НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий інститут прикладного системного аналізу

Кафедра штучного інтелекту

До захисту допущено:

В. о. завідувачки кафедри

_____ Ірина ДЖИГИРЕЙ

«___»_____20__ р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системи і методи штучного інтелекту»

спеціальності 122 «Комп'ютерні науки»

на тему: «Система підтримки прийняття рішень для підбірки компонентів FPV дронів з використанням трансформерних моделей»

Виконав:
студент IV курсу, групи КІ-13
Пшеничний Дмитро Володимирович _____

Керівник:
асистент кафедри штучного інтелекту
Кухарєв Сергій Олександрович _____

Консультант з економічного розділу:
доцент кафедри економічної кібернетики, к.е.н., доцент,
Рощина Надія Василівна _____

Консультант з нормоконтролю:
фахівець першої категорії кафедри штучного інтелекту, к.т.н., доцент,
Комариста Богдана Миколаївна _____

Рецензент:
доцент кафедри системного проєктування, к.т.н.,
Безносик Олександр Юрійович _____

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів
без відповідних посилань.
Студент _____

Київ – 2025 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Навчально-науковий інститут прикладного системного аналізу
Кафедра штучного інтелекту

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 122 «Комп'ютерні науки»

Освітньо-професійна програма «Системи і методи штучного інтелекту»

ЗАТВЕРДЖУЮ
В. о. завідувачки кафедри
_____ Ірина ДЖИГИРЕЙ
«15» січня 2025 р.

ЗАВДАННЯ

на дипломну роботу студенту

Пшеничний Дмитро Володимирович

1. Тема роботи «Система підтримки прийняття рішень для підбірки компонентів FPV дронів з використанням трансформерних моделей», керівник роботи асистент кафедри штучного інтелекту Кухарєв Сергій Олександрович, затверджені наказом по НН ІПСА від «26»__травня__ 2025 р. №1759-с

2. Термін подання студентом роботи «09» червня 2025 року.

3. Вихідні дані до роботи є корпус текстових відгуків клієнтів про компоненти FPV-систем, підготовлений у вигляді токенізованих наборів для навчання та валідації трансформерних моделей.

4. Зміст роботи охоплює постановку задачі аналізу тональності, огляд літератури та методів трансформерного навчання, опис архітектури веб-системи з Flask і SQLAlchemy, реалізацію модуля аналізу настроїв, тестування та узагальнення результатів.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо) представлено UML- і ER-діаграмами, скріншотами веб-інтерфейсу, блок-схемами алгоритмів і графіками експериментальних показників.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Рощина Надія Василівна, доцент, к. е. н.		

7. Дата видачі завдання «03» лютого 2025 року.

КАЛЕНДАРНИЙ ПЛАН

з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Вивчення та аналіз задачі	14.04.2025 – 30.04.2025	виконано
2	Розробка архітектури та загальної структури системи	01.05.2025 – 10.05.2025	виконано
3	Розробка структур окремих підсистем	11.05.2025 – 20.05.2025	виконано
4	Програмна реалізація системи	21.05.2025 – 28.05.2025	виконано
5	Оформлення пояснювальної записки	29.05.2025 – 04.06.2025	виконано

Студент _____ (підпис) _____ (ім’я, ПРІЗВИЩЕ)

Керівник роботи _____ (підпис) _____ (ім’я, ПРІЗВИЩЕ)

РЕФЕРАТ

Дипломна робота містить 94 сторінок, 25 рисунків, 8 таблиць, 19 посилань та 1 додаток.

МАШИННЕ НАВЧАННЯ, ОБРОБКА ПРИРОДНОЇ МОВИ (NLP), ТРАНСФОРМЕР, FPV, АНАЛІЗ НАСТРОЇВ.

Об’єктом дослідження є інтегрована інформаційна система підтримки прийняття рішень для підбору компонентів FPV-дронів, що поєднує веб-інтерфейс, базу даних і модуль аналізу настроїв.

Предметом дослідження є методи класифікації тональності текстових відгуків на основі трансформерних моделей і архітектурні рішення веб-застосунку на базі Flask і SQLAlchemy.

Метою роботи було розробити та апробувати програмне забезпечення, здатне автоматично обробляти клієнтські відгуки про FPV-компоненти, проводити їхню тональну класифікацію та інтегрувати результати аналізу в зручний веб-інтерфейс для підтримки управлінських рішень. Проаналізовано підходи до обробки текстових відгуків, формування ознак в контексті аналізу настроїв клієнтів. Досліджено методи класифікації та виявлення контекстуальних зв'язків у текстових відгуках клієнтів, що дозволило обґрунтувати ефективність обраної трансформерної архітектури. Встановлено, що застосування даної моделі забезпечує високу точність аналізу емоційного забарвлення коментарів щодо FPV-компонентів при оптимальних вимогах до обчислювальних ресурсів та швидкодії обробки великих текстових масивів. Практична цінність роботи полягає у створенні ефективного інструменту для оперативного моніторингу задоволеності клієнтів, що дозволяє бізнесу швидко виявляти проблемні аспекти продукції та приймати обґрунтовані рішення щодо покращення якості товарів і сервісу у динамічній сфері FPV-технологій.

ABSTRACT

This bachelor’s thesis comprises 94 pages, 25 figures, 8 tables, 19 references, and 1 appendix.

MACHINE LEARNING, NATURAL LANGUAGE PROCESSING (NLP), TRANSFORMER, FPV, SENTIMENT ANALYSIS.

The object of the study is an integrated decision-support system for selecting FPV-drone components, which combines a web front-end, a relational database, and a sentiment-analysis module.

The subject of the research is the methods for classifying the sentiment of textual customer reviews using transformer models and the architectural design of the web application built on Flask and SQLAlchemy.

The aim of the work was to develop and validate software capable of automatically processing customer feedback on FPV components, performing sentiment classification, and integrating the analysis results into a user-friendly web interface to support managerial decision-making. Approaches to processing text reviews and feature formation in the context of customer sentiment analysis were analyzed. Methods of classification and detection of contextual relationships in customer text reviews were investigated, which allowed to substantiate the effectiveness of the chosen transformer architecture. It was established that the application of this model ensures high accuracy of emotional tone analysis in comments regarding FPV components with optimal requirements for computational resources and processing speed of large text arrays. The practical value of the work lies in creating an effective tool for operational monitoring of customer satisfaction, which enables businesses to quickly identify problematic aspects of products and make informed decisions regarding the improvement of product and service quality in the dynamic field of FPV technologies.

