Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Штучного інтелекту
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження ефективності нейронних мереж генерування тексту для мобільної текстової quest-гри
(тема)

Виконав:
студент 2 курсу, групи СШІм-21-1
Пономаренко Д.Р.
(прізвище, ініціали)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системи штучного інтелекту
(повна назва спеціалізації)

Керівник проф. Смеляков К.С.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ В.О. Філатов
(підпис) (прізвище, ініціали)

2023 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Штучного інтелекту
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системи штучного інтелекту (СШІ)
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_____» _____ 20 ___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Пономаренку Денису Романовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження ефективності нейронних мереж генерування тексту для мобільної текстової quest-гри

затверджена наказом університету від 31 березня 20 23 р. № 306Ст

2. Термін подання студентом роботи до екзаменаційної комісії 19 травня 20 23 р.

3. Вихідні дані до роботи мобільний додаток, нейронна мережа. ОС Android, середовище Android Studio, GPT-2, GPT-3, Flutter, Dart, Tensorflow lite, Дослідження мовних моделей, Аналітика, Firebase.

4. Перелік питань, що потрібно опрацювати в роботі

1) Аналіз проблемної галузі

2) Постановка задачі

3) Аналіз ефективності існуючих та розроблених алгоритмів для вирішення задачі

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри)_____

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
	Аналіз предметної галузі	03.04.2023	Виконано
	Огляд існуючих методів	20.04.2023	Виконано
	Аналіз ефективності методів	27.04.2023	Виконано
	Підготовка пояснювальної записки	01.05.2023	Виконано
	Підготовка презентації та доповіді	14.05.2023	Виконано
	Попередній захист	16.05.2023	Виконано
	Захист кваліфікаційної роботи	19.05.2023	

Дата видачі завдання 3 квітня 2023 р.

Студент _____
(підпис)

Керівник роботи _____ проф. Смеляков К.С.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 77 с., 23 рис., 2 дод., 12 джерел.

AI, ANDROID, ANDROID STUDIO, AVD, CLIENT, DART, FLUTTER, GRADLE, GPT-2, GPT-3, KOTLIN, MODEL, SDK, TENSORFLOW, TENSORFLOW-LITE.

Об'єктом дослідження є роботи з деякої мовною моделлю gpt-3 від OpenAI, та використовуючи його побудувати мобільний додаток генерації квестів, історій.

Метою роботи є дослідження ефективності мовних моделей генерації тексту, та створення сучасної та конкурентоспроможної додатка-генератора на мобільний пристрій яка повинна мати змогу зацікавити користувача, використовуючи програмні засоби Android/IOS розробки, у нашому випадку мови Flutter, та роботи з засобами деякого великої мовної моделі gpt-3.

Методами досліджень є аналіз технологій розробки на Android/Ios та дослідження ефективності мовних моделей генерування тексту, що використовують технології штучного інтелекту та нейронних мереж, вивчення можливостей роботи з мовними моделями. Також аналіз інших джерел таких як література та електронні ресурси.

Здійснено практичне створення програмного додатка за допомогою мови flutter із застосуванням можливостей роботи з мовною моделью gpt-3. Результатом атестаційної роботи є додаток для мобільних пристроїв, деякий індивідуальний квест генератор.

ABSTRACT

Explanatory note: 77 p., 23 fig., 2 ann.,12 sources.

AI, ANDROID, ANDROID STUDIO, AVD, CLIENT, DART, FLUTTER, GRADLE, GPT-2, GPT-3, KOTLIN, MODEL, SDK, TENSORFLOW, TENSORFLOW-LITE.

The object of research is to work with some language model gpt-2 from OpenAI, and using it to build a mobile application for generating quests / stories.

The aim of the work is to create a modern and competitive application-generator for a mobile device that should be able to interest the user using Android development tools, in our case Flutter, and work with some large language model gpt-3.

Research methods are the analysis of development technologies on Android/IOS, using technologies of artificial intelligence and neural networks in app, the study of the possibilities of working with language models. Also analysis of other sources such as literature and electronic resources.

The practical creation of a software application using the Kotlin language using the capabilities of working with the gpt-3 language model. The result of the certification work is an application for mobile devices, some individual quest generator.

ЗМІСТ

ВСТУП

Штучний інтелект (ШІ) швидко став невід'ємною частиною нашого повсякденного життя. Штучний інтелект вже впливає на те, як ми живемо, працюємо та взаємодіємо з технологіями: від віртуальних помічників, таких як Siri та Alexa, до систем рекомендацій, таких як Netflix і Amazon [12].

У наш час штучний інтелект використовується в різних галузях, таких як охорона здоров'я, фінанси, транспорт і виробництво. Він використовується для підвищення точності медичних діагнозів, автоматизації фінансових операцій, підвищення безпеки транспортування та оптимізації виробничих процесів. ШІ також використовується для соціального блага, зокрема для реагування на стихійні лиха та збереження дикої природи.

Дивлячись у майбутнє, очікується, що ШІ продовжуватиме розвиватися та змінювати наш світ. Передбачається, що штучний інтелект відіграватиме дедалі важливішу роль у таких сферах, як освіта, розваги та сільське господарство. Штучний інтелект також продовжить революціонізувати спосіб нашої роботи, маючи потенціал для автоматизації багатьох завдань, які зараз виконують люди.

Однак із цими досягненнями виникає потреба в тому, щоб штучний інтелект розроблявся та використовувався етично та відповідально. Занепокоєння щодо переміщення робочих місць, конфіденційності даних і упередженості в системах штучного інтелекту потрібно буде вирішити в міру розвитку технології.

Загалом, майбутнє штучного інтелекту захоплююче та повне можливостей, але воно потребуватиме ретельного управління, щоб гарантувати, що він принесе користь суспільству в цілому.

GPT (Generative Pre-trained Transformer) –це тип архітектури нейронної мережі, який широко використовується в задачах обробки природної мови (NLP). GPT особливо добре підходить для завдань, які

вимагають генерації природного звучання тексту, наприклад перекладу мовою, завершення тексту та відповіді на запитання [11].

У наш час GPT використовується в різних галузях і сферах застосування. Ось кілька прикладів:

– мовний переклад: моделі на основі GPT, як-от система нейронного машинного перекладу Google (NMT), використовуються для автоматичного перекладу тексту між різними мовами;

– створення вмісту. Моделі GPT можна використовувати для створення тексту для різних додатків, включаючи чат-ботів, віртуальних помічників і публікацій у соціальних мережах;

– пошук інформації: моделі GPT можна використовувати для виконання завдань із відповідями на запитання, де вони надають відповіді на запитання, поставлені природною мовою, такою як ті, що використовуються в пошукових системах;

– аналіз настрою: моделі GPT можна використовувати для аналізу настрою тексту, що корисно в таких програмах, як аналіз відгуків клієнтів і моніторинг соціальних мереж;

– творче написання: моделі GPT також можна використовувати для створення творчого тексту, наприклад віршів або оповідань.

Загалом GPT є потужним інструментом, який можна використовувати в широкому діапазоні програм, де потрібне генерування тексту природною мовою. Його універсальність і здатність навчатися на великих обсягах даних зробили його популярним вибором у сфері обробки природної мови.

Інтеграція технології GPT (Generative Pre-trained Transformer) у мобільні програми може бути хорошим рішенням з кількох причин [1].

Обробка природної мови: GPT особливо добре підходить для завдань обробки природної мови (NLP), які стають все більш важливими в мобільних додатках. Наприклад, мобільна програма, яка використовує GPT, може надавати точніші та природні відповіді на запити користувачів або створювати релевантний вміст на основі уподобань користувача.

Персоналізація: GPT можна використовувати для персоналізації взаємодії з користувачем у мобільних додатках. Аналізуючи дані користувачів, GPT може надавати персоналізовані рекомендації щодо продуктів, послуг або вмісту на основі індивідуальних уподобань і поведінки.

Автоматизація: GPT може автоматизувати багато завдань, які інакше вимагали б людського втручання, наприклад запити до служби підтримки клієнтів, чат-боти та завершення тексту. Це може зменшити робоче навантаження на людей-операторів і забезпечити більш безпроблемний досвід для користувачів.

Ефективність: GPT може швидко й ефективно виконувати складні завдання обробки природної мови, що особливо важливо в мобільних додатках, де швидкість і реакція мають вирішальне значення.

Конкурентна перевага: інтеграція GPT у мобільну програму може надати конкурентну перевагу, оскільки дає змогу використовувати більш просунуті та персоналізовані функції, яких інші програми можуть не пропонувати.

Загалом інтеграція технології GPT у мобільні додатки може надати численні переваги, зокрема покращену обробку природної мови, персоналізований досвід, автоматизацію завдань і ефективність.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз предметної області розробки мобільних додатків та нейронних мереж

Мобільна розробка стосується процесу створення мобільних програм для мобільних пристроїв, таких як смартфони та планшети. Мобільна розробка зазвичай передбачає написання програмного коду мовою програмування, такою як Java або Swift, розробку інтерфейсів користувача та тестування програми, щоб переконатися, що вона працює правильно на різних мобільних пристроях і операційних системах.

Мобільну розробку можна розділити на дві основні категорії: нативну та гібридну. Нативна розробка передбачає написання коду, специфічного для конкретної мобільної операційної системи, наприклад iOS або Android. Це дозволяє точніше контролювати функції та продуктивність програми, але вимагає окремої розробки для кожної операційної системи. З іншого боку, гібридна розробка передбачає використання єдиної кодової бази для створення програми, яку можна використовувати в кількох операційних системах. Цей підхід може бути економічнішим і ефективнішим, але може призвести до менш оптимізованої взаємодії з користувачем [2].

Мобільний розвиток стає все більш важливим, оскільки мобільні пристрої стали повсюдними в сучасному суспільстві. Мобільні програми можна використовувати для різноманітних цілей, включаючи розваги, соціальні мережі, продуктивність та електронну комерцію. Розробка мобільних пристроїв –це динамічна галузь, яка швидко розвивається, у якій постійно розробляються нові інструменти та технології для покращення можливостей і функцій мобільних програм [3].

Dart – це мова програмування, розроблена Google, яка використовується переважно для створення мобільних і веб-додатків. Dart – це об’єктно-орієнтована мова на основі класів, яка розроблена для легкого

вивчення та використання, із синтаксисом, подібним до таких мов, як Java та JavaScript. Dart часто використовується разом із фреймворком Flutter для мобільної розробки.

Flutter – це фреймворк для розробки мобільних додатків із відкритим вихідним кодом, також розроблений Google. Flutter використовує мову програмування Dart і дозволяє створювати високоякісні кросплатформні мобільні додатки як для iOS, так і для Android. Flutter надає низку віджетів та інструментів для створення красивих і чуйних інтерфейсів користувача, а також підтримку анімації, сенсорних подій і датчиків пристрою.

Одна з головних переваг використання Dart і Flutter для розробки мобільних пристроїв полягає в тому, що вони дозволяють створювати настроювані та продуктивні програми з єдиною кодовою базою. Це означає, що розробники можуть створити програму, яка бездоганно працює як на платформах iOS, так і на Android, без необхідності писати окремий код для кожної платформи. Крім того, висока продуктивність Dart і функція гарячого перезавантаження Flutter забезпечують швидкий цикл розробки та швидшу ітерацію [3].

Загалом Dart і Flutter – це потужні інструменти для мобільної розробки, які пропонують численні переваги для розробників, зокрема спрощену мову програмування, ефективний код і кросплатформну сумісність. У результаті вони стали популярним вибором для розробки мобільних додатків у різних галузях.

Нейронні мережі та глибоке навчання – це потужні методи, які використовуються в машинному навчанні для вирішення складних проблем, таких як розпізнавання зображень, обробка природної мови та ігри. Ці методи базуються на структурі та функціях людського мозку та передбачають навчання мережі взаємопов’язаних вузлів або нейронів навчатися на основі даних і робити прогнози. Глибоке навчання стосується використання нейронних мереж із кількома рівнями, що дозволяє створювати складніші та точніші моделі. Ці методи революціонізували

багато галузей, включаючи охорону здоров'я, фінанси та транспорт, і продовжують прогресувати, оскільки дослідники досліджують нові архітектури та програми.

Глибоке навчання – це підгалузь машинного навчання, яка останніми роками привернула багато уваги завдяки своїй здатності вирішувати складні проблеми, які раніше вважалися нерозв'язними за допомогою традиційних методів машинного навчання. Він заснований на штучних нейронних мережах, які розроблені для самостійного навчання та вдосконалення через процес навчання на великих обсягах даних [11].

Глибоке навчання знайшло численні застосування в таких сферах, як комп'ютерне зір, обробка природної мови, розпізнавання мови та навіть відкриття ліків. Це революціонізувати багато галузей, включаючи охорону здоров'я, фінанси та транспорт, і дозволило розробити інтелектуальні системи, які можуть виконувати завдання з рівнем точності та швидкості, який раніше вважали неможливим.

Ключ до успіху глибокого навчання полягає в його здатності вивчати ієрархічні представлення даних. Іншими словами, він може автоматично ідентифікувати складні шаблони та зв'язки в даних, вивчаючи простіші функції. Це робить його особливо придатним для таких завдань, як розпізнавання зображень і обробка природної мови, де базові дані можуть бути надзвичайно складними. Поглиблене навчання характеризується різким підвищенням точності аналізу завдяки збільшенню кількості шарів та складності. Як випливає з назви, глибока нейронна мережа (DNN) – це механізм, за допомогою якого шари нейронної мережі є багатошаровими (глибокими).

