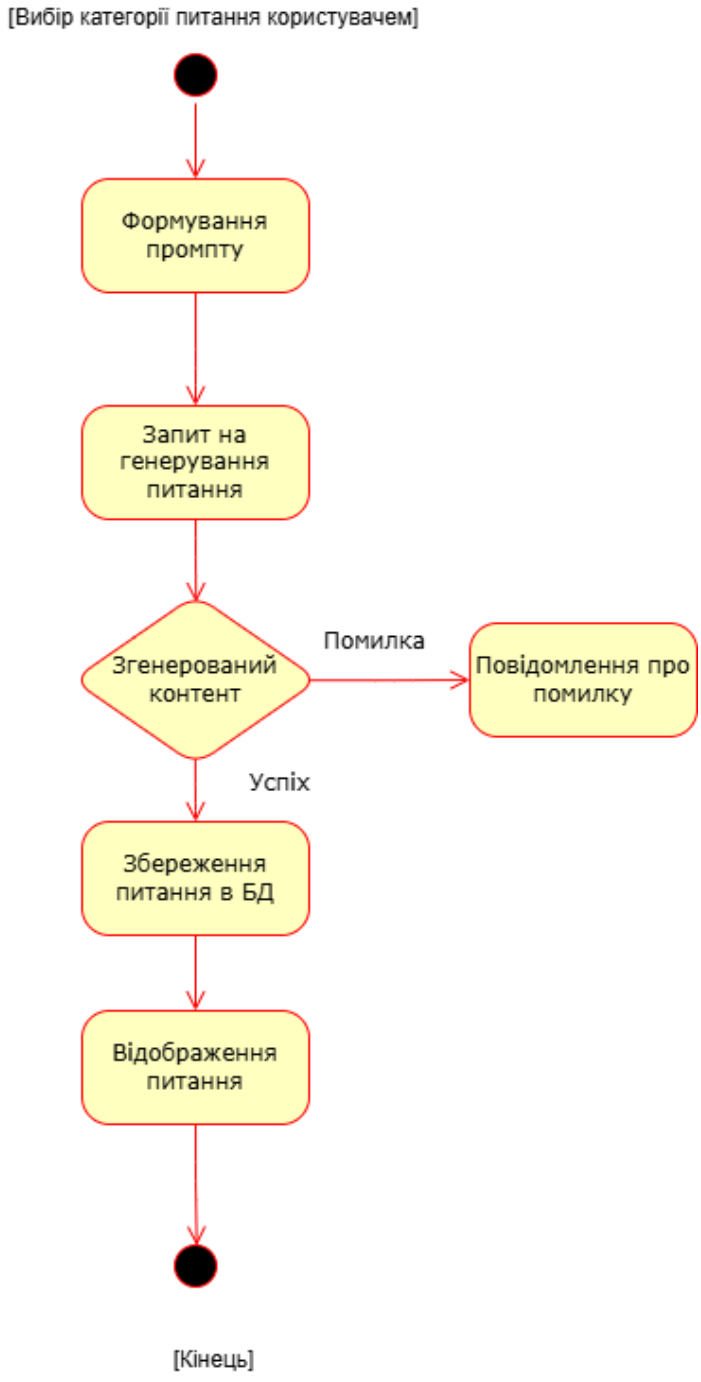


Рисунок 3.2 – UML-діаграма станів

Процес починається з того, що користувач ініціює дію, обираючи певну категорію питання. Це є відправною точкою для подальшої взаємодії. Після вибору категорії система формує промпт (запит) для генерації питання. Цей промпт, базується на обраній категорії. Сформований промпт відправляється до зовнішнього сервісу або внутрішнього механізму для генерування питання. Це є викликом API до моделі ШІ (наприклад, ChatGPT, Gemini, DeepSeek). Система в свою чергу очікує відповідь від API штучного інтелекту. На цьому етапі відбувається перевірка результату: якщо «Успіх»: питання було успішно згенеровано, процес переходить до наступного стану. Якщо «Помилка»: наприклад виникла помилка під час генерації контенту (помилка API, невірний промпт, технічний збій), процес переходить до стану "Повідомлення про помилку".

У випадку помилки система інформує користувача (або адміністратора) про проблему. Це може бути спливаюче повідомлення, логування помилки тощо. Після цього процес, ймовірно, завершується або повертається до початкового стану для нової спроби. Якщо генерація пройшла успішно, згенероване питання зберігається в базі даних. Згідно з реляційною моделлю, це відповідає додаванню нового запису до таблиці Questions із зазначенням тексту питання, джерела генерації (generated_by), зв'язком з користувачем (userId) та категорією (categoryId). Після успішного збереження питання в базі даних, воно відображається користувачеві. Процес завершується після відображення питання користувачеві.

Для виконання даної роботи була обрана технологія Flutter для розробки мобільного застосунку «Gather Together». А для серверної частини з яким буде працювати застосунок – Appwrite.

Flutter було обрано для розробки застосунку з кількох ключових причин:

По-перше, кросплатформеність Flutter дозволяє створювати мобільний застосунок одразу для Android і iOS, використовуючи єдину кодову базу. Це суттєво знижує витрати часу та ресурсів на розробку й підтримку проєкту. А також є можливість випуску цього ж застосунку для Windows, Web, MacOS та Linux.

По-друге, Flutter має високу продуктивність завдяки тому, що використовує власний рушій рендерингу, минаючи нативні компоненти. Це забезпечує плавну анімацію, швидку реакцію інтерфейсу та зручне користування навіть на менш потужних пристроях.

Крім того, Flutter пропонує багатий набір віджетів для створення гарного, сучасного UI, який можна гнучко кастомізувати під потреби бренду або користувачів. Це дозволяє швидко створити візуально привабливий інтерфейс.

Ще одна перевага — гаряче перезавантаження (hot reload), що дає змогу миттєво бачити зміни в коді без перезапуску застосунку. Це значно прискорює процес розробки й тестування.

І зрештою, Flutter має сильну спільноту, велику кількість пакетів та активну підтримку від Google, що робить його надійним вибором для створення сучасного мобільного застосунку.

Лістинг бізнес-логіки виконаний у додатку А. Лістинг генерування питань за допомогою ШІ виконаний у додатку В.

В той час, Appwrite було обрано як бекенд-рішення для застосунку через його сучасний підхід до управління даними, зручність у розгортанні та підтримку ключових функцій, необхідних для мобільного застосунку.

Насамперед, Appwrite — це повноцінний open-source Backend-as-a-Service (BaaS), який забезпечує все необхідне: автентифікацію, управління базою даних, зберігання файлів, функції серверної логіки тощо. Це дозволяє зосередитися на фронтенд-розробці без необхідності писати складний серверний код з нуля.

Appwrite має простий REST та Realtime API, який легко інтегрується з Flutter через офіційний SDK. Це значно спрощує розробку, дозволяючи швидко реалізовувати авторизацію, управління користувачами та збереження контенту.

Важливою перевагою є також різноманітна кількості видів автентифікації — Appwrite підтримує різні методи входу (email, OAuth, анонімний вхід), що дає змогу адаптувати систему під конкретні потреби користувачів.

Крім того, Appwrite легко розгортається локально або в хмарі через Docker. Це надає повний контроль над даними, що важливо для безпеки та масштабованості проєкту. Так як, активна спільнота та регулярні оновлення роблять Appwrite надійним та сучасним бекенд-рішенням, ідеальним для швидкої розробки мобільних застосунків із повноцінною серверною логікою, його і було обрано для серверної частини.

3.2 Інтерфейс користувача

Початкова сторінка застосунку (рисунок 3.3) дозволяє користувачам пройти авторизацію за допомогою стандартних методів таких як-от з використанням email та паролю або увійти через свій Google акаунт. Це допоможе ідентифікувати користувача у системі котрий генерує питання.