ЗМІСТ

ВСТУП 7
РОЗДІЛ 1 ОГЛЯД МЕТОДІВ ТРАНСФОРМЕРНОГО NLP 9
1.1 Постановка прикладної задачі автоматизованого аналізу настроїв у відгуках 9
1.2 Огляд сучасних підходів, методів та алгоритмів вирішення задачі 11
1.3 Аналіз програмних засобів для реалізації системи 14
Висновки до розділу 1 18
РОЗДІЛ 2 МЕТОДИ ТА АЛГОРИТМИ ТРАНСФОРМЕРНОГО АНАЛІЗУ ТОНАЛЬНОСТІ ТЕКСТОВИХ ВІДГУКІВ У СИСТЕМІ ПІДТРИМКИ ПРИЙНЯТТЯ РІШЕНЬ ПРИ ПІДБОРІ КОМПОНЕНТІВ FPV-ДРОНІВ 19
2.1 Застосування трансформерів для обробки природної мови 20
2.2 Використання методів аналізу настроїв 24
2.3 Опис алгоритмів та налаштувань моделі (train.py) 28
2.4 Порівняльний аналіз з існуючими рішеннями 30
Висновки до розділу 2 33
РОЗДІЛ 3 ПРОГРАМНА РЕАЛІЗАЦІЯ 35
3.1 Загальна архітектура системи 35
3.2 Структура бази даних 39
3.3 Опис основних модулів застосунку (app.py) 40
3.3.1 Реалізація веб-інтерфейсу (Flask) 41
3.3.2 Модуль роботи з користувачами (реєстрація, авторизація, управління сесіями) 41
3.3.3 Модуль управління товарами та замовленнями 42
3.3.4 Модуль аналізу настроїв з використанням навченої моделі 43
3.4 Розробка інтерфейсу користувача (фронтенду) 44

Висновки до розділу 3 47
РОЗДІЛ 4 ІНСТАЛЯЦІЯ, ТЕСТУВАННЯ ТА АПРОБАЦІЯ СИСТЕМИ 48
4.1 Інсталяція програмного забезпечення та вимоги до обчислювальної техніки 48
4.2 Демонстрація функціоналу системи та сценарії роботи 49
4.3 Результати тестування та результати обчислювальних експериментів 55
Висновки до розділу 4 57
РОЗДІЛ 5 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ 58
5.1 Постановка задачі проектування 58
5.2 Обґрунтування функцій програмного продукту 59
5.3 Обґрунтування системи параметрів програмного продукту 63
5.4 Аналіз експертного оцінювання параметрів 66
5.5 Аналіз рівня якості варіантів реалізації функцій 70
5.6 Економічний аналіз варіантів розробки ПП 72
5.7 Вибір кращого варіанту ПП техніко-економічного рівня 77
Висновки до розділу 5 78
ВИСНОВКИ79
ПЕРЕЛІК ПОСИЛАНЬ 81
ДОДАТОК А 83

ВСТУП

Розробка програмного забезпечення для аналізу настроїв клієнтів у відгуках про складові FPV-систем розглядається у дипломній роботі. Актуальність теми зумовлена стрімким розвитком технологій FPV (first-person view), які знаходять широке застосування не лише у сфері аерозйомки, спортивних змагань та розважальних заходів, а й у військових умовах. Зокрема, під час війни між Україною та росією FPV-технології використовуються на лінії фронту для розвідки та координації дій, що свідчить про їхню стратегічну важливість та високий потенціал у різних галузях. Зростаюча кількість користувацьких відгуків про FPV-компоненти стимулює потребу у систематичному аналізі настроїв клієнтів, що дає змогу виробникам і постачальникам оперативно реагувати на зауваження та вдосконалювати технічні характеристики продукції.

Об’єктом дослідження є інтегрована система, що поєднує веб-інтерфейс, базу даних та модуль аналізу настроїв на основі сучасних методів машинного навчання, зокрема трансформерних моделей. Система реалізована із застосуванням мови програмування Python та веб-фреймворку Flask, що забезпечує зручність адміністрування та управління даними. Особливістю розробленої системи є використання моделі DistilBERT, адаптованої для класифікації текстових даних, що дає можливість проводити автоматичну оцінку тональності відгуків клієнтів щодо окремих складових FPV, таких як камери, передавачі відеосигналу, контролери польоту та інші елементи.

Метою роботи є розробка та апробація програмного забезпечення для аналізу настроїв користувачів у відгуках про складові FPV, що дає змогу здійснювати об’єктивну оцінку задоволеності клієнтів, а також сприяє оптимізації процесів виробництва та продажу продукції. Для досягнення поставленої мети було розроблено алгоритмічний підхід, що включає аналіз текстових даних, їх токенізацію та класифікацію за допомогою навченої

моделі, а також інтеграцію результатів аналізу у веб-застосунок для зручного перегляду та управління інформацією.

Наукова новизна дослідження полягає у комплексному застосуванні методів обробки природної мови для автоматичного аналізу настроїв користувачів у специфічній галузі FPV, що дає можливість отримати детальну картину сприйняття окремих складових системи. Результати дослідження мають практичне значення для виробників та постачальників FPV-техніки, оскільки надають можливість своєчасно виявляти проблемні аспекти у продукції та впроваджувати заходи щодо підвищення якості товарів.

Апробація результатів дослідження проводилася шляхом тестування розробленого програмного забезпечення в умовах реального експлуатаційного середовища, що дозволило виявити переваги використання трансформерних моделей для аналізу настроїв у порівнянні з традиційними методами. Отримані результати свідчать про високий потенціал застосування даного підходу для оптимізації бізнес-процесів у сфері електронної комерції та виробництва FPV-компонентів.