Як ви можете бачити на зображенні нижче, існує так багато шарів, об'єднаних (рисунок 1.1). Ця багатошарова структура розглядає складність інформації.

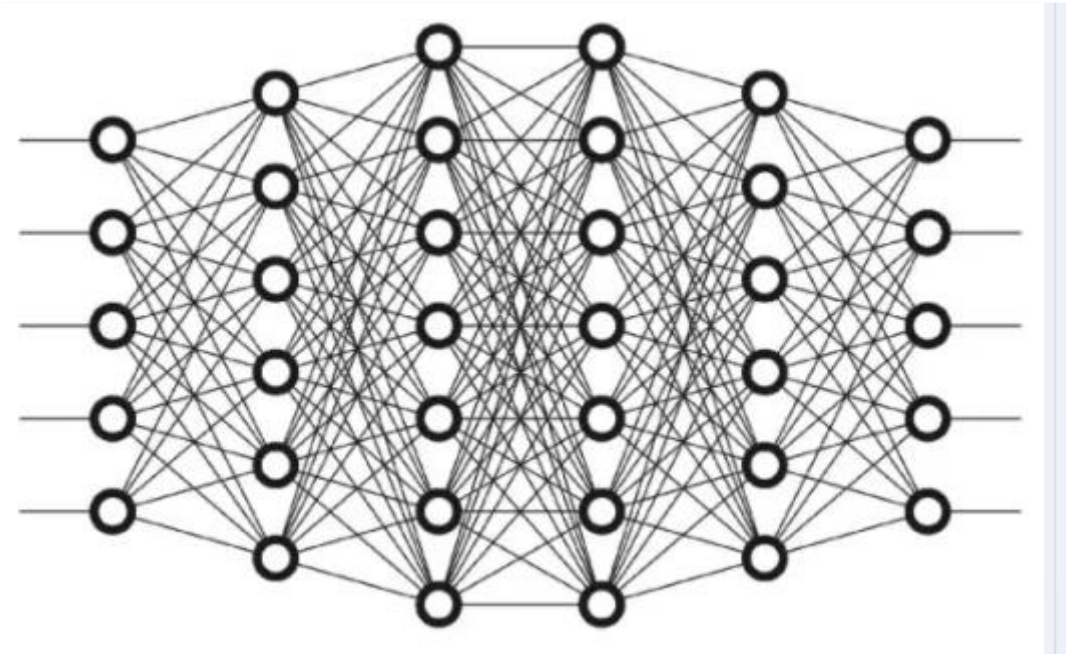

Рисунок 1.1 – Вид нейронної мережі на глибокому навчанні

Незважаючи на те, що глибоке навчання багатообіцяюче, воно все ще є відносно новою сферою, і існує багато проблем, які потрібно вирішити. До них належать проблеми, пов’язані з якістю даних, переобладнанням та можливістю інтерпретації, а також потреба в більш ефективних алгоритмах навчання та апаратному забезпеченні [6].

Незважаючи на ці проблеми, глибоке навчання готове мати значний вплив на широкий спектр галузей і застосувань, і дослідження в цій галузі тривають у міру розробки нових методів і архітектур.

Нейронні мережі знайшли численні застосування в мобільних додатках, зокрема в областях комп’ютерного зору та обробки природної мови. Кілька прикладів досліджень нейронних мереж у мобільних додатках.

Розпізнавання об’єктів: одним із поширених застосувань нейронних мереж у мобільних додатках є розпізнавання об’єктів. Дослідники розробили моделі глибокого навчання, які можуть ідентифікувати об’єкти в реальному часі, наприклад, у додатках доповненої реальності. Наприклад,

ARCore від Google використовує нейронні мережі для розпізнавання об’єктів і поверхонь у середовищі користувача.

Обробка природної мови. Нейронні мережі також використовуються в мобільних додатках для обробки природної мови, наприклад для розпізнавання мовлення та мовного перекладу. Siri від Apple і Assistant від Google використовують нейронні мережі, щоб розуміти запити користувачів і відповідати на них.

Виявлення емоцій. Виявлення емоцій – це ще одна сфера, де нейронні мережі досліджуються для мобільних додатків. Дослідники розробили моделі глибокого навчання, які можуть розпізнавати емоції за виразом обличчя або мовою, які можна використовувати в таких програмах, як моніторинг психічного здоров’я.

Мобільні ігри. Нейронні мережі також використовуються в мобільних іграх, зокрема для анімації та поведінки персонажів. Наприклад, дослідники розробили моделі глибокого навчання, які можуть навчитися грати в такі ігри, як Flappy Bird і Super Mario Bros.

Загалом, нейронні мережі мають широкий спектр застосувань у мобільних додатках, і дослідження в цій галузі тривають у міру розробки нових методів і архітектур [11].

Раніше ми використовували GPT-2.

GPT-2, або Generative Pre-trained Transformer 2, – це модель глибокого навчання, розроблена OpenAI у 2019 році. Це найсучасніша модель мови, яка використовує нейронну мережу для створення тексту, схожого на людину. GPT-2 привернув широку увагу завдяки своїй вражаючій здатності створювати високоякісний, зв’язний і плавний текст, який часто важко відрізнити від тексту, написаного людьми.

Модель була навчена на величезному наборі даних із понад 45 терабайт тексту з широкого кола джерел, включаючи книги, статті та веб-сайти. Його було навчено за допомогою трансформаторної архітектури, яка

є типом нейронної мережі, яка особливо добре підходить для завдань обробки природної мови [9].

GPT-2 здатний виконувати широкий спектр мовних завдань, включаючи доповнення тексту, резюмування та переклад. Його також використовували для створення творчих творів, поезії та навіть музики. Модель може генерувати текст у різних стилях, від офіційної академічної мови до більш невимушеної розмовної мови.

Однак розробка GPT-2 також викликала занепокоєння щодо можливого зловживання такою технологією, зокрема для створення фейкових новин або пропаганди. У результаті OpenAI спочатку вирішив не випускати повну версію моделі, посилаючись на занепокоєння щодо можливого неправильного використання. Однак з тих пір вони випустили модель з деякими обмеженнями.

Загалом GPT-2 є значним проривом у обробці природної мови та має потенціал революціонізувати багато галузей і програм, зокрема журналістику, створення контенту та обслуговування клієнтів. Поточні дослідження в цій галузі, ймовірно, призведуть до подальших удосконалень і застосувань цієї потужної технології.

Проте зараз було прийняте рішення використовувати GPT-3.

GPT-3, або Generative Pre-trained Transformer 3 – це мовна модель, розроблена OpenAI у 2020 році. Це остання та найбільша версія серії мовних моделей GPT і вважається однією з найдосконаліших мовних моделей у світі.

Як і його попередник GPT-2, GPT-3 – це модель нейронної мережі на основі трансформатора, яка попередньо навчена на величезному наборі текстових даних. Однак GPT-3 є набагато більшим і потужнішим, ніж GPT- 2, зі 175 мільярдами параметрів, що робить його однією з найбільших моделей глибокого навчання, коли-небудь створених [5].

GPT-3 може виконувати широкий спектр завдань обробки природної мови, включаючи переклад мови, відповіді на запитання, завершення тексту та навіть творче письмо. Його розмір і складність дозволяють генерувати текст, який часто неможливо відрізнити від тексту, написаного людиною, і може виконувати завдання, які раніше вважалися неможливими для машин.

Однією з найвидатніших особливостей GPT-3 є його здатність виконувати кілька кроків навчання, що означає, що він може навчитися виконувати нові завдання лише на кількох прикладах. Це робить його дуже адаптованим до широкого спектру застосувань і галузей.

GPT-3 вже знайшов численні застосування в різних сферах, включаючи чат-ботів, обслуговування клієнтів, створення контенту та навіть кодування. Його потенційний вплив величезний, і очікується, що поточні дослідження призведуть до подальшого вдосконалення та застосування цієї потужної технології.

Однак, як і інші вдосконалені мовні моделі, GPT-3 також викликає занепокоєння щодо можливого зловживання, зокрема під час створення фейкових новин або пропаганди. Як і з будь-якою новою технологією, дуже важливо використовувати GPT-3 відповідально та етично.

1.2 Аналіз існуючих аналогів проекту

Якщо не брати у приклад мобільну розробку, можу сказати що використання нейронних мереж є вже реалізованим підходом.

AI Dungeon – це текстова пригодницька гра, створена AI, і є останнім прикладом творчого AI. Оригінальний AI Dungeon використовував AI генерації речень для створення сцен та вибору гравців, але AI Dungeon 2 має одну головну відмінність. Гравці цієї гри можуть вводити все, що завгодно, без традиційних обмежених команд та створених людиною сюжетних ліній, що обмежують їх свободу, головну сторінку цього проекту можна подивитися на рисунку 1.2.

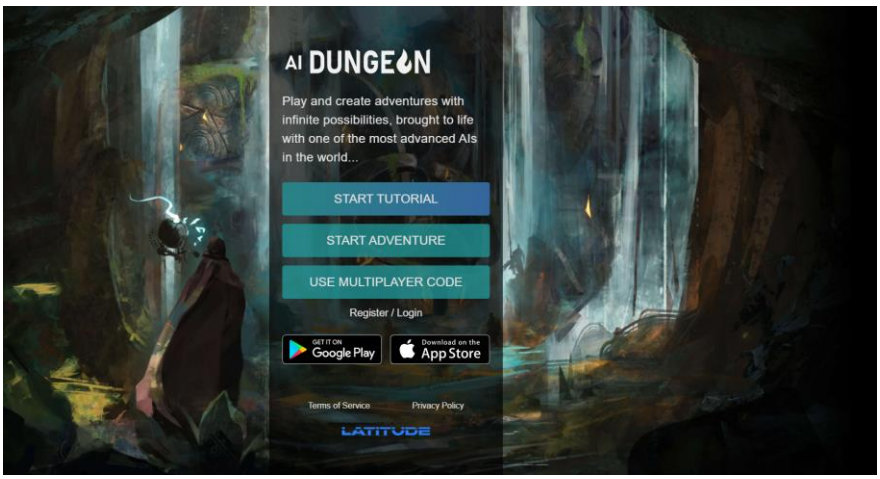

Рисунок 1.2 – Вид ігрового додатка AI Dungeon

AI Dungeon – це текстова пригодницька гра на основі штучного інтелекту, розроблена Ніком Волтоном у 2019 році. Гра використовує обробку природної мови та методи машинного навчання, щоб створити унікальний і захоплюючий ігровий досвід для гравців.

Гра працює, дозволяючи гравцям вводити текст, що описує їхні дії та рішення, який потім обробляється штучним інтелектом для створення описової відповіді. AI може розуміти та реагувати на широкий спектр вхідних даних, дозволяючи гравцям досліджувати ігровий світ і взаємодіяти з ним дуже персоналізованим способом.

Однією з ключових особливостей AI Dungeon є його здатність генерувати дуже творчі та несподівані сюжетні лінії. Штучний інтелект може створювати широкий спектр сценаріїв і персонажів, адаптуючи розповідь у реальному часі на основі даних гравця. Це забезпечує захоплюючий і динамічний ігровий досвід, унікальний для кожного гравця.

AI Dungeon набув широкої популярності з моменту свого запуску, і гравці хвалять його здатність створювати багаті та захоплюючі ігрові світи, які добре реагують на їхні дії. Гра також надихнула спільноту розробників і моддерів, які створили власні версії гри, додавши нові функції та розширивши сферу ігрового процесу.

Загалом AI Dungeon являє собою захоплюючий приклад того, як штучний інтелект і обробку природної мови можна використовувати для створення інноваційних і захоплюючих ігрових процесів. Оскільки технологія штучного інтелекту продовжує розвиватися, цілком ймовірно, що ми побачимо ще більш складні та захоплюючі ігри, які використовують потужність штучного інтелекту, щоб розширити межі традиційних ігор.

Jukedeck – це інструмент для створення музики, який використовує штучний інтелект (ШІ) для створення унікальних музичних треків, які можна налаштувати. Проект був заснований у 2012 році Едом Рексом і Патріком Стоббсом, які хотіли створити інструмент, який дозволив би будь-кому легко створювати власні персоналізовані музичні треки, не вимагаючи великих навичок створення музики (рисунок 1.3).

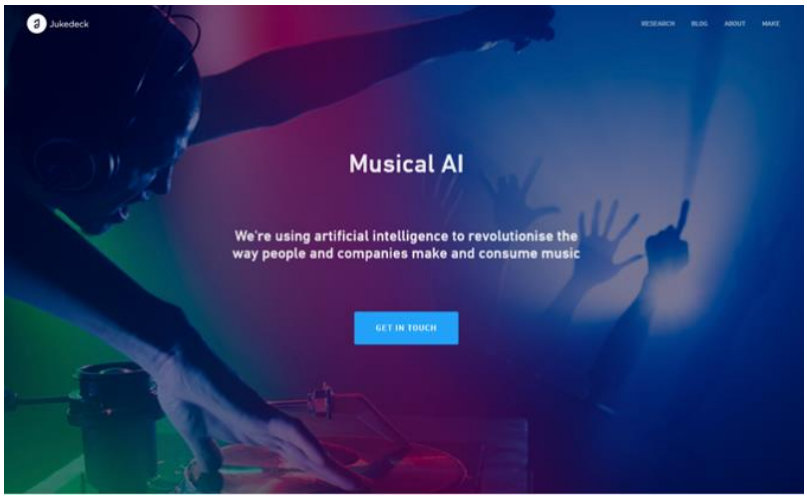

Рисунок 1.3 – Вигляд головної сторінки Jukedeck

Платформа працює за допомогою алгоритмів машинного навчання для аналізу та синтезу музичних моделей і стилів на основі ряду вхідних даних, таких як темп, настрій і жанр. Користувачі можуть вибирати з безлічі різних жанрів, включаючи класичну, поп-музику та електронну музику, а також налаштовувати темп, тривалість та інші параметри, щоб створити абсолютно унікальний трек.