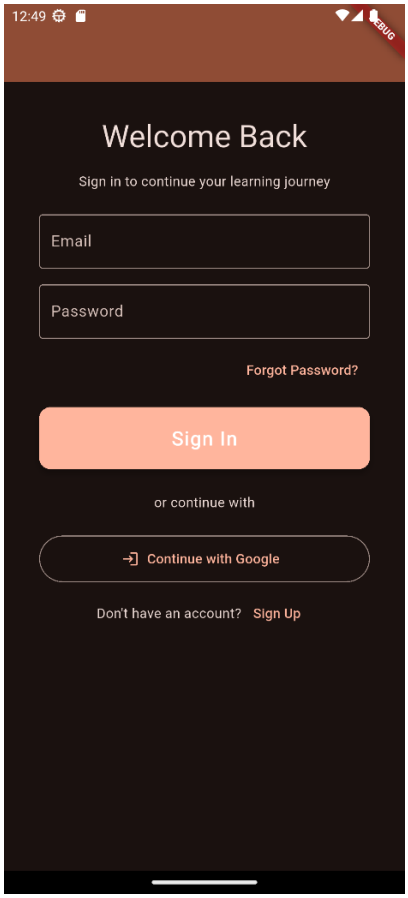


Рисунок 3.3 – Сторінка авторизації

На головному екран (рисунок 3.4) де є різні категорії користувач може обрати таку, до котрої хоче аби було згенероване питання. Кожна категорія містить іконку, назву та короткий опис про цю категорію, щоб при виборі категорії користувач зможе оцінити чого йому очікувати. Далі користувач нажимає кнопку на генерування питання і його перекидує на наступний екран.

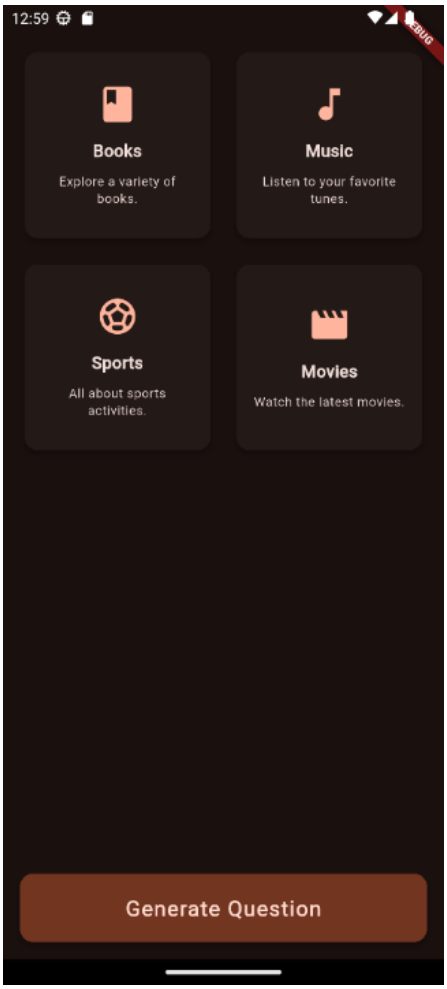


Рисунок 3.4 – Головний екран застосунку

Після не довгого очікування користувач отримує на екрані (рисунок 3.5) питання, яке може обговорювати зі своїм співбесідником. Також він має можливість оцінити якість даного питання, аби в майбутньому питання з великою кількістю негативний оцінок не відображались користувачам. Також користувач має

можливість повідомити про це питання розробникам, аби його прибрали, якщо його вважають не коректним.

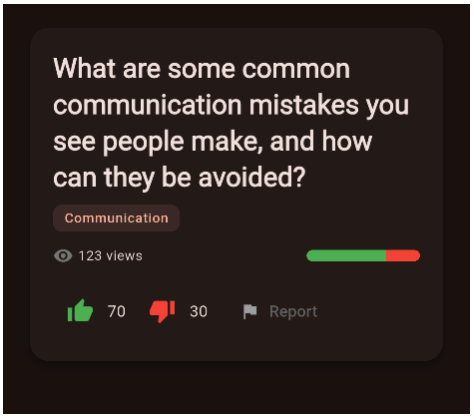


Рисунок 3.5 – Сторінка зі згенерованим питанням

Завдяки використання технології Flutter, застосунок виглядає привабливо та зрозуміло для користування, а також він працює швидко.

3.3 Тестування розробленого мобільного застосунку

Тестування є невід'ємною частиною процесу розробки мобільного застосунку, що дозволяє виявити та усунути помилки на ранніх етапах. У контексті мобільної розробки на Flutter передбачено вбудовану підтримку тестування, яка охоплює декілька рівнів: модульне тестування (unit testing), тестування віджетів (widget testing) та інтеграційне тестування (integration testing). Це дозволяє забезпечити високу якість мобільного застосунку, перевіряючи як окремі компоненти, так і взаємодію між ними.

Модульне тестування у Flutter реалізується за допомогою пакету test. Цей тип тестування використовується для перевірки логіки окремих функцій, сервісів або класів, які не залежать від фреймворку Flutter. Наприклад, тести на обробку введених даних, бізнес-логіку або обробку мережевих запитів. Модульні тести виконуються швидко і дозволяють оперативно виявляти помилки в ізольованих

компонентах мобільного застосунку. Таким чином тестувались окремі важливі елементи застосунку, де, наприклад, здійснюються запити до API.

Widget-тестування виконується за допомогою пакету flutter_test і призначене для перевірки UI-компонентів у «мокованому» середовищі без запуску реального емулятора. За допомогою WidgetTester можна:

— будувати конкретні віджети в тестовому середовищі (pumpWidget);

— перевіряти наявність елементів у дереві віджетів (finders: find.byType, find.text);

— імітувати введення тексту та натискання на кнопки (enterText, tap);

— відстежувати зміни стану та анімації (pump з різними Duration).

Ці тести дозволяють гарантувати, що кожен екран або складний віджет поводиться відповідно до вимог і коректно реагує на взаємодію користувача.

Інтеграційне тестування виконується із застосуванням пакету integration_test (або застарілого flutter_driver) і дозволяє перевірити роботу всього застосунку або його великих фрагментів на реальному пристрої чи емуляторі. В інтеграційних тестах моделюються сценарії кінцевого користувача:

— Запуск застосунку з початкового екрану.

— Виконання послідовності взаємодій: авторизація → вибір категорії → генерація питання → оцінка.

— Перевірка відправлення даних на сервер (мокування HTTP-клієнта або перевірка логів).

— Оцінка продуктивності основних операцій (час відгуку UI, швидкість генерації).

Завдяки проведенню таких тестів забезпечується впевненість у тому, що всі шари програми (інтерфейс та бізнес-логіка) коректно інтегровані та не містять критичних дефектів. Інтеграційні тести рекомендовано запускати в CI/CD-пайплайні після кожного релізу.

CI/CD та звітність. Для автоматизації перевірки якості було налаштовано GitHub Actions (або інший CI-інструмент), який при кожному пуші:

— виконує flutter test (модульні та widget-тести);

— запускає інтеграційні тести на емуляторах Android і IOS;

— генерує звіти про покриття коду (coverage);

— попереджає команду про регресії за допомогою email або Slack.

Цей підхід дозволяє не допускати падіння якості коду й оперативно реагувати на нові дефекти.

Ручне та експлорати́вне тестування також проводилось шляхом крокової перевірки всього функціоналу: від реєстрації та авторизації до генерації запитання й його оцінки. Окремо перевірялися випадкові та граничні сценарії (відсутність інтернет-з’єднання, некоректний ввод, швидкі послідовні натискання), щоб упевнитися в стабільності взаємодії та коректній обробці помилок.

У результаті комбінування автоматизованого та ручного тестування забезпечено всебічну перевірку якості застосунку: від окремих одиничних модулів до повноцінних користувацьких сценаріїв. Це гарантує високу надійність і стабільність «Gather Together» перед випуском у повноцінний реліз.

.

ВИСНОВКИ

У цій роботі було проведено аналіз існуючих систем генерування питань для покращення людських взаємовідносин, використання сервісів штучного інтелекту таких як-от ChatGPT, Gemini та DeepSeek, для генерації питань, а також розробку мобільного засосунку використовуючи Flutter & Dart і об’єднючи це все навколо Appwrite як серверного середовища для обробки запитів та збереження даних.

У цій роботі досягнено наступних цілей:

1. Результатом проведених досліджень і розробок став мобільний застосунок, котрий за рахунок генерування питань, допоможе людям покращувати їхні взаємовідносин шляхом комунікування на тему цих питань, які націлені на бесіди на більш «глибшому рівні».

2. Створено інтуїтивно зрозумілий інтерфейс для користувача, за допомогою якого буде легко користуватись застосунком. Також завдяки тому що він є продуктивним, у користувачів не буде негативного досвіду від користування застосунком.

3. Застосунок успішно генерує питання, які можна використати для обговорень. А також завдяки методі оцінці користувачами можна відсіювати не якісні питання згенеровані штучним інтелектом.

4. Завдяки інтеграції ChatGPT, Gemini та DeepSeek вирішене питання обмеженої кількості питань у застосунку, так як їх можна генерувати нескінченно.