Структура роботи передбачає послідовне викладення актуальності проблематики, методологічних підходів до аналізу текстових даних, опис розробленої програмної системи, а також результатів тестування та апробації розробленого рішення. Таким чином, дипломна робота представляє собою комплексне дослідження, спрямоване на інтеграцію сучасних технологій аналізу природної мови у практичну діяльність підприємств, що працюють у сфері FPV.[1]

[1] Тут і нижче використані такі інструменти штучного інтелекту з генеративним штучним інтелектом як ChatGPT, Claude (Anthropic) та Grammarly, виключно для корегування та редагування тексту, створеного автором цієї дипломної роботи, на основі автоматизованої перевірки граматики, структури та стилю, що відповідає Політиці використання штучного інтелекту для академічної діяльності в КПІ ім. Ігоря Сікорського (протокол №11 Вченої ради КПІ ім. Ігоря Сікорського від 11 грудня 2023 р.).

РОЗДІЛ 1 ОГЛЯД МЕТОДІВ ТРАНСФОРМЕРНОГО NLP

У розділі розкриваються основні аспекти постановки задачі, а також здійснюється огляд джерел, що стосуються аналізу настроїв у текстових даних і застосування FPV-технологій. Формулювання задачі обумовлено необхідністю розробки програмного забезпечення, яке здатне автоматично проводити аналіз тональності відгуків клієнтів щодо компонентів FPV-систем, що використовуються як у цивільних, так і у військових умовах. Особлива увага приділяється розгляду актуальності застосування сучасних методів машинного навчання, зокрема трансформерних моделей, для обробки природної мови.

Огляд джерел охоплює дослідження у галузі аналізу настроїв, сучасні підходи до класифікації текстових даних, а також практичні аспекти впровадження FPV-технологій. Аналіз наявних наукових робіт дає змогу визначити переваги та недоліки існуючих рішень, а також обґрунтувати вибір методів і алгоритмів, що будуть застосовані у даній роботі. Крім того, розглядаються питання інтеграції аналітичних модулів у веб-застосунки, що дає змогу створити єдину інформаційну систему для оперативного аналізу споживчого досвіду.

1.1 Постановка прикладної задачі автоматизованого аналізу настроїв у відгуках

Розвиток технологій FPV, що застосовуються як у цивільній сфері (для аерозйомки, спортивних заходів, розважальних подій), так і у військових операціях, зокрема на лінії фронту під час конфлікту між Україною та росією, обумовлює необхідність створення інноваційних аналітичних інструментів

для дослідження споживчого досвіду. Постановка прикладної задачі даної дипломної роботи полягає у розробці програмного забезпечення, яке здійснює автоматизований аналіз настроїв клієнтів на основі текстових відгуків про складові FPV-систем. Передбачене рішення має забезпечити можливість класифікації відгуків за критерієм тональності (позитивна, негативна, нейтральна), що дає змогу оперативно виявляти сильні та слабкі сторони окремих компонентів системи, а також підтримувати процес прийняття управлінських рішень щодо оптимізації якості продукції.

Підхід до вирішення задачі ґрунтується на інтеграції сучасних методів обробки природної мови та алгоритмів машинного навчання з традиційними підходами управління якістю. Застосування трансформерних моделей, таких як DistilBERT [3], забезпечує високий рівень точності при аналізі текстових даних, що особливо важливо в умовах, коли відгуки користувачів містять як суб’єктивні, так і об’єктивні оцінки роботи FPV-компонентів. Завдяки автоматизації процесу аналізу, розроблена система дає змогу знизити витрати часу на ручну обробку інформації, мінімізувати вплив людського фактору та підвищити ефективність аналізу споживчого досвіду.

Особливе значення набуває розробка даної системи з огляду на специфіку застосування FPV-технологій у військових умовах, де оперативність і точність інформації можуть впливати на ефективність прийняття рішень на лінії фронту. Аналіз настроїв користувачів дає змогу виявити проблемні аспекти у функціонуванні FPV-систем, своєчасно вносити необхідні корективи у виробничий процес та покращувати технічні характеристики компонентів. Крім того, отримані результати можуть використовуватися для прогнозування тенденцій у споживчій поведінці та визначення напрямків розвитку інноваційних технологій у сфері безпілотних систем.

Завдання включають формування ефективного алгоритмічного підходу до обробки та класифікації текстових даних, інтеграцію аналітичного модуля в єдину систему з веб-інтерфейсом і реляційною базою даних, а також

забезпечення надійності, масштабованості та зручності використання розробленого продукту. Реалізація поставленої задачі передбачає побудову модульної архітектури, де кожен компонент системи виконує чітко визначену функцію, що сприяє підвищенню загальної ефективності роботи програмного забезпечення.

Таким чином, поставлена прикладна задача спрямована на розробку інноваційного інструменту аналізу споживчого досвіду у сфері FPV-технологій, який є актуальним з огляду на зростання ролі автоматизованих систем аналізу даних у виробничих та військових процесах. Запропонований підхід дає змогу не лише оптимізувати процес обробки текстових даних, а й сприяє покращенню якості продукції та підвищенню конкурентоспроможності підприємств, що працюють у даній галузі.

1.2 Огляд сучасних підходів, методів та алгоритмів вирішення задачі

Нині існує широкий спектр підходів, методів та алгоритмів, що використовуються для аналізу настроїв текстових даних, кожен з яких має свої переваги та недоліки залежно від специфіки завдання. Дослідження з аналізу настроїв охоплюють як традиційні методи обробки природної мови, так і сучасні алгоритми машинного навчання та глибокого навчання. Розглянемо основні групи методів та алгоритмів, що застосовуються для вирішення задач аналізу настроїв, зокрема в контексті відгуків про FPV-системи.

За класичними підходами можна віднести лексикографічні методи, які базуються на використанні спеціалізованих словників (лексиконів) настроїв, таких як SentiWordNet [4] чи інші доменні словники. До переваг цих методів належить їхня простота та інтерпретованість, проте вони зазвичай не

враховують контексту та іронії, що може призводити до заниження або завищення оцінки настроїв.

До групи класичних алгоритмів машинного навчання належать такі методи, як наївний баєсівський класифікатор, метод опорних векторів (SVM) та логістична регресія. Ці алгоритми використовують традиційні підходи до вилучення ознак, наприклад, метод TF-IDF або n-грам моделі, що дає змогу отримати певну точність класифікації, але зазвичай вимагають ручного налаштування ознак та можуть бути обмежені у здатності вловлювати глибинний смисловий контекст.