Jukedeck також пропонує бібліотеку вже існуючих музичних композицій, які користувачі можуть додатково налаштувати або використовувати як є для широкого спектру додатків, таких як фонова музика для відео, реклами чи іншого мультимедійного вмісту.

Однією з ключових переваг Jukedeck є його здатність швидко та ефективно створювати музичні треки високої якості. Замість того, щоб складати та записувати кожен окремий інструмент окремо, штучний інтелект створює повний трек лише за кілька хвилин, який потім можна налаштувати та відредагувати за потреби.

Платформа набула популярності серед творців контенту, режисерів та інших творчих професіоналів, які потребують високоякісної музики для своїх проектів, але можуть не мати часу чи ресурсів для створення оригінальної музики самостійно. Він також використовувався в різноманітних комерційних програмах, таких як рекламні кампанії, де можливість швидкого створення настроюваних музичних треків може бути головною перевагою.

Загалом, Jukedeck являє собою захоплюючий приклад того, як ШІ можна використовувати для автоматизації та спрощення складних творчих завдань, дозволяючи будь-кому швидко та легко створювати високоякісні музичні треки. Оскільки технології штучного інтелекту продовжують розвиватися, цілком ймовірно, що ми побачимо ще більше інноваційних і доступних інструментів для створення музики та іншого творчого контенту.

1.3 Аналіз методів рішення

Існує кілька різних методів створення текстових генераторів, зокрема:

– системи на основі правил: ці системи використовують набір попередньо визначених правил і шаблонів для створення тексту. Правила можуть ґрунтуватися на лінгвістичних правилах, таких як граматика та синтаксис, або на правилах, що стосуються предметної області, наприклад,

на галузевій термінології чи юридичній мові. Системи, засновані на правилах, можуть бути ефективними для створення простого та передбачуваного тексту, але вони можуть бути недостатньо гнучкими для вирішення складніших мовних завдань [12];

– моделі Маркова: ці моделі використовують статистичні методи для створення тексту на основі ймовірності появи певних слів або фраз у певному контексті. Моделі можна навчати на великому корпусі тексту, що дозволяє їм генерувати текст, який є більш природним і різноманітним, ніж системи на основі правил. Однак моделі Маркова можуть мати проблеми з довшими послідовностями тексту або складнішими мовними завданнями;

– повторювані нейронні мережі (RNN): RNN – це тип моделі глибокого навчання, яку можна навчити на великих наборах тексту для створення нового тексту. RNN особливо корисні для генерації тексту, який є більш складним і різноманітним, ніж системи на основі правил або моделі Маркова. RNN працюють, обробляючи кожне слово в послідовності, а потім використовуючи цю інформацію для створення наступного слова. Це дозволяє їм фіксувати довгострокові залежності та генерувати більш природний текст;

– моделі-трансформери. Моделі-трансформери – це тип моделі глибокого навчання, яка використовує механізми уваги для створення тексту. Ці моделі особливо ефективні для завдань обробки природної мови, таких як переклад мови, підсумовування тексту та генерація тексту. Моделі трансформаторів можна навчити на великих наборах тексту та генерувати дуже точний текст із природним звучанням.

Кожен метод має свої сильні та слабкі сторони, і вибір методу залежатиме від конкретного завдання та наявних ресурсів. Зрештою, метою генераторів тексту є створення природного та точного тексту, і використовувані методи слід вибирати з цією метою.

BERT (Bidirectional Encoder Representations from Transformers) – це потужна мовна модель, яка була навчена на великому корпусі тексту та

досягла найсучаснішої продуктивності в широкому діапазоні завдань обробки природної мови. BERT особливо ефективний у створенні високоякісного тексту та використовується в різноманітних додатках, таких як чат-боти, мовний переклад і резюмування тексту [7].

Однак BERT не завжди є найкращим вибором для завдань генерації тексту, особливо якщо метою є генерація довшого та складнішого тексту. BERT розроблено для генерування тексту по одному слову, що може бути обмеженням при спробі генерувати довші послідовності тексту. Крім того, BERT не завжди може охопити всю складність мови, особливо у випадках, коли мова неоднозначна або вимагає тонкого розуміння контексту.

З цих причин інші методи, такі як RNN або трансформаторні моделі, можуть бути кращим вибором для певних завдань генерації тексту. Ці моделі створені для генерування довших послідовностей тексту та краще вловлюють складні залежності та нюанси мови. Крім того, ці моделі можна навчити на великих наборах тексту, що може допомогти підвищити їх точність і природність.

Підводячи підсумок, хоча BERT є потужною мовною моделлю, яку можна використовувати для створення тексту, він не завжди може бути найкращим вибором для кожного завдання створення тексту. Вибір моделі залежатиме від конкретних вимог поставленого завдання та наявних ресурсів.

GPT-3 і BERT є потужними мовними моделями, які досягли найсучаснішої продуктивності в широкому діапазоні завдань обробки природної мови. Однак між двома моделями є деякі ключові відмінності.

GPT-3 (Generative Pre-trained Transformer 3) – це модель мови на основі трансформатора, розроблена OpenAI. GPT-3 розроблено для генерування тексту на основі заданої підказки та може використовуватися для широкого спектру завдань обробки природної мови, включаючи переклад мови, відповіді на запитання та резюмування тексту. GPT-3 вирізняється своїм великим розміром із 175 мільярдами параметрів, що

робить його однією з найбільших доступних мовних моделей. GPT-3 також використовує неконтрольоване навчання, що означає, що його можна навчати на великих обсягах немаркованих даних, що дозволяє генерувати дуже точний текст, що звучить природно.

BERT (Bidirectional Encoder Representations from Transformers) також є мовною моделлю на основі трансформатора, розробленою Google. BERT розроблено для генерації контекстуалізованого вбудовування для кожного слова в даному реченні, що дозволяє охопити повний контекст речення. Це дозволяє BERT добре виконувати широкий спектр завдань обробки природної мови, включаючи переклад мови, відповіді на запитання та класифікацію тексту. BERT відомий своєю здатністю генерувати дуже точні та нюансовані представлення мови, що дозволяє йому добре виконувати завдання, які вимагають тонкого розуміння контексту [12].

Однією з ключових відмінностей між GPT-3 і BERT є відповідні розміри. Хоча GPT-3 набагато більший за BERT, це не обов’язково означає, що він завжди точніший або краще підходить для кожного завдання. Крім того, незважаючи на те, що обидві моделі дуже ефективні для створення природного звучання тексту, кожна з них має свої сильні та слабкі сторони, і вибір моделі залежатиме від конкретних вимог поставленого завдання.

Підводячи підсумок, хоча GPT-3 і BERT є високоефективними мовними моделями, кожна з них має свої сильні та слабкі сторони, і вибір моделі залежатиме від конкретних вимог поставленого завдання.

1.4 Постановка задачі

Метою роботи є дослідження різних нейронних мереж для використовуванні для додатка генерирующего квест/історію на мобільні пристрої. Програмне забезпечення для цього повинне складатись з роботи з моделью для її використання та роботи з інтерфейсом.

Для того щоб вирішити цю задачу треба вирішити наступні завдання:

- провести аналіз GPT-2/GPT-3/BERT та можливості працювати з ними;
- знайти спосіб зберегти інформацію у мобільному додатку;
- розробити дизайн для легкої взаємодії з користувачем;
- визначити яку найбажану версію Android SDK/Gradle та інших бібліотек треба обрати для роботи на більшому кількості приладів;
- розробити програмну реалізацію вбудови GPT-2/GPT-3/BERT у додаток з його моделью.

Також треба щоб даний додаток був гібкий, тото в ньго можно було встроїти щось інше, додатковий функціонал. Це також впливає на її функціонал у потенціалі.

Програмно реалізований продукт повинна виконувати наступні функції:

- можливість користувачеві начати генерацію тексту;
- можливість змінити стартовий текст на інший рандомный;
- перегляд сгенерированного тексту;
- можливість забракувати текст та начати перегенерування;
- можливість обрати сгенерированный текст;
- можливість перглянути текст який вийшов завдяки продовження генеруваннем;
- можливість виконувати усі потрібні дії так щоб можна було нескінченно генерувати текст;
- можливість Зберегти отриману історію;
- можливість переглянути історію яку зберегли.

Головною функцією є можливість генерації тексту. Завдяки цьому користувач повинен мати змогу отримати цікаву історію для своїх потреб.

Зібрати статистику праці у додатку та оцінити ефективність роботи потрібної мовної моделі.

2 ОПИС ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

Даний програмний продукт повинен розроблятися як мобільний додаток, у якості ядра повинна виступати нейронна мережа, деяка мовна модель. Це буде сприяти більш різноманітної свободи діяльності історій/квестів генератора.

База даних повинна зберігати історії, та мати змогу розширюватися, якщо реалізовувати потенціал цього мобільного додатку інтелектуального квест-генератора.

Також потрібно створити цей мобільний додаток з виконанням архітектури проектування MVVM. Архітектура MVVM - це архітектура Model-View-ViewModel, яка усуває тісне зчеплення між кожним компонентом, діаграма вигляду цього патерну проектування зображена на рисунку 2.1.

Рисунок 2.1 – Діограмма паттерна проектирования MVVM (Model - View-Model View)

Bloc (Business Logic Component) і MVVM (Model-View-ViewModel) – це два популярні архітектурні шаблони, які використовуються в розробці додатків Flutter для керування станом додатків. Bloc – це архітектурний шаблон, який відокремлює бізнес-логіку програми від інтерфейсу користувача, полегшуючи керування та тестування. За допомогою Bloc

інтерфейс користувача взаємодіє з бізнес-логікою через потоки подій і потоки станів. Коли подія отримана, бізнес-логіка обробляє її та генерує новий стан, який потім надсилається назад до інтерфейсу користувача. Таке поділ проблем полегшує міркування про поведінку програми та полегшує тестування бізнес-логіки окремо [1].

MVVM – це архітектурний шаблон, який також використовується для відділення інтерфейсу користувача від бізнес-логіки. У MVVM інтерфейс користувача складається з двох компонентів: представлення та моделі представлення. Представлення відповідає за рендеринг інтерфейсу користувача, тоді як модель представлення відповідає за керування станом програми та надання даних для представлення. Модель перегляду взаємодіє з бізнес-логікою через репозиторій, який абстрагує деталі того, як дані витягуються та зберігаються. Такий розподіл завдань полегшує тестування інтерфейсу користувача та бізнес-логіки окремо, а також полегшує підтримку та масштабування коду.

У Flutter як Bloc, так і MVVM можна реалізувати за допомогою пакетів сторонніх розробників. Пакет `flutter_bloc` надає набір інструментів для реалізації шаблону Bloc у Flutter, тоді як пакет `provider` надає набір інструментів для реалізації шаблону MVVM. Обидва візерунки мають свої сильні та слабкі сторони, і вибір шаблону залежатиме від конкретних вимог програми. Flutter і Dart є популярним вибором для розробки мобільних додатків через їхні численні переваги.

По-перше, Flutter забезпечує швидкий і ефективний процес розробки. Функція гарячого перезавантаження Flutter дозволяє розробникам вносити зміни в код і миттєво бачити результати, що полегшує ітерацію та тестування програми. Крім того, архітектура Flutter на основі віджетів полегшує створення та налаштування елементів інтерфейсу користувача, зменшуючи потребу у сторонніх бібліотеках.

По-друге, Flutter є кросплатформною структурою, що означає, що єдину кодову базу можна використовувати для створення програм як для

iOS, так і для Android, а також для веб-платформ і настільних платформ. Це скорочує час і вартість розробки, а також полегшує підтримку кодової бази.

По-третє, Dart – сучасна мова програмування, розроблена спеціально для побудови користувальницьких інтерфейсів. Dart легко освоїти та надає такі функції, як статична перевірка типів і збирання сміття, що робить його надійною та ефективною мовою для розробки програм.

Нарешті, Google підтримує Flutter і Dart, а це означає, що вони постійно розвиваються і вдосконалюються. Це надає розробникам доступ до великої спільноти розробників, а також до широкого спектру ресурсів та інструментів, які допоможуть їм створювати високоякісні програми.

Підводячи підсумок, Flutter і Dart забезпечують швидкий і ефективний процес розробки, є кросплатфомними, простими у вивченні та використанні та постійно розвиваються і вдосконалюються. Ці переваги роблять їх популярним вибором для розробки мобільних додатків.

GPT-3 – це потужна мовна модель, до якої можна отримати доступ через API (інтерфейс прикладного програмування). Цей API дозволяє розробникам інтегрувати GPT-3 у свої додатки та використовувати його можливості обробки природної мови для створення тексту, відповідей на запитання, перекладу мов та виконання низки інших завдань, пов’язаних із мовою. Є кілька способів використання GPT-3 API. Одним із способів є використання OpenAI Playground, який є веб-інтерфейсом, який дозволяє користувачам взаємодіяти з GPT-3 без написання коду. Іншим способом є використання однієї з багатьох оболонок API GPT-3, доступних у різних мовах програмування, таких як Python, JavaScript і Ruby. Ці оболонки забезпечують більш зручний для користувача спосіб взаємодії з API та обробки необхідних автентифікації та форматування даних.