У підсумку, результатом роботи став зручний і ефективний мобільний застосунок, який за допомогою генерації глибоких запитань сприяє покращенню взаємин між людьми. Завдяки інтуїтивному інтерфейсу, можливості оцінювання якості запитань та інтеграції сучасних AI-моделей, застосунок забезпечує безперервний, змістовний та приємний досвід користування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Іванюш А.Р. Штучний інтелект як інструмент покращення людських взаємовідносин. Стан, досягнення і перспективи інформаційних систем і технологій: тези доп. XXV Всеукраїнська науково-технічна конференція молодих вчених, аспірантів та студентів, м. Одеса, 17–18 квітня 2025 р. Одеса: Видавництво ОНТУ, 2025. С. 204–205.
2. Комар М.П., Саченко А.О., Васильків Н.М., Гладій Г.М., Коваль В.С., Ліп'яніна-Гончаренко Х.В. Методичні рекомендації до виконання кваліфікаційної роботи з освітньо-професійної програми «Комп'ютерні науки» спеціальності 122 «Комп'ютерні науки» за першим (бакалаврським) рівнем вищої освіти. Тернопіль: ЗУНУ, 2024. 52 с.
3. Коновальчук Н. Перспективи застосування інструментів на основі штучного інтелекту у викладанні української мови як іноземної. Український педагогічний журнал. 2024. № 3. URL: https://surl.li/skcses
4. Рекомендації щодо використання ШІ в науковій комунікації. URL: https://surl.li/ciqrqo
5. AI in Dating Apps – Use Cases, Impact, Benefits, Future in 2025. URL: https://surl.li/cietfw
6. AI-ce breakers: AI-powered question generator. URL: https://surl.li/najwzt
7. Batini C., Scannapieco M. Data and Information Quality: Dimensions, Principles and Techniques. Springer, 2016. (P. 29–32)
8. Brown, H. D. Principles of Language Learning and Teaching. Pearson Longman, 2007. (P. 9–10)
9. Chapelle C. A. Computer Applications in Second Language Acquisition. Cambridge University Press, 2001. (P. 14–18)
10. Could friendship apps be the key to combatting millennial loneliness? URL: https://surl.li/yfsola

11. Ellis, R. Second Language Acquisition. Oxford University Press, 1997. (P. 14–18)

12. Halpin T., Morgan T. Information Modeling and Relational Databases. Morgan Kaufmann, 2008. (P. 29–31)

13. Harmer J. The Practice of English Language Teaching. Pearson Longman, 2007. (P. 9)

14. Harris J. Virtual Connections: A Dive into Conscious Relationship Design with AI. URL: https://surl.li/huwkie

15. Hartford A., J Stein D. The Machine Speaks: Conversational AI and the Importance of Effort to Relationships of Meaning. URL: https://surl.li/vvvahe

16. Hay D. C. Data Model Patterns: Conventions of Thought. Dorset House Publishing, 2006. (P. 30–34)

17. Ijaz H. Practical Open-Ended Questions to Build Strong Connections. URL: https://surl.li/azbjhe

18. Johnson, K. An Introduction to Foreign Language Learning and Teaching. Pearson Education, 2008. (P. 9–10)

19. Kachru B. B. The Other Tongue: English across Cultures. University of Illinois Press, 1992. (P. 29)

20. Krashen S. D. Principles and Practice in Second Language Acquisition. Pergamon Press, 1982. (P. 14–18)

21. Marlynn Wei M.D., J.D. How AI Could Shape Our Relationships and Social Interactions. URL: https://surl.li/ragvjp

22. Marlynn Wei M.D., J.D. Spending Too Much Time With AI Could Worsen Social Skills. URL: https://surl.gd/jzxtln

23. Nicole K. McNichols Ph.D. Are Artificial Intelligence Companions the Future of Love? URL: https://surl.li/vdfqua

24. Nunan D. Second Language Teaching & Learning. Heinle & Heinle Publishers, 1999. (P. 20–21)

25. Othman A. The Rise of AI Relationships: A New Frontier in Human Connection Predicting the Future of Human-AI Relationships. URL: https://surl.li/iodlkv

26. Priestley S. Empathy Erosion in the Age of Artificial Intelligence: Nurturing Genuine Human Connections in a Digitised World. URL: https://surl.li/pahggs

27. Simsion G. C. Data Modeling Essentials. Morgan Kaufmann, 2007. (P. 30–33)

28. Stallard M., Stallard K. P. Human Connection in the Age of AI. URL: https://surl.li/esimxh

29. The Transformative Power of Asking Questions in Relationships: A Psychotherapist's Perspective. URL: https://surli.cc/jqcjen