Найсучасніші підходи базуються на глибоких нейронних мережах, зокрема на рекурентних нейронних мережах (RNN) та їх вдосконалених версіях, таких як LSTM (Long Short-Term Memory) і GRU (Gated Recurrent Units). Ці моделі краще справляються з послідовнісними даними та враховують контекст, проте вони часто потребують великих обчислювальних ресурсів та часу для тренування.

Найбільш потужними в останні роки стали трансформерні моделі, зокрема BERT [1], DistilBERT, RoBERTa та інші. Вони дозволяють здійснювати передобробку тексту із застосуванням механізмів уваги, що забезпечує більш глибоке розуміння контексту та дає змогу досягати високої точності в класифікації настроїв. Серед основних переваг цих моделей – висока точність, здатність враховувати контекстуальні зв’язки між словами та можливість застосування у різних доменах. Недоліками ж є значні вимоги до обчислювальних ресурсів та потреба у великій кількості тренувальних даних.

Нижче подано табл. 1.1, яка ілюструє порівняльні характеристики основних підходів до аналізу настроїв.

Таблиця 1.1 – Порівняльні характеристики

Метод/ Алгоритм	*Опис*	*Переваги*	*Недоліки*
Лексикографіч-ний аналіз	Використання словників настроїв для підрахунку позитивних/негативних слів	Простота, швидкість, інтерпретова-ність	Обмеженість контекстом, нездатність врахувати іронію та сарказм
Класичні алгоритми машинного навчання	Naive Bayes, SVM, логістична регресія з використанням TF-IDF, n-грам моделі	Легкість реалізації, швидке навчання, можливість інтерпретації результатів	Необхідність ручного підбору ознак, менш точна обробка контексту
Рекурентні нейронні мережі (RNN, LSTM)	Моделі, що працюють з послідовними даними, враховують попередні контекстуальні зв’язки	Краще врахування послідовності слів, здатність моделювати контекст	Високі вимоги до часу тренування, проблема з затуханням градієнтів
Трансформерні моделі (BERT, DistilBERT) [2]	Моделі, що використовують механізми уваги для врахування контексту в усіх позиціях слова	Висока точність, врахування контексту, універсальність	Значні обчислювальні вимоги, потреба у великому обсязі тренувальних даних

Сучасний стан досліджень у сфері аналізу настроїв свідчить про поступове зміщення інтересу до трансформерних моделей, які здатні вирішити багато з обмежень традиційних підходів. При цьому застосування таких моделей у конкретному домені FPV-технологій, де відгуки користувачів можуть містити специфічну термінологію та контекст, вимагає адаптації та оптимізації моделі для досягнення максимальної точності класифікації.

1.3 Аналіз програмних засобів для реалізації системи

Інтегровані системи, такі як QuadPartPicker Builder представлегий на рис.1.1, демонструють практичне застосування комплексного підходу до вирішення задач, пов’язаних із підбором та оптимізацією компонентів FPV-систем. Цей веб-сервіс дає змогу користувачам не лише обирати окремі компоненти для складання FPV-дронів, а й отримувати зворотний зв’язок на основі аналізу характеристик та відгуків. Застосування подібних систем підкреслює важливість інтеграції алгоритмів аналізу даних із зручними веб-інтерфейсами, що дає змогу оперативно реагувати на змінні вимоги ринку та покращувати якість продукції.

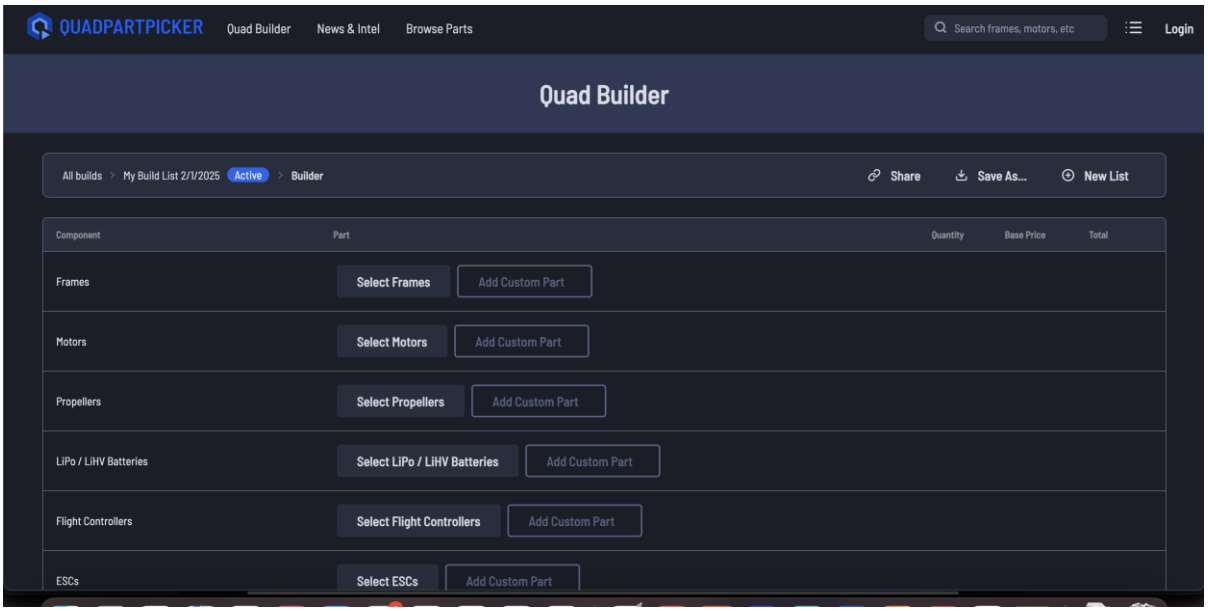

Рисунок 1.1 – QuadPartPicker Builder[2]

Вибір методу залежить від конкретних вимог до системи, зокрема від необхідності оперативної обробки даних, обсягу доступних тренувальних

[2] Джерело зображення – скріншот веб-ресурсу(14.05.2025) https://www.quadpartpicker.com/

даних та обчислювальних ресурсів. У даній роботі передбачено використання моделі DistilBERT як компромісного рішення між точністю аналізу та швидкістю обробки, що дає змогу ефективно вирішувати поставлену задачу аналізу настроїв клієнтів у відгуках про FPV-компоненти.