Щоб використовувати API GPT-3, розробникам потрібно створити обліковий запис у OpenAI та отримати ключ API. Отримавши ключ API, вони можуть використовувати його для автентифікації своїх запитів і доступу до API GPT-3. Потім вони можуть надсилати запити до API для

виконання таких завдань, як створення тексту, відповіді на запитання та заповнення форм. API повертає результати у форматі JSON, який розробники можуть аналізувати та використовувати у своїх програмах [5].

Деякі приклади додатків, які використовують API GPT-3, включають чат-боти, генератори вмісту, інструменти мовного перекладу та віртуальних помічників. API GPT-3 – це потужний інструмент, який може розширити можливості багатьох різних типів програм і допомогти розробникам створити більш привабливу та інтерактивну взаємодію з користувачем.

Також програмний додаток повинен мати інтерфейс, зручний для користувача. Також треба додати такі функції як:

- Activity яка буде відображати деяке привітання;
- головна сторінка з усіма функціями які повинні бути потрібні користувачеві у використанні мобільного додатку;
- мобільний додаток також повинен бути оптимізованим до роботи на різних мобільних пристроях.

До роботи цього мобільного додатку були висунуті наступні технічні потреби:

- для запуску інтелектуального генератора потрібен пристрій який має операційну систему Android/IOS;
- версія Android не меньш ніж 8.1;
- версія IOS не меньш ніж 13.

До роботи з розробкою мобільного додатку, були висунуті наступні вимоги:

- наявність Android SDK мінімально версії API 26 компілювання буде проходити до версії API 31;
- наявність JVM версії 1.8;
- мати сборщик Gradle версії 4.6.14;
- встановити AVD на потрібному SDK;
- наявності AVD під Pixel 3a API 31.

3 ОПИС ПРИЙНЯТИХ ПРОЕКТНИХ РІШЕНЬ

3.1 Аналіз та UML моделювання предметної області

UML (Unified Modeling Language) – це графічна нотація, яка використовується для моделювання програмних систем. Це стандартизована мова, яка використовується розробниками програмного забезпечення, архітекторами та дизайнерами для створення діаграм, які представляють різні аспекти програмної системи. Існує кілька типів діаграм, які можна створити за допомогою UML, включаючи діаграми варіантів використання, діаграми класів, діаграми послідовності, діаграми діяльності та діаграми станів.

Діаграми варіантів використання використовуються для опису функціональності системи з точки зору користувача. Вони зображують взаємодію між користувачами та системою та допомагають визначити варіанти використання та учасників, задіяних у системі.

Діаграми класів використовуються для опису класів у системі, їхніх атрибутів і зв’язків з іншими класами. Вони допомагають розробникам визначити об'єкти, які будуть використовуватися в системі, і їх властивості.

Діаграми послідовності використовуються для зображення взаємодії між об’єктами в системі протягом певного часу. Вони показують порядок, у якому обмінюються повідомленнями між об’єктами, і допомагають розробникам визначити послідовність дій, що відбуваються в системі.

Діаграми діяльності використовуються для опису потоку діяльності в системі. Вони показують етапи процесу та допомагають розробникам визначити рішення та розгалуження, які відбуваються в системі.

Діаграми стану використовуються для опису поведінки об’єктів у системі. Вони показують різні стани, в яких може перебувати об’єкт, і події, які викликають його перехід з одного стану в інший.

UML-моделювання – це потужний інструмент, який допомагає розробникам і дизайнерам чітко й ефективно передавати свої ідеї та концепції. Це невід'ємна частина процесу розробки програмного забезпечення та використовується для створення схем, які керують розробкою програмних систем (рисунок 3.1).

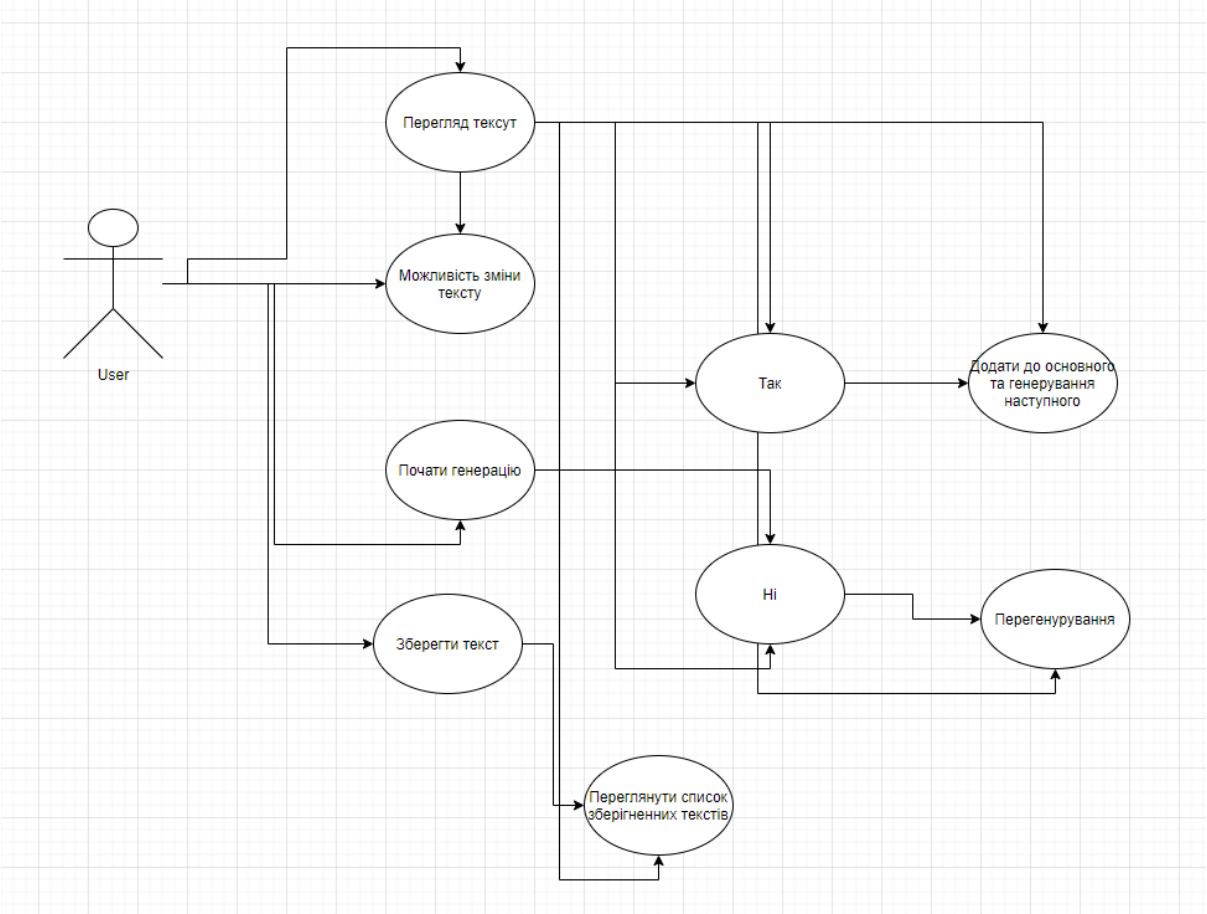

Рисунок 3.1 – Use-case діаграма

Тому якщо подивитися на рисунок 3.1 ми можемо побачити що наш програмний додаток виконує ці функції для користувача або на його можливостях.Виконувати наступні функції:

- перегляд початкового тексту;
- можливість користувачеві начати генерацію тексту;
- можливість змінити стартовий текст на інший рандомний;

- перегляд сгенерированного тексту;
- можливість забракувати текст та начати перегенерування;
- можливість обрати сгенерированный текст;
- можливість перглянути текст який вийшов завдяки продовження генеруванням;
- можливість виконувати усі потрібні дії так щоб можна було нескінченно генерувати текст;
- можливість зберегти отриману історію;
- можливість переглянути історію яку зберегли.

Головною функцією є можливість генерації тексту.

3.2 Аналіз та вибір метода генерування тексту

GPT-2 (Generative Pre-trained Transformer 2) – це модель мови, розроблена OpenAI. Це глибока нейронна мережа, яка використовує неконтрольоване навчання для створення тексту природною мовою. GPT-2 було навчено на великому корпусі тексту з Інтернету, і він може генерувати текст у широкому діапазоні стилів і форматів.

GPT-2 має 1,5 мільярда параметрів, що робить його однією з найбільших мовних моделей на момент випуску. Він використовує трансформаторну архітектуру, що дозволяє обробляти та генерувати текст паралельно. GPT-2 здатний генерувати зв’язний текст, схожий на людину, і його можна використовувати для різноманітних завдань обробки природної мови, включаючи завершення тексту, переклад і резюмування.

Однією з унікальних особливостей GPT-2 є його здатність генерувати текст у широкому діапазоні стилів і форматів. Він може генерувати текст, який нагадує новинні статті, огляди продуктів, вірші та навіть комп’ютерний код. Це робить його корисним інструментом для різноманітних додатків, включаючи чат-ботів, створення вмісту та навіть творче написання.

GPT-2 також піддається критиці за його здатність генерувати оманливий або шкідливий вміст. Через його здатність генерувати текст, який нагадує написане людиною, існує занепокоєння щодо його потенційного використання для фейкових новин, пропаганди чи інших зловмисних цілей. У результаті OpenAI спочатку вирішила не випускати повну версію GPT-2 для громадськості через побоювання щодо її можливого неправильного використання. Однак з тих пір модель стала доступною для дослідників і розробників за певних умов, схема на рисунку 3.2.

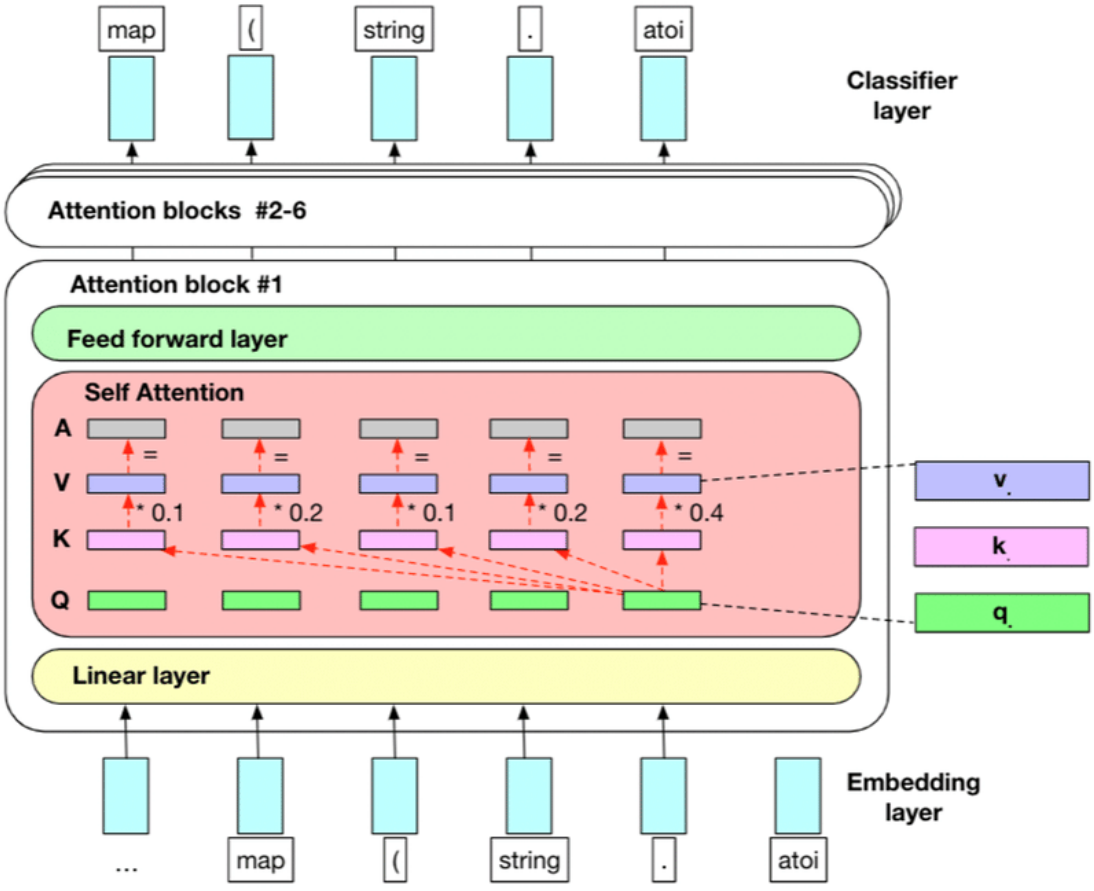

Рисунок 3.2 – Схема GPT-2

GPT-3 (Generative Pre-trained Transformer 3) – це найдосконаліша модель мови, розроблена OpenAI, яка була випущена в червні 2020 року.

GPT-3 має значно більше параметрів, ніж його попередник GPT-2, до 175 мільярдів параметрів, що робить її однією з найбільших існуючих мовних моделей [1].

GPT-3 – це модель глибокого навчання, яка попередньо навчена на великому масиві текстових даних і може генерувати текст природною мовою в широкому діапазоні форматів і стилів. Однією з ключових особливостей GPT-3 є його здатність виконувати різноманітні завдання обробки природної мови, включаючи переклад мови, завершення тексту, резюмування та відповіді на запитання, з надзвичайною точністю та плавністю [7].

GPT-3 також має здатність вчитися та виконувати нові завдання з мінімальними навчальними даними, відоме як кількакратне навчання. Це означає, що він може швидко адаптуватися до нових завдань обробки мови, що робить його універсальним інструментом для широкого спектру програм.