Додаток А

Лістинг основної бізнес-логіки застосунку

```
import 'package:appwrite/appwrite.dart';
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:gather_together/features/auth/login/pages/login_page.dart';
import
'package:gather_together/features/auth/register/di/register_injection.dart'
;
import
'package:gather_together/features/auth/register/presentation/cubit/register
_cubit.dart';
import
'package:gather_together/features/auth/register/presentation/pages/register
_page.dart';
import 'package:gather_together/features/question/page/question_page.dart';
import 'package:get_it/get_it.dart';

import 'core/theme.dart';
import 'core/util.dart';
import 'features/categories/pages/categories_page.dart';

void main() async {
  // Ensure Flutter is initialized before creating Appwrite client
  WidgetsFlutterBinding.ensureInitialized();

  Client client = Client()
      .setEndpoint('https://fra.cloud.appwrite.io/v1')
      .setProject('gather-together-project');

  Account account = Account(client); // Initialize the Appwrite account
instance

  GetIt.I.registerSingleton(client); // Register the instance of Appwrite
client
  GetIt.I.registerSingleton(account); // Register the instance of Appwrite
account

  // Initialize register feature dependencies
  initRegisterDependencies();

  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    final brightness =
View.of(context).platformDispatcher.platformBrightness;

    // Retrieves the default theme for the platform
```

```
    //TextTheme textTheme = Theme.of(context).textTheme;

    // Use with Google Fonts package to use downloadable fonts
    TextTheme textTheme = createTextTheme(context, "Roboto", "Roboto");

    MaterialTheme theme = MaterialTheme(textTheme);

    return MultiBlocProvider(
      providers: [BlocProvider<RegisterCubit>(create: (_) =>
GetIt.I.get<RegisterCubit>())],
      child: MaterialApp(
        title: 'Gather Together',
        theme: brightness == Brightness.light ? theme.light() :
theme.dark(),
        initialRoute: '/',
        routes: {
          '/': (context) => const MyHomePage(),
          CategoriesPage.routeName: (context) => const CategoriesPage(),
          QuestionPage.routeName: (context) => const QuestionPage(),
          LoginPage.routeName: (context) => const LoginPage(),
          RegisterPage.routeName: (context) => const RegisterPage(),
        },
      ),
    );
  }
}

class MyHomePage extends StatefulWidget {
  const MyHomePage({super.key});

  @override
  State<MyHomePage> createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {
  @override
  void initState() {
    super.initState();
    // Redirect to login page after widget is initialized
    WidgetsBinding.instance.addPostFrameCallback((_) {
      Navigator.pushReplacementNamed(context, LoginPage.routeName);
    });
  }

  @override
  Widget build(BuildContext context) {
    // This is just a placeholder as we're redirecting to login page
    return const Scaffold(body: Center(child:
CircularProgressIndicator()));
  }
}

import 'package:appwrite/appwrite.dart';
import 'package:appwrite/enums.dart';
import 'package:appwrite/models.dart';
import 'package:flutter/material.dart';
```

```
import
'package:gather_together/features/auth/register/presentation/pages/register
_page.dart';
import 'package:get_it/get_it.dart';

class LoginPage extends StatefulWidget {
  static const String routeName = '/login';

  const LoginPage({super.key});

  @override
  State<LoginPage> createState() => _LoginPageState();
}

class _LoginPageState extends State<LoginPage> {
  final TextEditingController _emailController = TextEditingController();
  final TextEditingController _passwordController =
TextEditingController();

  // Function to handle email/password login
  void _handleEmailSignIn() {
    Navigator.pushReplacementNamed(context, '/categories');
    // try {
    //   final Account account = GetIt.I.get<Account>();
    //
    //   account
    //       .createEmailPasswordSession(
    //         email: _emailController.text,
    //         password: _passwordController.text,
    //       )
    //       .then((Session result) {
    //         if (result.userId.isNotEmpty) {
    //           // Get user details after successful login
    //           return account.get();
    //         }
    //         throw AppwriteException('Login failed');
    //       })
    //       .then((User user) {
    //         debugPrint('User logged in: ${user.name}');
    //         // Navigate to the next page
    //         Navigator.pushReplacementNamed(context, '/nextPage');
    //       })
    //       .catchError((error) {
    //         if (error is AppwriteException) {
    //           debugPrint('Login error: ${error.message}');
    //           ScaffoldMessenger.of(context).showSnackBar(
    //             SnackBar(
    //               content: Text('Login failed: ${error.message}'),
    //               backgroundColor: Colors.red,
    //             ),
    //           );
    //         } else {
    //           debugPrint('Unexpected error: $error');
    //           ScaffoldMessenger.of(context).showSnackBar(
    //             SnackBar(
    //               content: Text('An unexpected error occurred. Please
```

```
try again.'),
    //                backgroundColor: Colors.red,
    //              ),
    //            );
    //          }
    //        });
    // } catch (e) {
    //   debugPrint('Error initiating login: $e');
    //   ScaffoldMessenger.of(context).showSnackBar(
    //     SnackBar(
    //       content: Text('Error initiating login. Please try again.'),
    //       backgroundColor: Colors.red,
    //     ),
    //   );
    // }
  }

  // Function to handle Google OAuth login
  void _handleGoogleSignIn() {
    try {
      Account account = GetIt.I.get<Account>();

      account
          .createOAuth2Session(provider: OAuthProvider.google)
          .then((_) {
            // After successful OAuth login, fetch user details
            return account.get();
          })
          .then((User user) {
            debugPrint('User logged in with Google: ${user.name}');
            // Navigate to the next page
            Navigator.pushReplacementNamed(context, '/categories');
          })
          .catchError((error) {
            if (error is AppwriteException) {
              debugPrint('Google login error: ${error.message}');
              ScaffoldMessenger.of(context).showSnackBar(
                SnackBar(
                  content: Text('Google login failed: ${error.message}'),
                  backgroundColor: Colors.red,
                ),
              );
            } else {
              debugPrint('Unexpected error with Google login: $error');
              ScaffoldMessenger.of(context).showSnackBar(
                SnackBar(
                  content: Text('An unexpected error occurred. Please try
again.'),
                  backgroundColor: Colors.red,
                ),
              );
            }
          });
    } catch (e) {
      debugPrint('Error initiating Google login: $e');
      ScaffoldMessenger.of(context).showSnackBar(
```

```
        SnackBar(
          content: Text('Error initiating Google login. Please try
again.'),
          backgroundColor: Colors.red,
        ),
      );
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(backgroundColor:
Theme.of(context).colorScheme.inversePrimary),
      body: Padding(
        padding: const EdgeInsets.all(36.0),
        child: Center(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.start,
            children: <Widget>[
              Text("Welcome Back", style:
Theme.of(context).textTheme.headlineLarge),
              const SizedBox(height: 16),
              Text("Sign in to continue your learning journey"),
              const SizedBox(height: 24),
              TextField(
                controller: _emailController,
                decoration: InputDecoration(labelText: 'Email', border:
OutlineInputBorder()),
                keyboardType: TextInputType.emailAddress,
              ),
              const SizedBox(height: 16),
              TextField(
                controller: _passwordController,
                decoration: InputDecoration(labelText: 'Password', border:
OutlineInputBorder()),
                obscureText: true,
              ),
              const SizedBox(height: 8),
              Align(
                alignment: Alignment.centerRight,
                child: TextButton(
                  onPressed: () {
                    // TODO: Implement forgot password logic
                  },
                  child: const Text('Forgot Password?'),
                ),
              ),
              const SizedBox(height: 14),
              SizedBox(
                width: double.infinity,
                height: 64,
                child: ElevatedButton(
                  style: ElevatedButton.styleFrom(
                    backgroundColor: Theme.of(context).colorScheme.primary,
                    // Use primary color
```

```
                    shape: RoundedRectangleBorder(
                      borderRadius: BorderRadius.circular(12), // Rounded
corners
                    ),
                    elevation: 4,
                    // Add shadow
                    textStyle:
Theme.of(context).textTheme.titleLarge?.copyWith(
                      fontWeight: FontWeight.bold,
                      letterSpacing: 1.2,
                    ),
                    minimumSize: const Size.fromHeight(48),
                  ),
                  onPressed: _handleEmailSignIn,
                  child: const Text(
                    'Sign In',
                    style: TextStyle(
                      color: Colors.white,
                      fontSize: 20,
                      fontWeight: FontWeight.w600,
                      letterSpacing: 1.1,
                    ),
                  ),
                ),
              ),
              const SizedBox(height: 24),
              Text("or continue with", style:
Theme.of(context).textTheme.bodyMedium),
              const SizedBox(height: 24),
              SizedBox(
                width: double.infinity,
                height: 48,
                child: OutlinedButton.icon(
                  icon: const Icon(Icons.login),
                  label: const Text('Continue with Google'),
                  onPressed: _handleGoogleSignIn,
                ),
              ),
              const SizedBox(height: 8),
              Row(
                mainAxisAlignment: MainAxisAlignment.center,
                children: [
                  const Text("Don't have an account?"),
                  TextButton(
                    onPressed: () {
                      Navigator.pushReplacementNamed(context,
RegisterPage.routeName);
                    },
                    child: const Text('Sign Up'),
                  ),
                ],
              ),
            ],
          ),
        ),
      ),
```

```
    );
  }
}

import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:gather_together/features/auth/login/pages/login_page.dart';
import
'package:gather_together/features/auth/register/presentation/cubit/register
_cubit.dart';
import
'package:gather_together/features/auth/register/presentation/cubit/register
_state.dart';
import
'package:gather_together/features/categories/pages/categories_page.dart';

class RegisterPage extends StatefulWidget {
  static const String routeName = '/register';

  const RegisterPage({super.key});

  @override
  State<RegisterPage> createState() => _RegisterPageState();
}

class _RegisterPageState extends State<RegisterPage> {
  final _formKey = GlobalKey<FormState>();
  final _nameController = TextEditingController();
  final _emailController = TextEditingController();
  final _passwordController = TextEditingController();
  final _confirmPasswordController = TextEditingController();

  @override
  void dispose() {
    _nameController.dispose();
    _emailController.dispose();
    _passwordController.dispose();
    _confirmPasswordController.dispose();
    super.dispose();
  }

  void _handleRegister() {
    if (_formKey.currentState!.validate()) {
      context.read<RegisterCubit>().registerUser(
        name: _nameController.text,
        email: _emailController.text,
        password: _passwordController.text,
      );
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        backgroundColor: Theme.of(context).colorScheme.inversePrimary,
```

```
        title: const Text('Create Account'),
      ),
      body: BlocConsumer<RegisterCubit, RegisterState>(
        listener: (context, state) {
          if (state is RegisterSuccess) {
            ScaffoldMessenger.of(context).showSnackBar(
              const SnackBar(
                content: Text('Account created successfully!'),
                backgroundColor: Colors.green,
              ),
            );
            Navigator.pushReplacementNamed(context,
CategoriesPage.routeName);
          } else if (state is RegisterError) {
            ScaffoldMessenger.of(context).showSnackBar(
              SnackBar(
                content: Text('Registration failed: ${state.message}'),
                backgroundColor: Colors.red,
              ),
            );
          }
        },
        builder: (context, state) {
          return SingleChildScrollView(
            child: Padding(
              padding: const EdgeInsets.all(36.0),
              child: Form(
                key: _formKey,
                child: Column(
                  mainAxisAlignment: MainAxisAlignment.start,
                  children: <Widget>[
                    Text("Join Gather Together", style:
Theme.of(context).textTheme.headlineLarge),
                    const SizedBox(height: 16),
                    Text(
                      "Create an account to start your learning journey",
                      style: Theme.of(context).textTheme.bodyMedium,
                    ),
                    const SizedBox(height: 24),

                    // Name Field
                    TextFormField(
                      controller: _nameController,