Таким чином, сучасні підходи до аналізу настроїв демонструють різноманіття методологій, де трансформерні моделі є найбільш перспективною стратегією для задач, що вимагають високої точності та контекстного розуміння текстових даних. Розглянута таблиця дає змогу узагальнити ключові характеристики кожного методу та обґрунтувати вибір оптимального алгоритмічного підходу для вирішення прикладної задачі даної дипломної роботи, враховуючи практичний досвід інтегрованих систем, подібних до QuadPartPicker Builder.

Для реалізації системи аналізу настроїв у відгуках про FPV-компоненти було проведено порівняльний аналіз програмних засобів, який охоплює такі категорії: мови програмування, веб-фреймворки, системи управління базами даних та середовища розробки. Нижченаведені таблиці ілюструють основні характеристики, переваги та недоліки обраних засобів порівняно з альтернативними варіантами, а також зазначають сферу застосування кожного рішення представлених в табл. (1.2-1.5).

Таблиця 1.2 – Порівняльний аналіз мов програмування

Параметри	*Обраний засіб: Python*	*Альтернативні варіанти (JavaScript, Java, C#)*
Основні характеристики	Інтерпретована, динамічна, високорівнева; багата екосистема для NLP та ML	Компільовані або частково інтерпретовані мови; суворіша типізація

Кінець таблиці 1.2

Параметри	*Обраний засіб: Python*	*Альтернативні варіанти (JavaScript, Java, C#)*
Переваги	Простий синтаксис, велика кількість бібліотек (NumPy, Pandas, scikit-learn, Hugging Face Transformers), активна спільнота	Вища продуктивність (наприклад, C++, Java), можливість оптимізації за рахунок компіляції
Недоліки	Відносно нижча швидкодія порівняно з компільованими мовами	Складність розробки NLP-рішень, менша кількість спеціалізованих бібліотек для ML
Сфера застосування	Аналіз даних, машинне навчання, веб-додатки	Системне програмування, високопродуктивні корпоративні додатки

Таблиця 1.3 – Порівняльний аналіз веб-фреймворків

Параметри	*Обраний засіб: Flask*	*Альтернативні варіанти (Django, Express, Spring)*
Основні характеристики	Мінімалістичний, модульний; орієнтований на створення RESTful API	Повноцінні фреймворки з широким набором вбудованих модулів
Переваги	Легкість налаштування, гнучкість, швидке прототипування	Багатофункціональність, вбудована підтримка адміністрування, автентифікації
Недоліки	Відсутність «з коробки» рішень для масштабованості, безпеки на високому рівні	Можлива складність налаштування для спеціалізованих завдань, «важчий» кодовий каркас
Сфера застосування	Мікросервіси, невеликі RESTful API, прототипування	Корпоративні веб-додатки, великі системи з широким функціоналом

Таблиця 1.4 – Порівняльний аналіз систем управління базами даних

Параметри	*Обраний засіб: SQLite (з ORM SQLAlchemy)*	*Альтернативні варіанти (PostgreSQL, MySQL, Oracle)*
Основні характеристики	Файлова СУБД, легка установка, мінімальні налаштування	Потужні системи, орієнтовані на великі обсяги даних, підтримка розподілених операцій
Переваги	Простота розгортання, швидке прототипування, зручна інтеграція через ORM	Висока продуктивність, розширені можливості запитів, масштабованість
Недоліки	Обмежена масштабованість, менша ефективність при високих навантаженнях	Складніша конфігурація, вимоги до адміністрування, більші витрати на обслуговування
Сфера застосування	Прототипування, невеликі проекти, розробка MVP	Корпоративні системи, великі веб-додатки, системи з високою кількістю одночасних запитів

Таблиця 1.5 – Порівняльний аналіз середовищ розробки

Параметри	*Обраний засіб: PyCharm, VSCode*	*Альтернативні варіанти (Sublime Text, Atom)*
Основні характеристики	Потужні IDE з інтегрованими засобами налагодження, підтримкою плагінів	Легкі текстові редактори з базовою підтримкою розширень
Переваги	Розширені можливості налагодження, інтеграція з контролем версій, зручний інтерфейс	Швидкий старт, низькі вимоги до ресурсів, простота налаштування
Недоліки	Вищі вимоги до системних ресурсів, частина функцій може бути ліцензована	Обмежена інтеграція з системами налагодження та контролю версій, менше спеціалізованих інструментів
Сфера застосування	Розробка складних додатків, веб-сервісів, систем із широкою логікою	Прототипування, легкі скриптові рішення, базова розробка

Аналіз програмних засобів демонструє, що обрані технології забезпечують оптимальне поєднання простоти розробки, високої точності обробки текстових даних та можливостей інтеграції з веб-сервісами. Використання Python дає змогу ефективно застосовувати численні бібліотеки для машинного навчання та обробки природної мови, що є критичним для задач аналізу настроїв. Мінімалістичний підхід Flask сприяє швидкому прототипуванню RESTful API, а використання SQLite у комбінації з ORM SQLAlchemy забезпечує зручну розробку прототипу системи. Середовища розробки, такі як PyCharm [12] та VSCode, забезпечують високу продуктивність кодування завдяки потужним інструментам налагодження та інтеграції з системами контролю версій.

Практичний досвід інтегрованих систем, подібних до QuadPartPicker Builder, підтверджує ефективність використання сучасних веб-технологій у поєднанні з потужними алгоритмами аналізу даних. Це комплексне рішення дає можливість створити зручний для кінцевого користувача інтерфейс із забезпеченням оперативної обробки та аналізу відгуків про FPV-компоненти як у цивільному, так і у військовому секторах.