Однією з унікальних особливостей GPT-3 є його здатність генерувати текст, який є дуже зв’язним, плавним і навіть творчим. Він може створювати природно звучачий текст у різноманітних стилях і тонах, починаючи від звичайної розмови та закінчуючи академічним письмом.

GPT-3 має потенціал революціонізувати обробку природної мови та змінити спосіб взаємодії з комп’ютерами та іншими інтелектуальними системами. Однак він також викликав занепокоєння щодо його потенційного впливу на робочі місця та конфіденційність, а також щодо його можливого зловживання для створення оманливого або шкідливого змісту. Таким чином, тривають дебати та дискусії щодо відповідального розвитку та використання GPT-3 та інших передових мовних моделей. Схема на рисунку 3.3.

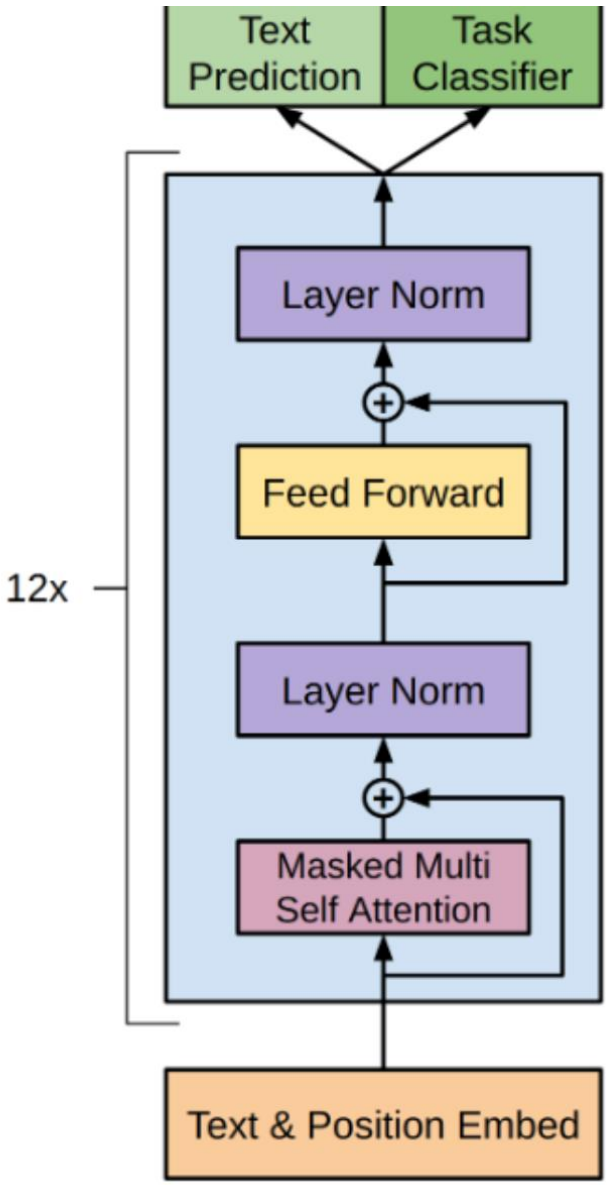

Рисунок 3.3 – Схема GPT-3

GPT-2 (Generative Pre-trained Transformer 2) і GPT-3 (Generative Pre-trained Transformer 3) – обидві мовні моделі, розроблені OpenAI, причому GPT-3 є новішою та вдосконаленою моделлю. Ось деякі з ключових відмінностей між двома моделями:

– розмір моделі: GPT-2 має 1,5 мільярда параметрів, тоді як GPT-3 має до 175 мільярдів параметрів. Це означає, що GPT-3 має значно більше можливостей для навчання та може генерувати більш складний і різноманітний текст, ніж GPT-2;

– генерація тексту: хоча обидві моделі можуть генерувати зв’язний і схожий на людину текст, GPT-3 зазвичай вважається більш плавним і універсальним, ніж GPT-2. Це пов’язано з його більшою місткістю та більш досконалими методами навчання;

– постійне навчання: GPT-3 має здатність навчатися та виконувати завдання з мінімальними навчальними даними, відоме як поодиноке навчання. Це означає, що він може навчитися виконувати нові завдання швидко й точно, не потребуючи великих навчальних даних. GPT- 2, з іншого боку, вимагає більше навчальних даних для виконання нових завдань [9].

Швидкість виведення: GPT-3 має вищу швидкість виведення, ніж GPT-2, тобто він може генерувати текст швидше та ефективніше.

Різноманітність вихідних даних: GPT-3 має можливість створювати більш широкий спектр вихідних даних, включаючи різні стилі та тони написання, а також більш різноманітний вміст. Це пов’язано з його більшою потужністю та більш досконалими методами навчання.

Загалом, GPT-3 є більш потужною та універсальною мовною моделлю, ніж GPT-2, з більшою здатністю до вивчення та створення більш складного та різноманітного тексту. Його можливості швидкого навчання та більш висока швидкість висновку також роблять його ефективнішим і ефективним інструментом для завдань обробки природної мови. Схема різниці на рисунку 3.4.

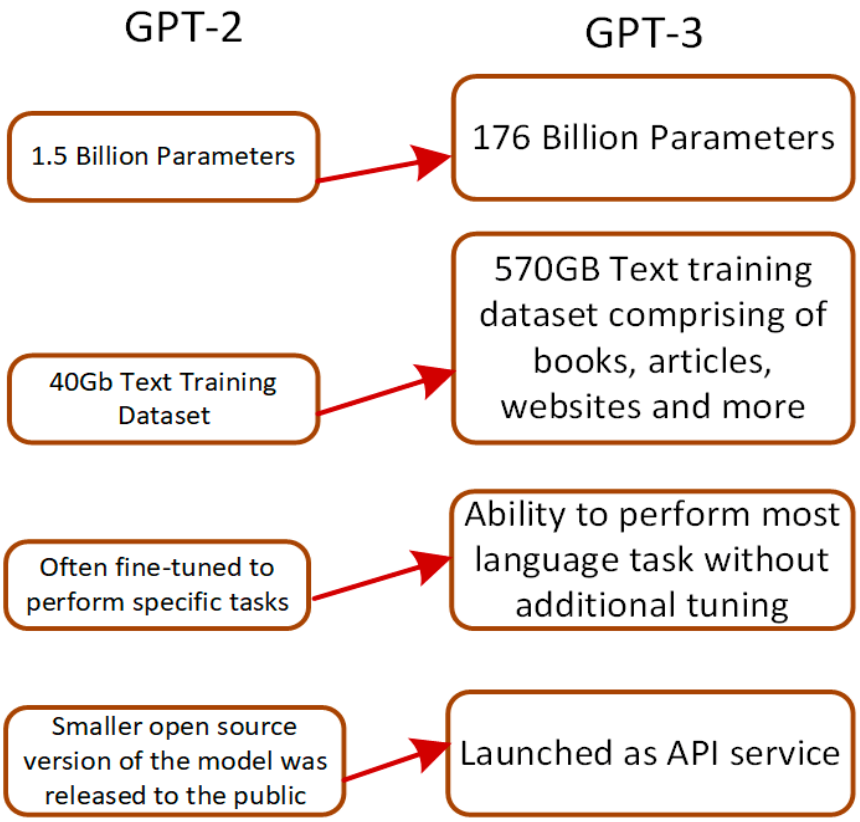

Рисунок 3.4 – Схема змін між GPT-2 та GPT-3

GPT-3 має кілька переваг перед GPT-2, коли справа доходить до використання його для мобільних програм. Ці переваги включають:

Швидкість і ефективність: GPT-3 може виконувати завдання обробки мови набагато швидше, ніж GPT-2, завдяки більшій кількості параметрів і оптимізованій архітектурі. Це означає, що він може надавати результати швидше та ефективніше, що важливо для мобільних додатків, які потребують обробки в режимі реального часу або майже в реальному часі.

Невелике навчання: GPT-3 має здатність навчатися та виконувати нові завдання з мінімальними навчальними даними, що робить його більш універсальним інструментом для мобільних додатків. Це означає, що розробники можуть використовувати GPT-3 для створення мобільних програм, які можуть виконувати широкий спектр завдань обробки природної мови, не витрачаючи багато часу на навчання моделі конкретним завданням.

Точність і плавність: GPT-3 точніший і сильніший, ніж GPT-2, коли йдеться про генерування тексту природною мовою. Це означає, що мобільні програми, які використовують GPT-3, можуть надавати точніші та природні результати, що важливо для таких програм, як чат-боти чи віртуальні помічники, яким потрібно спілкуватися з користувачами природним та інтуїтивно зрозумілим способом.

Більші набори даних: GPT-3 навчався на більшому наборі даних, ніж GPT-2, що означає, що він має ширше розуміння природної мови та ширший діапазон знань, на які можна спиратися. Це робить його потужним інструментом для мобільних додатків, яким необхідно обробляти та генерувати текст у різноманітних контекстах [11].

Загалом, хоча GPT-2 все ще є потужною мовною моделлю, GPT-3 має кілька переваг, які роблять його кращим вибором для мобільних програм, які потребують швидкої та ефективної обробки природної мови.

Тому як сказано вище, мовна модель gpt-3 є краща якою ми можемо користуватися. Крім нього зрозуміло є його наступник gpt-4. Але ми не можемо його використовувати тому що він у закритому доступі. Щоб получити цей доступ треба подавати заявку та чекати своєї черги і вірити що openAI цікаве мета його використовування тобою.

API і робота з мовними моделями на пристрої мають свої переваги та недоліки. API (інтерфейс прикладного програмування) – це спосіб віддаленого доступу до мовної моделі, зазвичай через веб-службу. API мають кілька переваг:

– доступність: доступ до API можна отримати з будь-якого пристрою з підключенням до Інтернету, що робить їх дуже доступними;

– масштабованість: API можуть обробляти великий обсяг запитів, що робить їх придатними для використання в програмах, які вимагають високих рівнів обчислювальної потужності;

– економічна ефективність: використання API може бути економічно ефективнішим, ніж навчання та підтримка мовної моделі на пристрої, особливо для невеликих програм.

Робота з мовними моделями на пристрої.

Робота з мовною моделлю на пристрої означає, що модель встановлюється безпосередньо на пристрій і не вимагає підключення до віддаленого сервера. Цей підхід має кілька переваг:

– конфіденційність: робота з мовною моделлю на пристрої гарантує, що дані користувача залишаються конфіденційними, оскільки вони не передаються через Інтернет;

– швидкість: мовні моделі, які працюють на пристрої, можуть надати результати набагато швидше, ніж API, оскільки вони не вимагають підключення до мережі;

– можливості в автономному режимі: мовні моделі, які працюють на пристрої, можуть надавати результати навіть за відсутності підключення до Інтернету.

Однак робота з мовними моделями на пристрої також може мати деякі недоліки:

– вимоги до ресурсів. Великі мовні моделі можуть потребувати значного обсягу пам’яті та потужності обробки, що може бути неможливим для всіх пристроїв;

– технічне обслуговування: робота з мовною моделлю на пристрої потребує обслуговування та оновлень, щоб гарантувати, що вона залишається точною та актуальною;

– загалом, використання API чи робота з мовною моделлю на пристрої залежить від конкретних вимог програми. API, як правило, є більш масштабованими та економічно ефективними, тоді як робота з мовними моделями на пристрої забезпечує більшу конфіденційність, швидкість і можливості офлайн.

4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Спочатку налаштуємо проект та додаємо необхідні бібліотеки (рисунок 4.1).

```yaml
name: ai_gpt_story_generator
description: AI Story Generator with GPT-3 from OpenAI

publish_to: 'none'

version: 1.0.0+1

environment:
  sdk: '>=2.19.3 <3.3.10'


dependencies:
  flutter:
    sdk: flutter

  #LocalDatabase
  hive: ^2.2.3
  hive_flutter: ^1.1.0
  build_runner: ^2.3.2

  #Architecture
  flutter_bloc: ^8.1.2

  #UI/UX
  flutter_neumorphic: ^3.2.0
  cupertino_icons: ^1.0.2
  flutter_switch: ^0.3.2
  google_fonts: ^3.0.1
  sizer: ^2.0.15
  show_up_animation: ^2.0.0
  animated_text_kit: ^4.2.2
```

Рисунок 4.1 – pubspec.yaml бібліотеки додатка(частина 1)

hive – Flutter HIVE – це легка та швидка база даних NoSQL, розроблена спеціально для програм Flutter. Це бібліотека з відкритим кодом, яка дозволяє розробникам Flutter зберігати та отримувати дані в локальній базі даних на пристрої користувача [2].

HIVE має кілька переваг для розробників Flutter, зокрема:

Висока продуктивність: HIVE оптимізовано для швидкості та може легко обробляти великі обсяги даних.

Простота: HIVE простий у використанні з простим API, який легко зрозуміти.

Незалежність від платформи: HIVE може працювати на кількох платформах, включаючи Android, iOS та веб.

Безпека типів: HIVE безпечний для типів, що означає, що розробники можуть визначати типи даних, які зберігаються в базі даних.

Шифрування: HIVE забезпечує вбудоване шифрування для даних, що зберігаються в базі даних, що допомагає зберігати дані користувача в безпеці.

Легкість: HIVE – це легка бібліотека, що означає, що вона не додає значних витрат для програм Flutter.