                      decoration: const InputDecoration(
                        labelText: 'Full Name',
                        border: OutlineInputBorder(),
                        prefixIcon: Icon(Icons.person),
                      ),
                      validator: (value) {
                        if (value == null || value.trim().isEmpty) {
                          return 'Please enter your name';
                        }
                        return null;
                      },
                    ),
                    const SizedBox(height: 16),
```

```
                        // Email Field
                        TextFormField(
                          controller: _emailController,
                          decoration: const InputDecoration(
                            labelText: 'Email',
                            border: OutlineInputBorder(),
                            prefixIcon: Icon(Icons.email),
                          ),
                          keyboardType: TextInputType.emailAddress,
                          validator: (value) {
                            if (value == null || value.isEmpty) {
                              return 'Please enter your email';
                            }
                            final emailRegex = RegExp(r'^[\w-\.]+@([\w-
]+\.)+[\w-]{2,4}$');
                            if (!emailRegex.hasMatch(value)) {
                              return 'Please enter a valid email';
                            }
                            return null;
                          },
                        ),
                        const SizedBox(height: 16),

                        // Password Field
                        TextFormField(
                          controller: _passwordController,
                          decoration: const InputDecoration(
                            labelText: 'Password',
                            border: OutlineInputBorder(),
                            prefixIcon: Icon(Icons.lock),
                          ),
                          obscureText: true,
                          validator: (value) {
                            if (value == null || value.isEmpty) {
                              return 'Please enter a password';
                            }
                            if (value.length < 8) {
                              return 'Password must be at least 8 characters';
                            }
                            return null;
                          },
                        ),
                        const SizedBox(height: 16),

                        // Confirm Password Field
                        TextFormField(
                          controller: _confirmPasswordController,
                          decoration: const InputDecoration(
                            labelText: 'Confirm Password',
                            border: OutlineInputBorder(),
                            prefixIcon: Icon(Icons.lock_outline),
                          ),
                          obscureText: true,
                          validator: (value) {
                            if (value == null || value.isEmpty) {
```

```
                        return 'Please confirm your password';
                      }
                      if (value != _passwordController.text) {
                        return 'Passwords do not match';
                      }
                      return null;
                    },
                  ),
                  const SizedBox(height: 24),

                  // Register Button
                  SizedBox(
                    width: double.infinity,
                    height: 64,
                    child: ElevatedButton(
                      style: ElevatedButton.styleFrom(
                        backgroundColor:
Theme.of(context).colorScheme.primary,
                        shape: RoundedRectangleBorder(borderRadius:
BorderRadius.circular(12)),
                        elevation: 4,
                        textStyle:
Theme.of(context).textTheme.titleLarge?.copyWith(
                          fontWeight: FontWeight.bold,
                          letterSpacing: 1.2,
                        ),
                      ),
                      onPressed: state is RegisterLoading ? null :
_handleRegister,
                      child:
                          state is RegisterLoading
                              ? const CircularProgressIndicator(color:
Colors.white)
                              : const Text(
                                'Create Account',
                                style: TextStyle(
                                  color: Colors.white,
                                  fontSize: 20,
                                  fontWeight: FontWeight.w600,
                                  letterSpacing: 1.1,
                                ),
                              ),
                    ),
                  ),
                  const SizedBox(height: 24),

                  // Already have an account
                  Row(
                    mainAxisAlignment: MainAxisAlignment.center,
                    children: [
                      const Text("Already have an account?"),
                      TextButton(
                        onPressed: () {
                          Navigator.pushReplacementNamed(context,
LoginPage.routeName);
                        },
```

```
                          child: const Text('Sign In'),
                        ),
                      ],
                    ),
                  ],
                ),
              ),
            ),
          );
        },
      ),
    );
  }
}

import 'package:flutter_bloc/flutter_bloc.dart';
import
'package:gather_together/features/auth/register/domain/usecases/register_us
ecase.dart';
import
'package:gather_together/features/auth/register/presentation/cubit/register
_state.dart';

class RegisterCubit extends Cubit<RegisterState> {
  final RegisterUseCase registerUseCase;

  RegisterCubit({required this.registerUseCase}) : super(const
RegisterInitial());

  Future<void> registerUser({
    required String name,
    required String email,
    required String password,
  }) async {
    emit(const RegisterLoading());

    try {
      final user = await registerUseCase(name: name, email: email,
password: password);

      emit(RegisterSuccess(user));
    } catch (e) {
      emit(RegisterError(e.toString()));
    }
  }
}

import 'package:equatable/equatable.dart';
import
'package:gather_together/features/auth/register/domain/entities/user_entity
.dart';

abstract class RegisterState extends Equatable {
  const RegisterState();

  @override
```

```
  List<Object?> get props => [];
}

class RegisterInitial extends RegisterState {
  const RegisterInitial();
}

class RegisterLoading extends RegisterState {
  const RegisterLoading();
}

class RegisterSuccess extends RegisterState {
  final UserEntity user;

  const RegisterSuccess(this.user);

  @override
  List<Object?> get props => [user];
}

class RegisterError extends RegisterState {
  final String message;

  const RegisterError(this.message);

  @override
  List<Object?> get props => [message];
}

import 'package:equatable/equatable.dart';

class UserEntity extends Equatable {
  final String id;
  final String name;
  final String email;

  const UserEntity({required this.id, required this.name, required
this.email});

  @override
  List<Object?> get props => [id, name, email];
}

import
'package:gather_together/features/auth/register/domain/entities/user_entity
.dart';

abstract class RegisterRepository {
  /// Registers a new user with email and password
  ///
  /// Returns the created [UserEntity] if successful
  /// Throws an exception if the registration fails
  Future<UserEntity> registerUser({
    required String name,
    required String email,
    required String password,
```

```
  });
}

import
'package:gather_together/features/auth/register/domain/entities/user_entity
.dart';
import
'package:gather_together/features/auth/register/domain/repositories/registe
r_repository.dart';

class RegisterUseCase {
  final RegisterRepository repository;

  RegisterUseCase(this.repository);

  /// Executes the register use case
  Future<UserEntity> call({
    required String name,
    required String email,
    required String password,
  }) async {
    return await repository.registerUser(name: name, email: email,
password: password);
  }
}

import 'package:appwrite/appwrite.dart';
import
'package:gather_together/features/auth/register/data/datasources/register_d
ata_source.dart';
import
'package:gather_together/features/auth/register/data/repositories/register_
repository_impl.dart';
import
'package:gather_together/features/auth/register/domain/repositories/registe
r_repository.dart';
import
'package:gather_together/features/auth/register/domain/usecases/register_us
ecase.dart';
import
'package:gather_together/features/auth/register/presentation/cubit/register
_cubit.dart';
import 'package:get_it/get_it.dart';

final sl = GetIt.instance;

Future<void> initRegisterDependencies() async {
  // Cubits
  sl.registerFactory(() => RegisterCubit(registerUseCase: sl()));

  // Use cases
  sl.registerLazySingleton(() => RegisterUseCase(sl()));

  // Repositories
  sl.registerLazySingleton<RegisterRepository>(() =>
RegisterRepositoryImpl(sl()));
```

```
  // Data sources
  sl.registerLazySingleton<RegisterDataSource>(() =>
RegisterDataSourceImpl(sl<Account>()));
}

import 'package:appwrite/appwrite.dart';
import 'package:appwrite/models.dart';

abstract class RegisterDataSource {
  Future<User> registerUser({
    required String name,
    required String email,
    required String password,
  });
}

class RegisterDataSourceImpl implements RegisterDataSource {
  final Account _account;

  RegisterDataSourceImpl(this._account);

  @override
  Future<User> registerUser({
    required String name,
    required String email,
    required String password,
  }) async {
    try {
      final user = await _account.create(
        userId: ID.unique(),
        name: name,
        email: email,
        password: password,
      );

      // Create session for the user after registration
      await _account.createEmailPasswordSession(email: email, password:
password);

      return user;
    } catch (e) {
      throw Exception('Failed to register user: $e');
    }
  }
}

import 'package:appwrite/models.dart';
import
'package:gather_together/features/auth/register/data/datasources/register_d
ata_source.dart';
import
'package:gather_together/features/auth/register/domain/entities/user_entity
.dart';
import
'package:gather_together/features/auth/register/domain/repositories/registe
```

```
r_repository.dart';

class RegisterRepositoryImpl implements RegisterRepository {
  final RegisterDataSource dataSource;

  RegisterRepositoryImpl(this.dataSource);

  @override
  Future<UserEntity> registerUser({
    required String name,
    required String email,
    required String password,
  }) async {
    try {
      final User user = await dataSource.registerUser(name: name, email:
email, password: password);

      return UserEntity(id: user.$id, name: user.name, email: user.email);
    } catch (e) {
      throw Exception('Repository error: $e');
    }
  }
}

import 'package:flutter/material.dart';
import 'package:gather_together/features/question/page/question_page.dart';

class CategoriesPage extends StatelessWidget {
  const CategoriesPage({super.key});

  static final routeName = '/categories';

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Column(
          children: [
            Expanded(
              child: GridView.count(
                crossAxisCount: 2,
                crossAxisSpacing: 16,
                mainAxisSpacing: 16,
                children: [
                  _CategoryCard(
                    icon: Icons.book,
                    name: 'Books',
                    description: 'Explore a variety of books.',
                  ),
                  _CategoryCard(
                    icon: Icons.music_note,
                    name: 'Music',
                    description: 'Listen to your favorite tunes.',
                  ),
                  _CategoryCard(
```

```
                    icon: Icons.sports_soccer,
                    name: 'Sports',
                    description: 'All about sports activities.',
                  ),
                  _CategoryCard(
                    icon: Icons.movie,
                    name: 'Movies',
                    description: 'Watch the latest movies.',
                  ),
                  // Add more categories as needed
                ],
              ),
            ),
            const SizedBox(height: 16),
            SizedBox(
              width: double.infinity,
              height: 64,
              child: ElevatedButton(
                style: ElevatedButton.styleFrom(
                  backgroundColor:
Theme.of(context).colorScheme.primaryContainer,
                  shape: RoundedRectangleBorder(borderRadius:
BorderRadius.circular(12)),
                  elevation: 4,
                  textStyle: Theme.of(
                    context,
                  ).textTheme.titleLarge?.copyWith(fontWeight:
FontWeight.bold, letterSpacing: 1.2),
                ),
                onPressed: () {
                  // TODO: Implement question generation logic
                  Navigator.pushNamed(context, QuestionPage.routeName);
                  ScaffoldMessenger.of(
                    context,
                  ).showSnackBar(const SnackBar(content: Text('Generating
question...')));
                },
                child: Text(
                  'Generate Question',
                  style: TextStyle(
                    color:
Theme.of(context).colorScheme.onPrimaryContainer,
                    fontSize: 20,
                    fontWeight: FontWeight.w600,
                    letterSpacing: 1.1,
                  ),
                ),
              ),
            ),
          ],
        ),
      ),
    );
  }
}
```