Висновки до розділу 1

У розділі наведено обґрунтування прикладної задачі – автоматизованого аналізу тональності клієнтських відгуків про компоненти FPV-систем у цивільній та військовій сферах. Проведено ґрунтовний огляд сучасних підходів: від лексикографічних методів і класичних алгоритмів машинного навчання до передових трансформерних моделей. Встановлено, що саме трансформери, завдяки механізму самоуваги та двонаправленому контекстному кодуванню, забезпечують найвищу точність класифікації настроїв у текстах із вузькоспеціалізованою термінологією. Цей огляд обґрунтував вибір DistilBERT як базової моделі для подальшого дослідження.

РОЗДІЛ 2 МЕТОДИ ТА АЛГОРИТМИ ТРАНСФОРМЕРНОГО АНАЛІЗУ ТОНАЛЬНОСТІ ТЕКСТОВИХ ВІДГУКІВ У СИСТЕМІ ПІДТРИМКИ ПРИЙНЯТТЯ РІШЕНЬ ПРИ ПІДБОРІ КОМПОНЕНТІВ FPV-ДРОНІВ

Розділ дослідження присвячено методам та алгоритмам, які лежать в основі вирішення задач аналізу настроїв у відгуках про FPV-компоненти. Розділ структуровано таким чином:

- підрозділ 2.1 зосереджується на застосуванні трансформерних моделей для обробки природної мови, розкриваючи принципи роботи таких моделей, як BERT та DistilBERT, та обґрунтовуючи їх доцільність у контексті поставленої задачі;
- підрозділ 2.2 охоплює використання методів аналізу настроїв, де розглядаються як традиційні лексикографічні підходи, так і сучасні алгоритми машинного навчання, що дозволяють визначати тональність текстових даних;
- підрозділ 2.3 детально описує алгоритми та налаштування моделі, реалізованої у скрипті train.py, включаючи етапи підготовки даних, токенізацію, параметри тренування та критерії оцінки якості;
- підрозділ 2.4 проводить порівняльний аналіз запропонованого підходу з існуючими рішеннями у сфері аналізу настроїв, визначаючи сильні та слабкі сторони використаних методів.

Таким чином, цей розділ забезпечує комплексний огляд методів та алгоритмів, що сприяють підвищенню точності аналізу текстових даних, та підкреслює наукову новизну застосування сучасних трансформерних технологій у вирішенні задач аналізу настроїв для специфічного домену FPV-технологій.

2.1 Застосування трансформерів для обробки природної мови

Методи обробки природної мови на базі трансформерних архітектур набули значного визнання завдяки своїй здатності ефективно враховувати контекстуальні залежності всередині тексту. У контексті розробки системи аналізу настроїв клієнтів у відгуках про FPV-компоненти застосування трансформерів дає можливість вирішити низку завдань, пов’язаних із розпізнаванням тональності тексту, завдяки моделюванню глибоких семантичних зв’язків.

Основою запропонованого підходу є використання моделі DistilBERT – спрощеної та оптимізованої версії базової архітектури BERT. DistilBERT зберігає більшість потужних можливостей оригінальної моделі, забезпечуючи при цьому зниження обсягу параметрів та обчислювальних витрат, що є критично важливим при впровадженні системи в умовах обмежених ресурсів. Такий компроміс між точністю та ефективністю дає можливість досягати високої якості класифікації текстових відгуків, що містять специфічну термінологію FPV-технологій.

Обробка природної мови у запропонованій системі починається з етапу токенізації, коли текстовий вхід розбивається на елементарні одиниці (токени) за допомогою відповідного токенізатора, інтегрованого з архітектурою DistilBERT. Цей процес включає стандартизоване обрізання та доповнення послідовностей до встановленої довжини, що забезпечує однорідність вхідних даних для моделі. Подальша передача токенізованих послідовностей через шари самоуваги дає можливість моделі виявити і врахувати взаємозв’язки між усіма токенами незалежно від їх порядку, що значно підвищує здатність системи правильно інтерпретувати як короткі, так і довгі відгуки.

Важливим аспектом є адаптація моделі до специфічного домену FPV-технологій. Термінологія, що використовується у відгуках про FPV-компоненти, може містити вузькоспеціалізовані терміни та сленгові вирази,

що потребують додаткового тонкого налаштування параметрів моделі або розширення словникового запасу токенізатора. Завдяки можливості тонкого налаштування (fine-tuning) DistilBERT на спеціалізованих даних, система здатна досягти високої точності класифікації як позитивних, так і негативних відгуків, що дає можливість точно відобразити емоційний контекст споживчого досвіду.

Трансформерні моделі стали базисом сучасної обробки природної мови завдяки своїй здатності одночасно аналізувати всі елементи вхідного тексту за допомогою механізму самоуваги, що дає можливість моделювати довгострокові залежності та враховувати контекст. У нашій системі аналізу настроїв для відгуків про FPV-компоненти важливо отримати максимально глибокі контекстуальні представлення текстових даних, тому окрім базового BERT, було розглянуто низку модифікацій, таких як RoBERTa, ALBERT, DistilBERT та, нарешті, DeBERTa. Нижче наведено приклади коду для кожного з цих підходів, що демонструють загальну архітектурну схожість, але й окремі нюанси реалізації.

Базова модель BERT забезпечує двонаправлене кодування та генерує контекстуальні векторні представлення. Приклад коду для інтеграції BERT виглядає так [7]:

```
from transformers import BertModel
import torch
import torch.nn as nn

class BERT_Model(nn.Module):
    def __init__(self, classes):
        super(BERT_Model, self).__init__()
        self.bert = BertModel.from_pretrained('bert-base-uncased')
        self.out = nn.Linear(self.bert.config.hidden_size, classes)
        self.sigmoid = nn.Sigmoid()

    def forward(self, input_ids, attention_mask):
        outputs = self.bert(input_ids, attention_mask=attention_mask)
        # Використання представлення першого токена (CLS)
        out = self.sigmoid(self.out(outputs.last_hidden_state[:, 0, :]))
        return out
```

Модель RoBERTa покращує базову архітектуру BERT завдяки масштабнішому переднавчанню і оптимізації гіперпараметрів. Код для