HIVE часто використовується в програмах Flutter, які вимагають офлайн-зберігання та отримання даних, таких як програми для створення нотаток, програми зі списками справ та інші програми для підвищення продуктивності. Його також можна використовувати в поєднанні з іншими бібліотеками та фреймворками, такими як Provider, для створення більш складних програм, які вимагають збереження даних (рисунок 4.2).

```
#Other
path_provider: ^2.0.11
url_launcher: ^6.1.7
share_plus: ^6.0.0
shared_preferences: ^2.0.18
google_mobile_ads: ^2.3.0
firebase_analytics: ^10.1.6
chat_gpt_sdk: ^2.0.7
```

Рисунок 4.2 – pubspec.yaml бібліотеки додатка(частина 2)

firebase_analytics – Flutter Firebase Analytics – це бібліотека, яка дозволяє розробникам Flutter інтегрувати відстеження аналітики у свої мобільні програми. Firebase Analytics – це безкоштовний інструмент від Google, який дозволяє розробникам відстежувати та аналізувати поведінку користувачів у своїх програмах. Flutter Firebase Analytics простий у налаштуванні та використанні завдяки простому API, який дозволяє розробникам почати відстежувати дані аналітики у своїх програмах за допомогою лише кількох рядків коду. Він надає цінну інформацію про поведінку користувачів, яка може допомогти розробникам приймати керовані даними рішення щодо розробки додатків і покращити загальну взаємодію з користувачем.

chat_gpt_sdk – Бібліотека яку ми будемо використовувати для роботу з мовною моделью, вона уявляє з себе швидкий API який можно використовувати у програмному додатку.

Таким чином ми розпочали працю над додатком. Першу версію додатка можно побачити на рисунку 4.3 вона розроблена у моїй бакалаврській роботі.

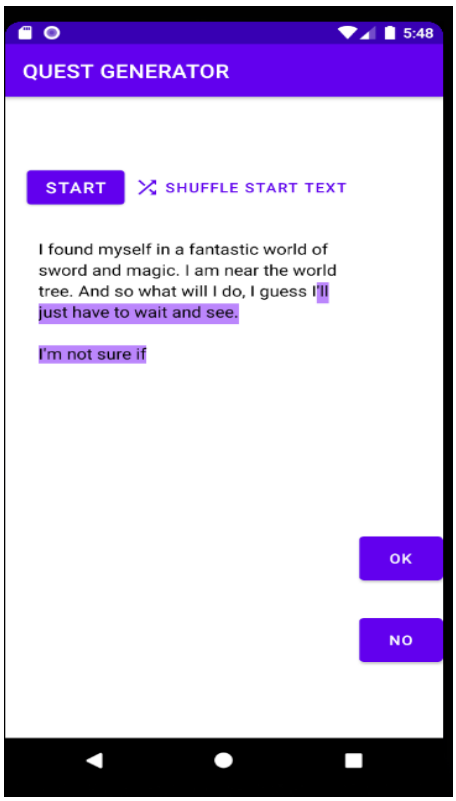

Рисунок 4.3 – Візуальний інтерфейс мобільного додатку(1 версія)

Спочатку так у першій версії ми працювали з мовою Kotlin. Kotlin – це мова програмування, розроблена компанією JetBrains, розробниками популярної IntelliJ IDEA IDE. Kotlin – це статично типізована мова загального призначення, яка розроблена для повної взаємодії з Java і може бути скомпільована для роботи на віртуальній машині Java (JVM [1].

Деякі з ключових функцій Kotlin включають:

Лаконічність: Kotlin дозволяє розробникам писати більш стислий код, ніж Java, зберігаючи при цьому читабельність і ясність.

Захист від нуля: Kotlin має вбудовані функції безпеки від нуля, які допомагають запобігти винятком нульового покажчика в коді.

Взаємодія: Kotlin може бути повністю сумісним з Java, дозволяючи розробникам використовувати Kotlin в існуючих проектах Java і навпаки.

Функції розширення: Kotlin дозволяє розробникам додавати нові функції до існуючих класів, навіть якщо вони не мають доступу до вихідного коду.

Співпрограми: Kotlin має вбудовану підтримку співпрограм, які є легким інструментом паралелізму, що дозволяє легко писати асинхронний код.

Функціональне програмування: Kotlin підтримує багато концепцій функціонального програмування, таких як лямбда, функції вищого порядку та незмінність.

Останніми роками Kotlin набув популярності, особливо в спільноті розробників Android, завдяки своїй простоті використання, сумісності з Java та сильним функціям нульової безпеки. Він також широко використовується в бекенд-розробці.

Треба розробити додаток не тільки для Android а ще і для IOS платформи тому ми перейшли до Flutter. Архітектуру проекта можно побачити на рисунку 4.4.

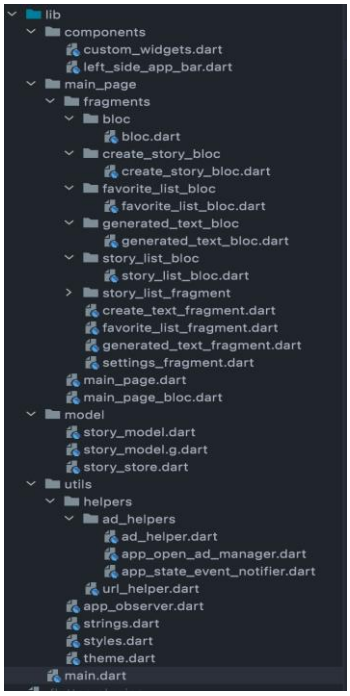

Рисунок 4.4 – Архітектура Flutter додатка.

У Flutter є кілька популярних архітектурних шаблонів, які розробники можуть використовувати для структурування своїх проектів. Деякі з найбільш часто використовуваних шаблонів включають:

Model-View-Controller (MVC), у цьому шаблоні код розділений на три основні частини: модель (яка керує даними), представлення (яке відображає дані) і контролер (який діє як посередник між модель і вид).

Model-View-ViewModel (MVVM), цей шаблон схожий на MVC, але з додаванням рівня ViewModel, який відповідає за показ даних у поданні та обробку введених користувачем даних.

Блок (компонент бізнес-логіки), це шаблон керування станом, який відокремлює рівень презентації від бізнес-логіки. У цьому шаблоні код розділений на три частини: рівень інтерфейсу користувача (який обробляє взаємодію з користувачем), рівень Bloc (який містить бізнес-логіку) і рівень даних (який керує даними).

Постачальник: це ще один шаблон керування станом, який спрощує процес передачі даних між віджетами в програмі Flutter. У цьому шаблоні віджет постачальника використовується для надання даних своїм нащадкам, які потім можуть прослуховувати зміни в цих даних і оновлюватися відповідно.

Redux: це передбачуваний шаблон керування станом, який відокремлює стан програми від рівня інтерфейсу користувача. У цьому шаблоні код розділений на три частини: сховище (яке містить стан програми), редуктор (який обробляє переходи між станами) і представлення (яке відображає стан для користувача).

Вибір архітектурного малюнка буде залежати від конкретних вимог і складності проекту. Важливо ретельно розглянути плюси та мінуси кожного шаблону, перш ніж прийняти рішення про те, що найкраще підходить для вашого проекту.

Тому так як ми обрали BloC архітектуру тут можна побачити як вона використовується та її структуру. На рисунку 4.4 її використання.

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

  @override
  Widget build(BuildContext context) {
    return BlocBuilder<ThemeCubit, NeumorphicThemeData>(builder: (_, theme) {
      return Sizer(builder: (context, orientation, deviceType) {
        return NeumorphicApp(
          theme: theme,
          home: const MainPage(),
        );  // NeumorphicApp
      });  // Sizer
    });  // BlocBuilder
  }
}
```

Рисунок 4.4 – Приклад використання BloC архітектури.

На рисунку 4.5 можно побачити як ініціалізовані різні бібліотеки які потрібні у додатку.

```dart
void initialize() {
  WidgetsFlutterBinding.ensureInitialized();
  Bloc.observer = const AppBlocObserver();
  MobileAds.instance.initialize();
  AppLifecycleReactor appLifecycleReactor;
  AppOpenAdManager appOpenAdManager;
  appOpenAdManager = AppOpenAdManager()..loadAd();
  appLifecycleReactor = AppLifecycleReactor(appOpenAdManager: appOpenAdManager);
  appLifecycleReactor.listenToAppStateChanges();

  initHive();
}
```

Рисунок 4.5 – Ініціалізація бібліотек.

На рисунку 4.6. можно побачити вигляд нового головного меню додатка. Використовували для теста Pixel 5 31 API тому можно побачити як воно виглядає зараз.

Рисунок 4.6 – Вигляд нової версії додатка.

Нейморфний дизайн – це відносно нова тенденція дизайну, яка набула популярності у розробці мобільних додатків. Це стиль дизайну, який поєднує елементи скевоморфизм та плоского дизайну, створюючи м’який тривимірний вигляд і відчуття, що імітує зовнішній вигляд м’якого формованого пластику.

Ключові особливості дизайну Neuromorphic включають м’які, закруглені краї: Neumorphic дизайн зазвичай має м’які, закруглені краї, які

створюють відчуття глибини та об’ємності. М’які, ледь помітні тіні: у дизайні використовуються тонкі тіні, щоб створити відчуття глибини та підкреслити краї елементів. Мінімалістичний дизайн: Neumorphic дизайн, як правило, мінімалістичний за своєю природою, зосереджений на чистих лініях і простій колірній палітрі.

Зосередьтеся на взаємодії з користувачем: дизайн призначений для створення більш інтуїтивно зрозумілого та зручного досвіду з акцентом на полегшення використання кнопок та інших інтерактивних елементів.

Деякі з переваг дизайну Neumorphic для розробки мобільних додатків включають:

– покращена взаємодія з користувачем: м’які заокруглені краї та тонкі тіні дизайну Neumorphic можуть полегшити користувачам навігацію та взаємодію з програмою;

– унікальний і привабливий: стиль дизайну все ще відносно новий, тому він може допомогти додаткам виділитися на переповненому ринку;

– гнучкість: Neumorphic дизайн можна використовувати в широкому діапазоні типів програм і галузей, що робить його універсальним варіантом для розробників.

Однак важливо зазначити, що Neumorphic дизайн все ще є відносно новою тенденцією, і він може не підійти для кожної програми. Перш ніж прийняти рішення про використання Neumorphic, розробникам слід уважно розглянути взаємодію з користувачем і специфічні вимоги до своєї програми.

Тому ми використовували цей патерн дизайну. На рисунку 4.7 можно побачити як воно виглядає у коді.

```dart
@override
Widget build(BuildContext context) {
  return BlocBuilder<GeneratedTextBloc, int>(builder: (context, count) {
    return Center(
      child: Column(
        children: [
          SizedBox(
            height: 6.h,
          ),  // SizedBox
          Container(
            decoration: AppStyle.containerStyle(context: context),
            child: Row(
              crossAxisAlignment: CrossAxisAlignment.center,
              mainAxisAlignment: MainAxisAlignment.spaceBetween,
              children: [
                Padding(
                  padding: const EdgeInsets.all(8.0),
                  child: Text(
                    AppString.youHave + count.toString() + AppString.points,
                    style: AppStyle.mediumTextStyleOther(context: context),
                  ),  // Text
                ),  // Padding
                Padding(
                  padding: const EdgeInsets.all(8.0),
                  child: NeumorphicButton(
                    style: AppStyle.appBarButtonStyle(context: context),
                    child: Icon(
                      Icons.add,
                      size: 25,
                      color: NeumorphicTheme.baseColor(context),
                    ),  // Icon
                    onPressed: () => {context.read<GeneratedTextBloc>().add(AddPointsEvent())},
                  ),  // NeumorphicButton
                ),  // Padding
              ],
            ),  // Row
          ),  // Container
```

Рисунок 4.7 – Вигляд нової версії додатка у коді.

Далі як використовується BloC для навігації у додатку. Навігація є важливим аспектом розробки мобільних програм, і Flutter надає кілька інструментів і віджетів для реалізації навігації в програмі. Ось деякі з ключових функцій навігації у Flutter.

Стек навігації: Flutter використовує стек навігації для керування різними екранами в програмі. Стек – це структура даних «останній прийшов, перший вийшов» (LIFO), яка відстежує відвідані екрани. Коли користувачі переміщуються програмою, нові екрани додаються до верхньої частини стека, а коли вони натискають кнопку «Назад», верхній екран віддаляється зі стеку.

Віджети Material App і Navigator: Віджет Material App використовується для визначення загальної структури програми, включаючи панель програми, тему та початковий маршрут. Віджет Navigator використовується для керування стеком навігації та перемикання між екранами.

Іменовані маршрути: у Flutter кожен екран пов’язаний з унікальним маршрутом, до якого можна отримати доступ за назвою. Іменовані маршрути спрощують навігацію між екранами за допомогою віджета Navigator.

Методи push і pop: віджет Navigator надає методи push і pop для додавання та видалення екранів із стеку навігації. Метод push використовується для додавання нового екрана на вершину стека, а метод pop використовується для видалення верхнього екрана зі стеку [2].

Аргументи маршруту: у Flutter маршрутам можна передавати аргументи, які можна використовувати для налаштування вмісту екрана. Це може бути корисним для передачі даних між екранами або налаштування поведінки екрана на основі введення користувача.

Панель навігації: Flutter надає віджет Navigation Drawer, який можна використовувати для реалізації меню навігації, яке ковзає з боку екрана.

Загалом, навігація у Flutter є гнучкою та настроюванню, і існує багато різних способів реалізації навігації в програмі. Під час розробки навігаційної системи розробники повинні ретельно враховувати досвід користувача та особливі вимоги своєї програми. Можно побачити на рисунку 4.8.