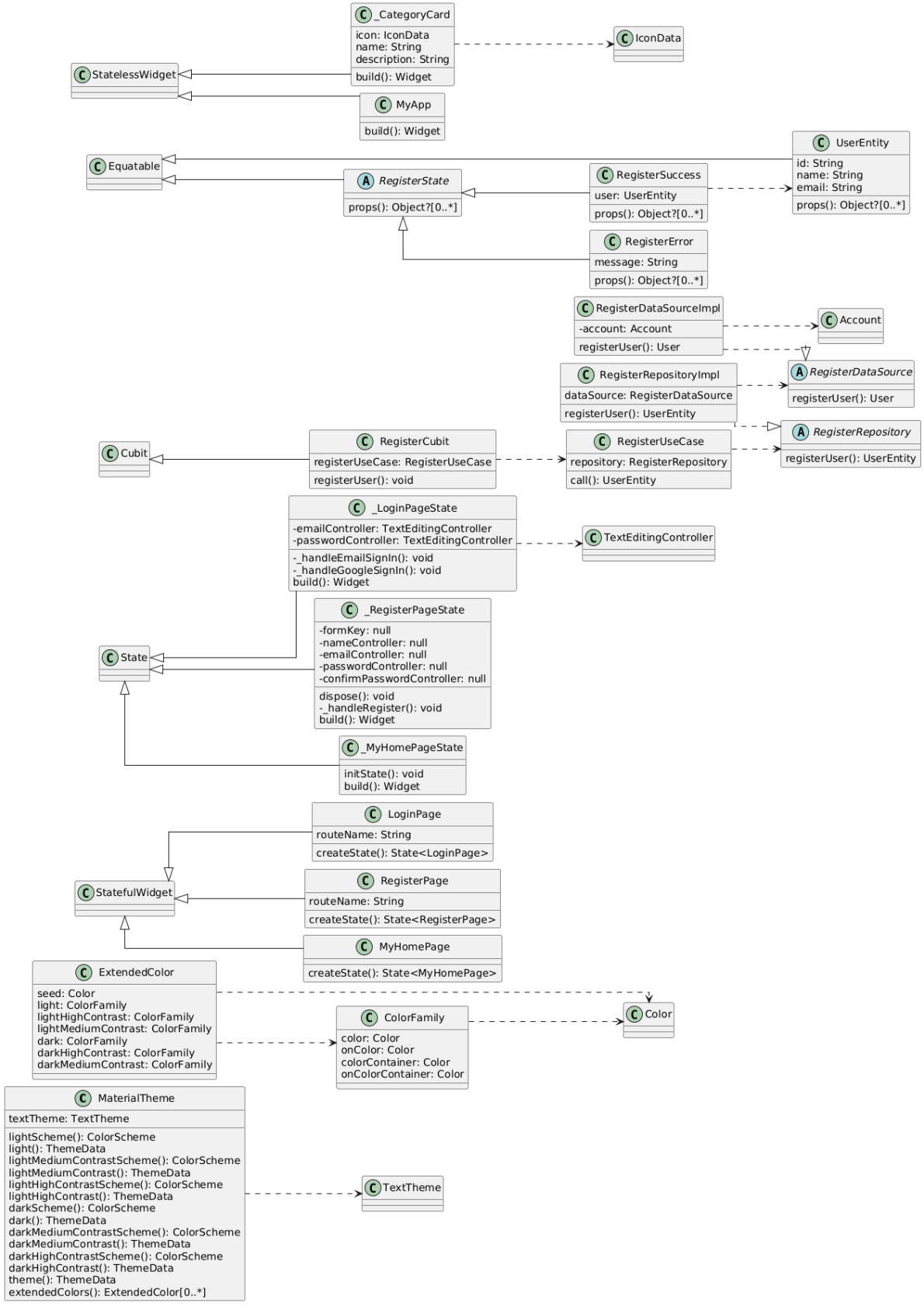
```
class _CategoryCard extends StatelessWidget {
  final IconData icon;
  final String name;
  final String description;

  const _CategoryCard({required this.icon, required this.name, required
this.description});

  @override
  Widget build(BuildContext context) {
    return Card(
      elevation: 3,
      shape: RoundedRectangleBorder(borderRadius:
BorderRadius.circular(12)),
      child: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Icon(icon, size: 40, color:
Theme.of(context).colorScheme.primary),
            const SizedBox(height: 12),
            Text(
              name,
              style:
Theme.of(context).textTheme.titleMedium?.copyWith(fontWeight:
FontWeight.bold),
              textAlign: TextAlign.center,
            ),
            const SizedBox(height: 8),
            Text(
              description,
              style: Theme.of(context).textTheme.bodySmall,
              textAlign: TextAlign.center,
            ),
          ],
        ),
      ),
    );
  }
}

import 'package:flutter/material.dart';

class QuestionPage extends StatelessWidget {
  const QuestionPage({super.key});

  static final routeName = '/question';

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        child: Card(
          elevation: 4,
          shape: RoundedRectangleBorder(borderRadius:
```

```
BorderRadius.circular(16)),
          margin: const EdgeInsets.all(24),
          child: Padding(
            padding: const EdgeInsets.all(20.0),
            child: Column(
              mainAxisSize: MainAxisSize.min,
              crossAxisAlignment: CrossAxisAlignment.start,
              children: [
                // Question Title
                Text(
                  'What are some common communication mistakes you see
people make, and how can they be avoided?',
                  style: Theme.of(
                    context,
                  ).textTheme.headlineSmall?.copyWith(fontWeight:
FontWeight.bold),
                ),
                const SizedBox(height: 8),
                // Category Tag
                Container(
                  padding: const EdgeInsets.symmetric(horizontal: 10,
vertical: 4),
                  decoration: BoxDecoration(
                    color:
Theme.of(context).colorScheme.primary.withOpacity(0.1),
                    borderRadius: BorderRadius.circular(8),
                  ),
                  child: Text(
                    'Communication',
                    style:
Theme.of(context).textTheme.labelMedium?.copyWith(
                      color: Theme.of(context).colorScheme.primary,
                      fontWeight: FontWeight.w600,
                    ),
                  ),
                ),
                const SizedBox(height: 12),
                // Views and Likes/Dislikes Bar
                Row(
                  children: [
                    Icon(Icons.remove_red_eye, size: 18, color:
Colors.grey[600]),
                    const SizedBox(width: 4),
                    Text('123 views', style:
Theme.of(context).textTheme.bodySmall),
                    const Spacer(),
                    // Likes/Dislikes Bar
                    SizedBox(
                      width: 100,
                      height: 10,
                      child: Row(
                        children: [
                          Expanded(
                            flex: 7, // likes
                            child: Container(
                              decoration: BoxDecoration(
```

```
                                color: Colors.green,
                                borderRadius: const
BorderRadius.horizontal(
                                  left: Radius.circular(5),
                                ),
                              ),
                            ),
                          ),
                          Expanded(
                            flex: 3, // dislikes
                            child: Container(
                              decoration: BoxDecoration(
                                color: Colors.red,
                                borderRadius: const
BorderRadius.horizontal(
                                  right: Radius.circular(5),
                                ),
                              ),
                            ),
                          ),
                        ),     ],     ),    ),     ],     ),
                const SizedBox(height: 16),
                // Like, Dislike, Report Buttons
                Row(
                  children: [
                    IconButton(
                  icon: const Icon(Icons.thumb_up, color: Colors.green),
                      onPressed: () {},
                    ),
              Text('70', style: Theme.of(context).textTheme.bodyMedium),
                    const SizedBox(width: 8),
                    IconButton(
                      icon: const Icon(Icons.thumb_down, color:
Colors.red),
                      onPressed: () {},              ),
                    Text('30', style:
Theme.of(context).textTheme.bodyMedium),
                    const SizedBox(width: 16),
                    TextButton.icon(
                      onPressed: () {},
                      icon: const Icon(Icons.flag, size: 18, color:
Colors.grey),
                      label: const Text('Report'),
                      style: TextButton.styleFrom(foregroundColor:
Colors.grey[700]),
                    ),
                  ],
                ),
              ],
            ),
          ),
        ),
      ),
    );
  }
}
```