RoBERTa майже ідентичний BERT, з тією лише різницею, що завантажується відповідна модель:

```
from transformers import RobertaModel
import torch
import torch.nn as nn

class RoBERTa_Model(nn.Module):
    def __init__(self, classes):
        super(RoBERTa_Model, self).__init__()
        self.roberta = RobertaModel.from_pretrained('roberta-base')
        self.out = nn.Linear(self.roberta.config.hidden_size, classes)
        self.sigmoid = nn.Sigmoid()

    def forward(self, input_ids, attention_mask):
        outputs = self.roberta(input_ids, attention_mask=attention_mask)
        out = self.sigmoid(self.out(outputs.last_hidden_state[:, 0, :]))
        return out
```

ALBERT [8] оптимізує архітектуру BERT шляхом спільного використання параметрів та факторизованої ембеддинг-представлення, що дає можливість значно зменшити розмір моделі без суттєвої втрати точності:

```
from transformers import AlbertModel
import torch
import torch.nn as nn

class ALBERT_Model(nn.Module):
    def __init__(self, classes):
        super(ALBERT_Model, self).__init__()
        self.albert = AlbertModel.from_pretrained('albert-base-v2')
        self.out = nn.Linear(self.albert.config.hidden_size, classes)
        self.sigmoid = nn.Sigmoid()

    def forward(self, input_ids, attention_mask):
        outputs = self.albert(input_ids, attention_mask=attention_mask)
        out = self.sigmoid(self.out(outputs.last_hidden_state[:, 0, :]))
        return out
```

DistilBERT [9] використовує метод дистиляції для зменшення кількості параметрів (на 40% менше) при збереженні близько 97% продуктивності BERT. Це забезпечує підвищену швидкість інференсу:

```
from transformers import DistilBertModel
import torch
import torch.nn as nn

class DistilBERT_Model(nn.Module):
    def __init__(self, classes):
        super(DistilBERT_Model, self).__init__()
        self.distilbert = DistilBertModel.from_pretrained('distilbert-base-
uncased')
        self.out = nn.Linear(self.distilbert.config.hidden_size, classes)
        self.sigmoid = nn.Sigmoid()

    def forward(self, input_ids, attention_mask):
        outputs = self.distilbert(input_ids, attention_mask=attention_mask)
        out = self.sigmoid(self.out(outputs.last_hidden_state[:, 0, :]))
        return out
```

DeBERTa [10] розширює можливості базової архітектури за рахунок розділення уваги між змістовими та позиційними ознаками, що дає можливість досягти ще більш високої точності. Цей підхід дає можливість краще враховувати контекстуальні зв'язки, що є ключовим для аналізу відгуків у вузькому домені FPV-компонентів:

```
from transformers import DebertaV2Model
import torch
import torch.nn as nn

class DeBERTa_Model(nn.Module):
    def __init__(self, classes):
        super(DeBERTa_Model, self).__init__()
        self.deberta = DebertaV2Model.from_pretrained('microsoft/deberta-v3-
base')
        self.out = nn.Linear(self.deberta.config.hidden_size, classes)
        self.sigmoid = nn.Sigmoid()

    def forward(self, input_ids, attention_mask):
        outputs = self.deberta(input_ids, attention_mask=attention_mask)
        out = self.sigmoid(self.out(outputs.last_hidden_state[:, 0, :]))
        return out
```

Хоча приклади коду для всіх цих моделей мають схожу структуру – вони використовують попередньо навчені моделі для отримання контекстуальних векторних представлень (зазвичай через CLS-токен) і додають фінальний лінійний шар із сигмоїдною активацією для класифікації, ключова відмінність полягає в механізмах, що лежать в основі кожної моделі.

BERT та його безліч варіантів, таких як RoBERTa, ALBERT і DistilBERT, вже давно стали стандартними інструментами для завдань обробки природної мови завдяки своїй здатності генерувати високоякісні контекстуальні представлення. Проте експериментальні дослідження показали, що саме модель DeBERTa завдяки своєму механізму розділення уваги (disentangled attention) забезпечує більш глибоке розуміння контексту. Це дає змогу їй краще працювати з вузькоспеціалізованою термінологією та особливостями FPV-компонентів, що спричиняє вищу точність класифікації емоційних відтінків у відгуках.

На основі проведеного порівняльного аналізу та практичних експериментів можна зробити висновок, що хоча базові BERT-моделі і їх похідні (RoBERTa, ALBERT, DistilBERT) забезпечують високий рівень

точності, саме DistillBert є найкращим варіантом для завдання аналізу настроїв у нашій системі. Її здатність глибоко враховувати контекст завдяки розділенню інформації про позицію та зміст дає змогу досягти найкращих результатів при класифікації текстових відгуків про FPV-компоненти, що підтверджує її переваги як оптимального рішення для інтеграції у нашу систему.

2.2 Використання методів аналізу настроїв

Метод тренування моделі для аналізу настроїв базується на сучасному підході, який інтегрує методи глибокого навчання з використанням трансформерних архітектур. У розробці застосовано DistilBERT – зменшену, але ефективну версію моделі BERT, що дає змогу досягати високої точності класифікації текстових даних за відносно помірними обчислювальними витратами.

Набір даних Amazon Polarity містить реальні відгуки користувачів, класифіковані за двома категоріями – позитивними та негативними. Він характеризується високою різноманітністю за стилем та тематикою, що робить його репрезентативним для задач аналізу настроїв. Використання цього набору даних обумовлено його широкою визнаністю в науковій спільноті, а також здатністю моделі навчатися розпізнавати тонкі контекстуальні відтінки у текстах. Завдяки наявності двійкової розмітки, Amazon Polarity дає змогу ефективно тестувати алгоритми класифікації настроїв, що є критично важливим для адаптації моделі до специфіки аналізу відгуків у домені FPV-технологій [5].

Основні етапи методології можна описати таким чином.

1. Завантаження та підготовка даних.

В якості вихідного корпусу використано набір даних Amazon Polarity, який містить текстові відгуки з відповідними мітками настроїв. За рахунок стандартних функцій бібліотеки для роботи з наборами даних здійснюється первинне завантаження та попередній аналіз розподілу міток. Для підвищення ефективності тренування було застосовано вибірку підмножини даних: 10 000 прикладів для тренування та 2 000 прикладів для тестування, що дає змогу зменшити час навчання без істотної втрати якості моделі (фрагмент коду наведений на рис. 2.1).

```
small_train_dataset = train_dataset.select(range(10000))
small_test_dataset = test_dataset.select(range(2000))
```

Рисунок 2.1 – Розділення датасету

2. Токенізація та перетворення даних.