```dart
abstract class MainPageEvent {}

class MainPageToSettings extends MainPageEvent {}

class MainPageToGeneratedTextView extends MainPageEvent {}

class MainPageToStartGeneration extends MainPageEvent {
  MainPageToStartGeneration({required this.story});
  final String story;
}

class MainPageToStoryList extends MainPageEvent {}

class MainPageToFavoriteList extends MainPageEvent {}

class MainPageBloc extends Bloc<MainPageEvent, Widget> {
  MainPageBloc() : super(const GeneratedTextFragment()) {
    on<MainPageToSettings>((event, emit) => emit(const SettingsFragment()));
    on<MainPageToGeneratedTextView>((event, emit) => emit(const GeneratedTextFragment()));
    on<MainPageToStartGeneration>((event, emit) => emit(CreateTextFragment(startingText: event.story)));
    on<MainPageToStoryList>((event,emit)=> emit(const StoryListFragment()));
    on<MainPageToFavoriteList>((event,emit)=> emit(const FavoriteListFragment()));

  }
```

Рисунок 4.8 – Навігація у додатку.

Також ми розробили додаткові функції додатку такі як настройки, список робіт. Їх можна побачити на рисунках 4.9 та 4.10.

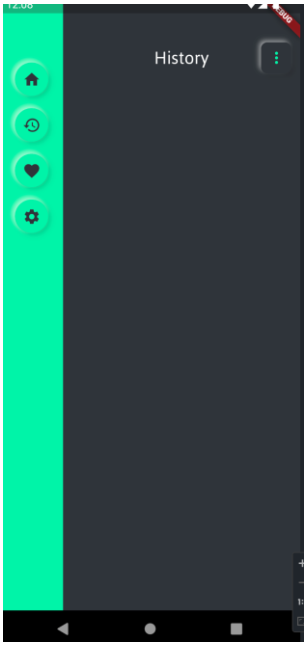

Рисунок 4.9 – History страница додатка зі списком того що вже зробив користувач.

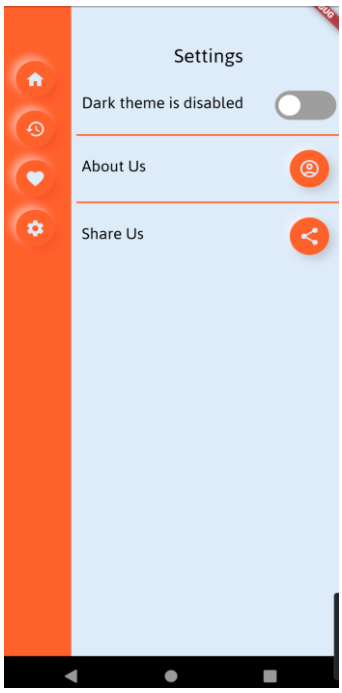

Рисунок 4.10 – Settings

Тут можна поділитися або зайти на сайт. Також можно змінити тему на темну (рисунок 4.11).

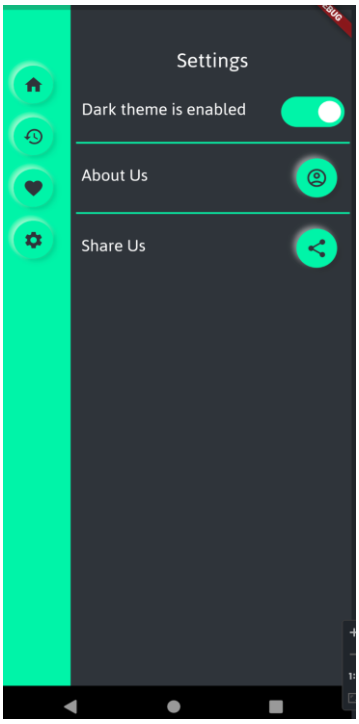

Рисунок 4.11 – Settings якщо вибрати темну тему.

Також ми працювали над базої даних за допомогою HIVE. Структуру можна подивитися на рисунку 4.12.

```
@HiveType(typeId: 0)
class StoryModel extends HiveObject{

  @HiveField(0)
  late String title;

  @HiveField(1)
  late bool favorites;

  @HiveField(2)
  late String story;

  @HiveField(3)
  late bool pined;

}
```

Рисунок 4.12 – Структура об'єкта бази даних.

Також розробили стор для того щоб було швидко та якісно з нею працювати. На рисунку 4.13.

Бази даних відіграють вирішальну роль у розробці мобільних програм, оскільки вони дозволяють програмам швидко й ефективно зберігати й отримувати дані. Ось деякі з ключових міркувань під час роботи з базами даних у мобільній розробці.

Вибір бази даних. Існує багато різних типів баз даних, доступних для мобільної розробки, включаючи локальні бази даних, такі як SQLite і Realm, а також хмарні бази даних, такі як Firebase і Amazon Web Services. Вибір бази даних залежатиме від таких факторів, як тип і розмір даних, що зберігаються, вимог до продуктивності програми та бюджету для розміщення бази даних.

Архітектура бази даних: розробляючи базу даних для мобільного додатка, важливо враховувати загальну архітектуру додатка та те, як база даних висуватиметься в цю архітектуру. Наприклад, програма використовуватиме централізовану базу даних чи розподілену базу даних? Чи потрібен додаток гібридний підхід, який поєднує локальне та хмарне сховище?

```dart
abstract class StoryStore{

  static Future<void> createStory(String title, bool favorites, String story,bool pined) async {
    var box = await Hive.openBox('myStoryList');
    box.add(StoryModel()
      ..title = title
      ..favorites = favorites
      ..story = story
      ..pined = pined);

    debugPrint(box.values.toString());
  }

  static Future<List<StoryModel>> getStoryList() async {
    var box = await Hive.openBox('myStoryList');
    List<StoryModel> list =[];
    for(int i =0; i< box.length; i++){
      list.add(box.getAt(i));
      //list[i].favorites = false;
      //list[i].save();
    }
    return list;
  }
  static Future<void> clearFavoriteList() async {
    var box = await Hive.openBox('myStoryList');
    List<StoryModel> list =[];
    for(int i =0; i< box.length; i++){
      list.add(box.getAt(i));
      list[i].favorites = false;
      list[i].save();
    }

  }

  static Future<void> clearStoryList() async {
    var box = await Hive.openBox('myStoryList');
    box.clear();
  }

  static void deleteStory(StoryModel dayModel){
    dayModel.delete();
  }
```

Рисунок 4.13 – Стор для роботи з БД.

Моделювання даних. Моделювання даних – це процес визначення структури даних, які зберігатимуться в базі даних. Це передбачає визначення таблиць, полів, зв’язків між таблицями та інших аспектів схеми даних. Хороша практика моделювання даних є важливою для забезпечення організованості та ефективності бази даних.

Доступ до бази даних: щоб отримати доступ до бази даних із програми, розробникам потрібно буде використовувати рівень доступу до бази даних, який забезпечує інтерфейс між програмою та базою даних. Цей рівень повинен обробляти такі завдання, як відкриття та закриття з’єднань з базою даних, виконання SQL-запитів і керування транзакціями бази даних.

Синхронізація даних: якщо програма використовує хмарну базу даних, важливо враховувати, як синхронізація даних буде оброблятися між локальною програмою та хмарною базою даних. Це може передбачати використання таких технологій, як реплікація або протоколи синхронізації, щоб забезпечити узгодженість даних на всіх пристроях.

Безпека: безпека є критичною проблемою під час роботи з базами даних, оскільки конфіденційні дані можуть зберігатися в базі даних. Розробники повинні вжити заходів для забезпечення захисту бази даних від несанкціонованого доступу, витоку даних та інших загроз безпеці.

Загалом, робота з базами даних у мобільній розробці вимагає ретельного розгляду широкого спектру факторів, від вибору бази даних до розробки схеми бази даних і впровадження функцій доступу до бази даних і синхронізації. Застосовуючи продуманий і стратегічний підхід до проектування та керування базами даних, розробники можуть переконатися, що їхні мобільні програми надійні, ефективні та безпечні.

Головною з функцій додатку є генерування тексту за допомогою GPT- 3. На рисунку 4.14 можна побачити як користувачу потрібно вписати перше речення. Наразі тільки англійська мова.

Рисунок 4.14 – Меню для користувача.

Після того як користувач вписав речення можно почати генерацію яку можна контролювати за допомогою кнопок (рисунок 4.15).

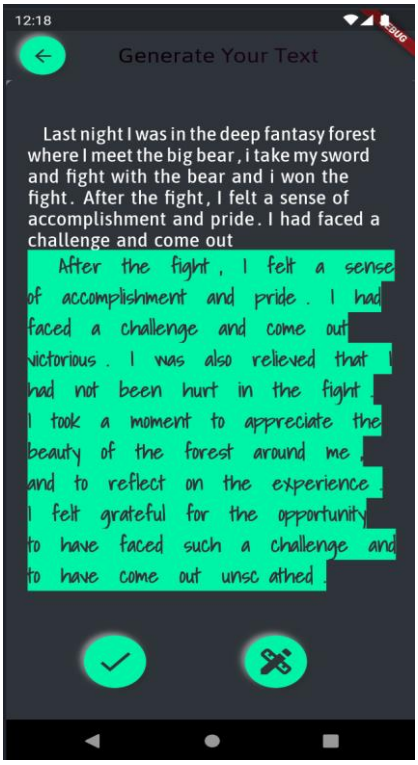

Рисунок 4.15 – Працююча генерація історії.

Отже ми зробили додаток генерації тексту на мобільні пристрої за допомогою Flutter та GPT-3.

ВИСНОВКИ

У даній роботі було розроблено програмний додаток на мобільні пристрої, та проведено дослідження ефективності різних моделей у роботі додатка для генерації тексту. для цього проведено аналіз існуючої предметної галузі розробки на мобільні пристрої. В ході проведеного аналізу було виявлено як правильно розробляти на Android/IOS. Також було проведено аналіз ймовірних методів рішень та вирішено обрати рішення використовувати нейронну мережу, деяку мовну модель gpt-3 від OpenAI. Був проведен аналіз цієї мовної моделі. Робота з мовними моделями дає нам змогу виконувати ті речі які не можуть зробити інші методи. Коли ми кажемо про різноманітність мовна модель це кращій вибір. У нейронної мережі не тільки великий потенціал, але й насамперед доцільність використання. Виконана робота з побудови роботи з gpt-3 та її використовування у додатку. Тобто зроблена клієнтська частина для роботи з цієї мовною моделью та побудова інтерфейс у стилі Neuromorphic користувача для роботи користувача з цією нейронною мережею (мовною моделлю). Було дуже важливо зробити так щоб інтерфейс користувача був user-friendly, та користувач не бачив ніяких багів, лагів, зависань і тому подібне. Тому даний прототип програмного додатку на мобільні пристрої був розроблений з урахуванням подальшого потенціалу. Отримані дані дали змогу створити цей прототип конкурентоспроможним та маючи змогу виконати надану ціль додатку:

− змогу зацікавити користувача;

− змогу генерувати нескінченну історію;

− інтерактивність з користувачем;

− змогу дослідити мовну модель;

− змогу використовуватись у суспільстві.

Також були приведені технічні вимогу до користування мобільним додатком. Спочатку було проведено тестування на приладах що досить

поширені, десь 98,8% приладів світу, але реалізація показала що не тільки технічні характеристики для роботи моделі а ще і недолік у використанні різних бібліотек так як велика кількість бібліотек були змінені та деприкейтнуті так як ми замінили GPT-2 на GPT-3. Потім ми почали використовувати більш нові версії SDK, але це зменшило кількість приладів у світі які мають змогу запустити цей мобільний додаток. Ми зупинилися на SDK 29, не тільки через недолік бібліотек але й тому що додаток має великі технічні характеристики додаток не може бути використаний на 33% мобільних пристроїв які зараз використовують у світі. Загалом дослідили ефективність GPT-3 проти GPT-2. Та обрали найкращу.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Better Language Models and their implication. URL: https://openai.com/blog/better-language-models/ (дата звернення: 17.04.2023).

2. Documentation for Android/Ios developers URL: https://developer.android.com/docs (дата звернення: 18.04.2023).

3. Kotlin Start documentation URL: https://kotlinlang.org/docs/home.html (дата звернення: 18.04.2023).

4. Теория многоуровневых семантических сетей : учеб. пособие / М. Ф. Бондаренко, В. А. Гребенюк, Е. Б. Кайкова, В. Я. Терзиян ; М-во образования Украины, Ин-т содержания и методов обучения, Харьк. гос. техн. ун-т радиоэлектроники. – Харьков : ХТУРЭ, 1997. 76 с.

5. Гребенюк В. А. Введение в специальность «Интеллектуальные системы принятия решений» : конспект лекций / МО Украины, Харьк. техн. ун-т радиоэлектроники. Харьков : ХТУРЭ, 1998. 137 с

6. Рик Роджерс, Джон Ломбардо, «Android Разработка приложений», ЭКОМ Паблишерз, 2010 г. ISBN 978-5-9790-0113-5.

7. Коматинени С., Маклин Д., Хашими С. Google Android: программирование для мобильных устройств = Pro Android 2. - 1-е изд. - СПб.:Питер, 2011. – 735 с.

8. Дэрси, Л. Android за 24 часа. Программирование приложений под операционную систему Google / Л. Дэрси. – Рид Групп, 2011. – 1489 с.

9. DJI mobile developer API. URL: https://developer.dji.com/mobile-sdk/documentation/introduction/. (дата звернення: 01.05.2023).

10. Хашими, С. Разработка приложений для Android / С. Хашими. – М. : Бином, 2011. – 2125 с.

11. What is Machine Learning A Definition. URL: https://www.expert.ai/blog/machine-learning-definition/ (дата звернення: 18.04.2023).

12. Neural networks basics. Elements of AI. URL: https://course.elementsofai.com/5/1 (дата звернення: 18.04.2021).

ДОДАТОК А

Лістинг коду