Додаток Б

UML-діаграма класів



Додаток В

Код генерування питання за допомогою ШІ

```
import json
import os

from appwrite.id import ID
from appwrite.services.databases import Databases
from openai import OpenAI
from appwrite.client import Client

from .utils import throw_if_missing


def main(context):
    throw_if_missing(os.environ, ["CHATGPT_API_KEY"])
    throw_if_missing(os.environ, ["APPWRITE_FUNCTION_PROJECT_ID"])
    throw_if_missing(os.environ, ["APPWRITE_DATABASE_ID"])
    throw_if_missing(os.environ, ["APPWRITE_COLLECTION_ID"])

    # Set project and set API key
    client = (
        Client()
           .set_project(os.environ["APPWRITE_FUNCTION_PROJECT_ID"])
           .set_key(context.req.headers["x-appwrite-key"])
    )

    databases = Databases(client)

    try:
        throw_if_missing(context.req.body, ["prompt"])
    except ValueError as err:
        context.error(f"Missing 'prompt' in request body: {err}")
        return context.res.json({"ok": False, "error": err.message}, 400)

    client = OpenAI(api_key=os.environ["CHATGPT_API_KEY"])

    try:
        response = client.chat.completions.create(
            model="gpt-4.1-nano",
            messages=[
                {"role": "system", "content": "Generate a list of questions
on the topic of topic which user said that are designed to encourage people
to discuss and get to know each other better. The questions should be open-
ended and promote meaningful conversation. Return as json response as
example {\"questions\":[\"question1\",\"question2\",\"etc\"]}."},
                {"role": "user", "content": context.req.body["prompt"]},
            ],
            max_tokens=512,
            temperature=1.5,
            stream=False
        )

        # Extract the content from the response object
        completion = response.choices[0].message.content
```

```
        # Clear from ```json and ```
        completion = completion.replace("```json", "").replace("```", "")

        # Convert to json
        completion = json.loads(completion)

        # Log the extracted content
        context.log("Completion:", completion)

        # Check if completion has questions as array if true save to
database
        if "questions" in completion:
            # Save to database every element
            for question in completion["questions"]:
                try:
                    databases.create_document(
                        database_id=os.environ["APPWRITE_DATABASE_ID"],
                        collection_id=os.environ["APPWRITE_COLLECTION_ID"],
                        document_id=ID.unique(),
                        data={
                            "question": question,
                            "generated_by": "ChatGPT",
                            "topic": context.req.body["prompt"],
                        }
                    )
                except Exception as e:
                    context.error("Failed to create document: " + e)
                    return context.response.text("Failed to create
document")



        return context.res.json({"ok": True, "completion": completion},
200)

    except Exception as err:
        context.error(f"Failed to complete: {err}")
        return context.res.json({"ok": False, "error": "Failed to query
model."}, 500)
import json
import os

from appwrite.id import ID
from appwrite.services.databases import Databases
from openai import OpenAI
from appwrite.client import Client

from .utils import throw_if_missing


def main(context):
    throw_if_missing(os.environ, ["DEEPSEEK_API_KEY"])
    throw_if_missing(os.environ, ["APPWRITE_FUNCTION_PROJECT_ID"])
    throw_if_missing(os.environ, ["APPWRITE_DATABASE_ID"])
    throw_if_missing(os.environ, ["APPWRITE_COLLECTION_ID"])
```

```
    # Set project and set API key
    client = (
        Client()
           .set_project(os.environ["APPWRITE_FUNCTION_PROJECT_ID"])
           .set_key(context.req.headers["x-appwrite-key"])
    )

    databases = Databases(client)

    try:
        throw_if_missing(context.req.body, ["prompt"])
    except ValueError as err:
        context.error(f"Missing 'prompt' in request body: {err}")
        return context.res.json({"ok": False, "error": err.message}, 400)

    client = OpenAI(api_key=os.environ["DEEPSEEK_API_KEY"],
base_url="https://api.deepseek.com")

    try:
        response = client.chat.completions.create(
            model="deepseek-chat",
            messages=[
                {"role": "system", "content": "Generate a list of questions
on the topic of topic which user said that are designed to encourage people
to discuss and get to know each other better. The questions should be open-
ended and promote meaningful conversation. Return as json response as
example {\"questions\":[\"question1\",\"question2\",\"etc\"]}."},
                {"role": "user", "content": context.req.body["prompt"]},
            ],
            max_tokens=512,
            temperature=1.5,
            stream=False
        )

        # Extract the content from the response object
        completion = response.choices[0].message.content

        # Clear from ```json and ```
        completion = completion.replace("```json", "").replace("```", "")

        # Convert to json
        completion = json.loads(completion)

        # Log the extracted content
        context.log("Completion:", completion)

        # Check if completion has questions as array if true save to
database
        if "questions" in completion:
            # Save to database every element
            for question in completion["questions"]:
                try:
                    databases.create_document(
                        database_id=os.environ["APPWRITE_DATABASE_ID"],
                        collection_id=os.environ["APPWRITE_COLLECTION_ID"],
```

```
                        document_id=ID.unique(),
                        data={
                            "question": question,
                            "generated_by": "DeepSeek",
                            "topic": context.req.body["prompt"],
                        }
                    )
                except Exception as e:
                    context.error("Failed to create document: " + e)
                    return context.response.text("Failed to create
document")



        return context.res.json({"ok": True, "completion": completion},
200)

    except Exception as err:
        context.error(f"Failed to complete: {err}")
        return context.res.json({"ok": False, "error": "Failed to query
model."}, 500)

import json
import os

from appwrite.id import ID
from appwrite.services.databases import Databases
from openai import OpenAI
from appwrite.client import Client

from .utils import throw_if_missing


def main(context):
    throw_if_missing(os.environ, ["GEMINI_API_KEY"])
    throw_if_missing(os.environ, ["APPWRITE_FUNCTION_PROJECT_ID"])
    throw_if_missing(os.environ, ["APPWRITE_DATABASE_ID"])
    throw_if_missing(os.environ, ["APPWRITE_COLLECTION_ID"])

    # Set project and set API key
    client = (
        Client()
           .set_project(os.environ["APPWRITE_FUNCTION_PROJECT_ID"])
           .set_key(context.req.headers["x-appwrite-key"])
    )

    databases = Databases(client)

    try:
        throw_if_missing(context.req.body, ["prompt"])
    except ValueError as err:
        context.error(f"Missing 'prompt' in request body: {err}")
        return context.res.json({"ok": False, "error": err.message}, 400)

    client = OpenAI(api_key=os.environ["GEMINI_API_KEY"],
base_url="https://generativelanguage.googleapis.com/v1beta/openai/")
```

```
    try:
        response = client.chat.completions.create(
            model="gemini-2.0-flash",   # "gemini-2.5-pro-exp-03-25",
            n=1,
            messages=[
                {"role": "system", "content": "Generate a list of questions
on the topic of topic which user said that are designed to encourage people
to discuss and get to know each other better. The questions should be open-
ended and promote meaningful conversation. Return as json response as
example {\"questions\":[\"question1\",\"question2\",\"etc\"]}."},
                {"role": "user", "content": context.req.body["prompt"]},
            ],
            max_tokens=512,
            temperature=1,
            stream=False
        )

        # Extract the content from the response object
        completion = response.choices[0].message.content

        # Clear from ```json and ```
        completion = completion.replace("```json", "").replace("```", "")

        # Convert to json
        completion = json.loads(completion)

        # Log the extracted content
        context.log("Completion:", completion)

        # Check if completion has questions as array if true save to
database
        if "questions" in completion:
            # Save to database every element
            for question in completion["questions"]:
                try:
                    databases.create_document(
                        database_id=os.environ["APPWRITE_DATABASE_ID"],
                        collection_id=os.environ["APPWRITE_COLLECTION_ID"],
                        document_id=ID.unique(),
                        data={
                            "question": question,
                            "generated_by": "Gemini",
                            "topic": context.req.body["prompt"],
                        }
                    )
                except Exception as e:
                    context.error("Failed to create document: " + e)
                    return context.response.text("Failed to create
document")



        return context.res.json({"ok": True, "completion": completion},
200)
```

```
    except Exception as err:
        context.error(f"Failed to complete: {err}")
        return context.res.json({"ok": False, "error": "Failed to query
model."}, 500)
import os

__dirname = os.path.dirname(os.path.abspath(__file__))
static_folder = os.path.join(__dirname, "../static")


def get_static_file(file_name: str) -> str:
    """
    Returns the contents of a file in the static folder

    Parameters:
        file_name (str): Name of the file to read

    Returns:
        (str): Contents of static/{file_name}
    """
    file_path = os.path.join(static_folder, file_name)
    with open(file_path, "r") as file:
        return file.read()


def throw_if_missing(obj: object, keys: list[str]) -> None:
    """
    Throws an error if any of the keys are missing from the object

    Parameters:
        obj (object): Object to check
        keys (list[str]): List of keys to check

    Raises:
        ValueError: If any keys are missing
    """
    missing = [key for key in keys if key not in obj or not obj[key]]
    if missing:
        raise ValueError(f"Missing required fields: {', '.join(missing)}")
```