```
import 'dart:developer';

import
'package:ai_gpt_story_generator/main_page/fragments/story_
list_bloc/story_list_bloc.dart';
import
'package:ai_gpt_story_generator/main_page/fragments/story_
list_fragment/pinde_list_bloc/pined_list_bloc.dart';
import
'package:ai_gpt_story_generator/main_page/main_page_bloc.d
art';
import
'package:ai_gpt_story_generator/model/story_model.dart';
import
'package:ai_gpt_story_generator/model/story_store.dart';
import
'package:ai_gpt_story_generator/utils/strings.dart';
import 'package:ai_gpt_story_generator/utils/styles.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import
'package:flutter_neumorphic/flutter_neumorphic.dart';
import 'package:show_up_animation/show_up_animation.dart';
import 'package:sizer/sizer.dart';

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

```
 @override
 Widget build(BuildContext context) {
   return Padding(
     padding: const EdgeInsets.all(2.0),
     child: Divider(color:
NeumorphicTheme.accentColor(context), thickness: 2),
   );
 }
}
class PinedListItem extends StatefulWidget {
 const PinedListItem({Key? key, required this.storyModel})
: super(key: key);
 final StoryModel storyModel;
 @override
 State<PinedListItem> createState() =>
_PinedListItemState();
}

class _PinedListItemState extends State<PinedListItem> {
 @override
 Widget build(BuildContext context) {
   return Padding(
     padding: const EdgeInsets.all(15),
     child: Row(
       children: [
         GestureDetector(
           onTap: () => {
             context.read<MainPageBloc>().add(
```

```
                MainPageToStartGeneration(story:
widget.storyModel.story))
            },
            child: Neumorphic(
              style: AppStyle.pinedItemStyle(context:
context),
              child: SizedBox(
                width: 20.w,
                height: 10.h,
                child: Center(
                  child: Text(
                    widget.storyModel.title,
                    style:
                    AppStyle.pinedListTextStyle(context:
context)                    ),
                ),
              ),
            ),
          ),
        ],
      ),
    );
 }
}

class StoryListItem extends StatefulWidget {
 const StoryListItem(
     {Key? key, required this.storyModel, required
this.voidCallback})
```

```
      : super(key: key);
 final StoryModel storyModel;
 final VoidCallback voidCallback;
 @override
 State<StoryListItem> createState() =>
_StoryListItemState();
}

class _StoryListItemState extends State<StoryListItem> {
 @override
 Widget build(BuildContext context) {
   return Padding(
     padding: const EdgeInsets.only(left: 15, bottom: 15),
     child: Row(
       children: [
         GestureDetector(
           onTap: () => {
             context.read<MainPageBloc>().add(
                 MainPageToStartGeneration(story:
widget.storyModel.story))
           },
           child: Neumorphic(
             style: AppStyle.listItemStyle(context:
context),
             child: SizedBox(
               width: 63.w,
               height: 10.h,
               child: Padding(
```

```
                padding: const EdgeInsets.only(left:
18.0),
                child: Column(
                  crossAxisAlignment:
CrossAxisAlignment.start,
                  children: [
                    Flexible(
                      flex: 1,
                      child: Text(
                        widget.storyModel.title,
                        style:

AppStyle.itemListTitleTextStyle(context: context),
                      ),
                    ),
                    Flexible(
                      flex: 2,
                      child: Text(

«${widget.storyModel.story.substring(0, 30)}...»,
                        style:
AppStyle.itemListTextStyle(context: context),
                        maxLines: 2,
                        overflow: TextOverflow.fade,
                      ),
                    ),
                  ],
                ),
              ),
```

```
            ),
          ),
        ),
        Padding(
          padding: const EdgeInsets.all(10.0),
          child: GestureDetector(
            onTap: widget.voidCallback,
            child: Icon(
              Icons.favorite,
              size: 6.w,
              color: !widget.storyModel.favorites
                  ?
NeumorphicTheme.defaultTextColor(context)
                  : NeumorphicTheme.accentColor(context),
            ),
          ),
        )
      ],
    ),
  );
 }
}

class DismissibleWidget extends StatefulWidget {
 const DismissibleWidget(
     {Key? key,
     required this.child,
     required this.keyItem,
     required this.dayModel})
```

```
      : super(key: key);
  final Widget child;
  final GlobalKey keyItem;
  final StoryModel dayModel;
  @override
  State<DismissibleWidget> createState() =>
_DismissibleWidgetState();
}

class _DismissibleWidgetState extends
State<DismissibleWidget> {
  double widt = 0;
  @override
  Widget build(BuildContext context) {
    return Dismissible(
      key: widget.keyItem,
      onUpdate: (direction) {
        if (direction.direction ==
DismissDirection.startToEnd) {
          setState(() {
            widt = direction.progress * 14.w;
          });
        } else if (direction.direction ==
DismissDirection.endToStart) {
          setState(() {
            widt = direction.progress * 14.w;
          });
        }
      },
```

```
      background: Align(
          alignment: Alignment.centerLeft,
          child: AnimatedPadding(
              duration: const Duration(milliseconds: 300),
              padding: EdgeInsets.only(left: widt),
              child: CustomCircularProgressBar(
                progressValue: widt,
                iconValue: Icons.push_pin,
                isPined: widget.dayModel.pined,
              ))),
      secondaryBackground: Align(
          alignment: Alignment.centerRight,
          child: AnimatedPadding(
            duration: const Duration(milliseconds: 300),
            padding: EdgeInsets.only(right: widt),
            child: CustomCircularProgressBar(
              progressValue: widt,
              iconValue: Icons.delete,
              isPined: widget.dayModel.pined,
            ),
          )),
      onDismissed: (direction) => {
        if (direction == DismissDirection.endToStart)
          {
            context
                .read<StoryListBloc>()
                .add(DeleteStory(storyModel:
widget.dayModel)),
```

```
context.read<PinedListBloc>().add(InitialPinedEvent())
          }
        else if (direction == DismissDirection.startToEnd)
          {
            context
                .read<StoryListBloc>()
                .add(PinStory(storyModel:
widget.dayModel)),

context.read<PinedListBloc>().add(InitialPinedEvent())
          }
      },
      confirmDismiss: (DismissDirection direction) async {
        if (direction == DismissDirection.startToEnd) {
          return true;
        } else {
          return true;
        }
      },
      dismissThresholds: const {
        DismissDirection.startToEnd: 0.5,
        DismissDirection.endToStart: 0.5
      },
      child: widget.child,
    );
  }
}
```

```
class CustomCircularProgressBar extends StatefulWidget {
 const CustomCircularProgressBar(
     {Key? key,
     required this.progressValue,
     required this.iconValue,
     required this.isPined})
     : super(key: key);
 final bool isPined;
 final double progressValue;
 final IconData iconValue;
 @override
 State<CustomCircularProgressBar> createState() =>
     _CustomCircularProgressBarState();
}

class _CustomCircularProgressBarState extends
State<CustomCircularProgressBar> {
 @override
 Widget build(BuildContext context) {
   return SizedBox(
     width: 10.w,
     height: 10.w,
     child: Stack(
       children: [
         Positioned(
             left: 0,
             right: 0,
             bottom: 0,
             top: 0,
```

```
            child: widget.iconValue == Icons.push_pin
                ? CircularProgressIndicator(
                    value: widget.progressValue / 27.3,
                    strokeWidth: 1.w,
                    color: widget.isPined == true
                        ?
NeumorphicTheme.accentColor(context)
                        :
NeumorphicTheme.variantColor(context),
                  )
                : CircularProgressIndicator(
                    value: widget.progressValue / 27.3,
                    strokeWidth: 1.w,
                    color:
NeumorphicTheme.accentColor(context),
                  )),
          Positioned(
              left: 0,
              right: 0,
              bottom: 0,
              top: 0,
              child: widget.iconValue == Icons.push_pin
                  ? Icon(
                      widget.iconValue,
                      size: 8.w,
                      color: widget.isPined
                          ?
NeumorphicTheme.accentColor(context)
```

```
                          :
NeumorphicTheme.defaultTextColor(context),
                    )
                  : Icon(
                      widget.iconValue,
                      size: 8.w,
                      color:
NeumorphicTheme.accentColor(context),
                    ))
        ],
      ),
    );
  }
}

class DropDownMenuWidget extends StatefulWidget {
  const DropDownMenuWidget({Key? key, required
this.blocContext})
      : super(key: key);
  final BuildContext blocContext;
  @override
  State<DropDownMenuWidget> createState() =>
_DropDownMenuWidgetState();
}

class _DropDownMenuWidgetState extends
State<DropDownMenuWidget> {
  String? value;
```

```
@override
Widget build(BuildContext context) {
  return NeumorphicFloatingActionButton(
    onPressed: () => {},
    child: PopupMenuButton(
      color: NeumorphicTheme.baseColor(context),
      enableFeedback: false,
      icon: Icon(
        Icons.more_vert,
        color: NeumorphicTheme.accentColor(context),
      ),
      position: PopupMenuPosition.under,
      shape: const RoundedRectangleBorder(),
      tooltip: "",
      itemBuilder: (context) => [
        PopupMenuItem<int>(
            value: 0,
            child: Row(
              children: [
                const Icon(Icons.delete_forever),
                Text(
                  AppString.storyDelete,
                  style:
AppStyle.popupMenuItemTextStyle(context: context),
                )
              ],
            )),
        PopupMenuItem<int>(
            value: 1,
```

```
              child: Row(
                children: [
                  const Icon(Icons.heart_broken),
                  Text(
                    AppString.favoriteDelete,
                    style:
AppStyle.popupMenuItemTextStyle(context: context),
                  )
                ],
              )),
        ],
        onSelected: (item) =>
selectedItem(widget.blocContext, item),
      ),
    );
 }

 void showCustomDialog(BuildContext blocContext, item) =>
showDialog(
      context: blocContext,
      builder: (BuildContext context) {
        return Dialog(
            backgroundColor: Colors.transparent,
            child: Neumorphic(
              style: AppStyle.listItemStyle(context:
context),
              child: Container(
                alignment: Alignment.center,
                width: 15.w,
```

```
                height: 30.h,
                decoration:
AppStyle.containerStyle(context: context),
                child: Column(
                  crossAxisAlignment:
CrossAxisAlignment.center,
                  mainAxisAlignment:
MainAxisAlignment.spaceEvenly,
                  children: [
                    Text(
                      item == 0
                          ? AppString.storyDelete
                          : AppString.favoriteDelete,
                      style:
AppStyle.titleTextStyle(context: context),
                    ),
                    Icon(
                      item == 0 ? Icons.delete_forever :
Icons.heart_broken,
                      color:
NeumorphicTheme.baseColor(context),
                      size: 18.w,
                    ),
                    Text(
                      item == 0
                          ? AppString.storyDeleteAccept
                          :
AppString.favouriteDeleteAccept,
```

```
                    style:
AppStyle.itemListTextStyle(context: context),
                  ),
                  Row(
                    mainAxisAlignment:
MainAxisAlignment.center,
                    children: [
                      item == 0
                          ? Padding(
                              padding:
EdgeInsets.all(4.w),
                              child:
NeumorphicFloatingActionButton(
                                child: const
Icon(Icons.check),
                                onPressed: () =>
setState(() {
                                  blocContext

.read<StoryListBloc>()

.add(DeleteAllStories());

Navigator.of(context).pop();
                                }),
                              ),
                            )
                          : Padding(
```

```
                                padding:
EdgeInsets.all(4.w),
                                child:
NeumorphicFloatingActionButton(
                                  child: const
Icon(Icons.check),
                                  onPressed: () =>
setState(() {
                                    blocContext

.read<StoryListBloc>()

.add(DeleteAllFavourite());

Navigator.of(context).pop();
                                  }),
                                ),
                              ),
                        Padding(
                          padding: EdgeInsets.all(4.w),
                          child:
NeumorphicFloatingActionButton(
                            child: const Icon(Icons.clear),
                            onPressed: () =>
{Navigator.of(context).pop()},
                          ),
                        ),
                      ],
                    ),
```

```
                ],
              ),
            ),
          ));
    });
 selectedItem(BuildContext context, item) {
   switch (item) {
     case 0:
       showCustomDialog(context, item);
       break;
     case 1:
       showCustomDialog(context, item);
       break;
   }
 }
}
class CustomSnackBar extends StatefulWidget {
 const CustomSnackBar({Key? key}) : super(key: key);

 @override
 State<CustomSnackBar> createState() =>
_CustomSnackBarState();
}

class _CustomSnackBarState extends State<CustomSnackBar> {
 @override
 Widget build(BuildContext context) {
   return const Placeholder();
 } }
```

ДОДАТОК Б

Відомість кваліфікаційної роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	77 с.
	Інші документи	
2.	Презентаційні матеріали	__ плакатів

.	.	Прізвище та ініціали.	Підп.	Дата			
Розробив		Пономаренко Д.Р.			Дослідження ефективності нейронних мереж генерування тексту для мобільної текстової quest-гри	Шифр групи	Код напр./спец.
Перевірив		Смеляков К.С.				СШІм-21-1	122
Н.контр.		Малєєва І.А.				ХНУРЕ кафедра ШІ	
Затв.		Філатов В.О.					