Додаток Г

Копія опублікованих результатів

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Одеський національний технологічний університет
ННІ Комп`ютерної інженерії, автоматизації робототехніки та програмування ім. П.М.Платонова
Національний технічний університет України
«Київський політехнічний інститут»
Університет Інформатики і прикладних знань м. Лодзь, Польща

XXV Всеукраїнська науково-технічна конференція молодих вчених, аспірантів та студентів

«СТАН, ДОСЯГНЕННЯ І ПЕРСПЕКТИВИ ІНФОРМАЦІЙНИХ СИСТЕМ І ТЕХНОЛОГІЙ»

Матеріали конференції

Одеса

17-18 квітня 2025 р.

УДК 004.8:159.9

ШТУЧНИЙ ІНТЕЛЕКТ ЯК ІНСТРУМЕНТ ПОКРАЩЕННЯ ЛЮДСЬКИХ ВЗАЄМОВІДНОСИН

ІВАНЮШ А. Р.
(adam.ivaniush@gmail.com)
Західноукраїнський національний університет

У сучасному цифровому світі зростає потреба у створенні рішень, що сприяють зміцненню людських зв'язків. Аби покращити людські взаємовідносини за допомогою штучного інтелекту будуть генеруватись питання на різні теми, які допоможуть користувачам обговорити ці питання, тим самим пізнаючи одне одного.

На сьогоднішній день, життя дуже тісно переплітається з цифровими технологіями і штучний інтелект активно впроваджується в різноманітні сфери діяльності. Стрес, ізоляція, цифрове перевантаження та недостатня емоційна взаємодія часто стають причинами погіршення міжособистісного спілкування. Штучний інтелект, який активно впроваджується у повсякденне життя, має потенціал не лише автоматизувати процеси, а й виступати як інструмент підтримки та розвитку емоційного інтелекту, комунікації та гармонізації стосунків між людьми. Проблемою є недостатнє вивчення та усвідомлення потенціалу ШІ як засобу покращення якості міжособистісних взаємин, а також відсутність простих інструментів з якими люди зможуть взаємодіяти, аби покращити міжособистісні відносини.

Було досліджено, які з сучасних штучних інтелектів, які мають API, краще підходять для генерування питань та покращення генерації коректний питань. А саме для генерувань питань було використано моделі ChatGPT, Gemini та DeepSeek. Також для коректної генерації було спроєктовані різні промпти та підібрані кращі описані вимоги та потреби для роботи з моделями штучного інтелекту. Також для отримання кращих питань було дослідженно кращі психологічні практики які використовувались для створення промптів.

Так як люди все частіше у сучасному світі спілкуються через мережу іноді це зводиться до простих діалогів або питань «Про що б поговорити?» або задаються питанням, що хотіли б краще когось пізнати, але не можуть придумати, якісь цікаві та унікальні питання, які відкриють особу з іншої сторони. Генерація питань на різні теми, зможе бути як корисним інструментом для кращого пізнання одне одного, а також для компаній осіб, які хочуть перезнайомитись краще та з певним інтерактивом. Для цього було використано API популярних моделей штучного інтелекту, таких як-от ChatGPT, Gemini, DeepSeek. Наступним кроком, необхідно було розробити коректний промпт, який б надавав інструкції, щоб відповідь була на певну тему, поверталась у певному форматі даних (до прикладу JSON), а також аби питання не були примітивними чи не коректними за своїм сенсом. Для цього було проаналізовані певну психологічну літературу та досліджено кращі психологічні практики. Ці знання були використанні у побудові промптів з якими будуть працювати моделі.

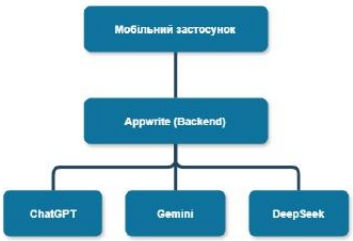


Рисунок 1 – Архітектура розробленого рішення

На рис. 1 відображено архітектура роботи. Де є мобільний застосунок, який відповідає за відображення та взаємодію з користувачем. Appwrite – backend частина, де зберігається згенеровані питання, а також відбувається взаємодія з API ChatGPT, Gemini та DeepSeek для генерування питань. Промпти за допомогою яких генеруються питань знаходяться також в базі даних, що дозволяє покращувати їх або доадвати нові.

Навчаючи достатніми коректними даними моделі штучного інтелекту, можна отримати гарні запитання, які можна буде використовувати для покращення відносин з друзями, колегами чи просто заводити нові знайомства. ChatGPT та Gemini краще підходять для даної цілі, так як надають можливість навчати їхні моделі своїми даними. DeepSeek програє у функціональності, а також у тому, що він більше напрямлений на технічні питання та вирішення задач ніж на творчі такі як-от генерації питань. Також завдяки простому інтерфейсі з якого можна обрати тему користувачам буде легко отримувати дані питання, які уже мають хороші промпти у своїй функціональності.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Н. Коновальчук. “Перспективи застосування інструментів на основі штучного інтелекту у викладанні української мови як іноземної.” *Український педагогічний журнал*. № 3. С. 160-166 2024. [Інтернет ресурс] Доступний: https://surl.li/skcses Дата звернення: 02.03.2025

2. Psychology Today (2025, March 7). *How AI Could Shape Our Relationships and Social Interactions* [Online]. Avaible: https://surl.li/ragvjp

3. Hartford A., J Stein D (2024, Jun. 18). “The Machine Speaks: Conversational AI and the Importance of Effort to Relationships of Meaning.” *JMIR Ment Health* [Online]. Vol. 11 Avaible: https://surl.li/vvvahe

4. Medium (2023, Oct. 31). *Virtual Connections: A Dive into Conscious Relationship Design with AI* [Online]. Avaible: https://surl.li/huwkie

Psychology Today (2024, Nov. 19). *Are Artificial Intelligence Companions the Future of Love?* [Online]. Avaible: https://surl.li/vdfqua