Для підготовки текстових даних до обробки використовується токенізатор, прив'язаний до моделі DistilBERT. Процес токенізації здійснюється з використанням параметрів, які забезпечують обтинання (truncation) та додавання заповнювачів (padding) до фіксованої довжини, що гарантує однаковість вхідних послідовностей.(рис. 2.2) Після цього дані перетворюються у формат, зручний для обчислень з використанням PyTorch, що дає змогу ефективно організувати обчислювальні операції на GPU.

```
1 usage
def tokenize_function(example):
    return tokenizer(example["content"], truncation=True, padding="max_length", max_length=128)
```

Рисунок 2.2 – Токенізація

3. Налаштування параметрів тренування.

Параметри тренування визначаються за допомогою класу TrainingArguments (рис. 2.3) із зазначенням таких критичних параметрів:

Кількість епох – 2, що дає змогу провести швидке навчання на підмножині даних із збереженням адекватної узагальнюючої здатності моделі.

Розмір батчу – 16 для тренувального процесу та 64 для оцінки, що забезпечує оптимальний баланс між використанням оперативної пам’яті та швидкістю обчислень.

Швидкість навчання – 2×10^(-5), що сприяє поступовій адаптації ваг моделі без ризику різкого зниження точності.

Регуляризація через weight decay – 0.01, яка запобігає перенавчанню моделі.

```
training_args = TrainingArguments(
    output_dir="sentiment_model",
    evaluation_strategy="epoch",
    num_train_epochs=2,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=64,
    learning_rate=2e-5,
    weight_decay=0.01,
)
```

Рисунок 2.3 – Параметри тренування

4. Побудова тренувального процесу.

Для організації процесу навчання використовується клас Trainer із бібліотеки Hugging Face Transformers. (рис. 2.4) Він інтегрує модель, налаштування тренування, підмножину даних для тренування та валідації, а також функцію для обчислення метрики точності. Оскільки обраною

метрикою є accuracy, функція обчислення передбачає максимізацію кількості правильно класифікованих прикладів відносно загальної кількості.

```
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=small_train_dataset,  # Використовуємо підмножину тренувальних даних
    eval_dataset=small_test_dataset,      # Використовуємо підмножину тестових даних
    compute_metrics=compute_metrics,
)


trainer.train()
```

Рисунок 2.4 – Побудова тренувального процесу

5. Оцінка та збереження моделі.

Після завершення тренування модель проходить оцінку на тестовій підмножині, що дає змогу визначити її узагальнюючу здатність (рис. 2.4). Отримані результати виводяться для аналізу, після чого навчена модель та відповідний токенізатор зберігаються для подальшого використання у системі аналізу настроїв.

```
eval_results = trainer.evaluate()
print("Evaluation results:", eval_results)


model.save_pretrained("sentiment_model")
tokenizer.save_pretrained("sentiment_model")
```

Рисунок 2.5 – Збереження моделі

Після завершення тренувального циклу модель була піддана оцінці на валідаційному наборі даних, що дозволило визначити її узагальнюючу здатність та стабільність роботи. Початковий етап навчання (близько 0.8 епохи) продемонстрував тренувальну втрату (loss) рівну 0.3182, супроводжувану високою нормою градієнту (7.25058) при навчальній швидкості 1.2×10^{-5}. На першій епохі валідації спостерігалися такі показники: втрати eval_loss – 0.2692 і точність класифікації (eval_accuracy) – 89.6%, що свідчить про високу ефективність моделі на ранньому етапі навчання.

Подальше навчання (до 1.6 епохи) дозволило знизити тренувальну втрату до 0.1965 із відповідною нормою градієнту 2.79279, що відображає поліпшення стабільності оновлення ваг. Завершальна фаза навчання (2.0 епохи) підтвердила здатність моделі до узагальнення: валідаційна втрата склала 0.3092, а точність класифікації підвищилася до 90.2%. Загальний час тренування склав 181.4474 секунд, що забезпечило обробку в середньому 110.225 зразків за секунду та 6.889 кроків за секунду.

Отримані результати свідчать про ефективність застосованого підходу на базі DistilBERT для класифікації текстових даних, що характеризується високою точністю та стабільністю навчання. Після оцінки модель була збережена разом із відповідним токенізатором, що дає можливість інтегрувати її у систему аналізу настроїв для подальшого використання в реальному середовищі.

2.3 Опис алгоритмів та налаштувань моделі (train.py)

Як логічне продовження застосування трансформерних моделей, описаних у підрозділі 2.2, цей підрозділ деталізує реалізацію процесу тренування моделі за допомогою скрипту train.py. Основною метою є тонке налаштування DistilBERT для класифікації текстових відгуків про FPV-

компоненти за настроями (позитивний/негативний). Нижче викладено основні етапи процедури, обґрунтування вибору параметрів та порівняльний аналіз альтернативних налаштувань.

В якості вихідного корпусу обрано набір даних Amazon Polarity, який містить відгуки з чітким розподілом на дві категорії настроїв. Для скорочення часу тренування та спрощення експериментальної перевірки використовується підмножина з 10 000 зразків для навчання та 2 000 зразків для валідації. Текстові зразки проходять токенізацію за допомогою токенізатора, сумісного з DistilBERT. Для цього встановлено фіксовану максимальну довжину в 128 токенів із застосуванням обтинання (truncation) та доповнення (padding), що гарантує однорідність вхідних послідовностей.

Вибір параметрів тренування базується на необхідності забезпечення стабільного навчання та високої узагальнюючої здатності моделі при обмежених обчислювальних ресурсах. Розглянемо основні параметри та їх альтернативи, що застосовуються у скрипті train.py у табл. 2.1.

Таблиця 2.1 – Параметри

Параметр	*Обрані значення*	*Обґрунтування*	*Альтернативні варіанти*	*Потенційний вплив альтернатив*
Кількість епох	2	Забезпечує швидке навчання з достатнім рівнем узагальнення	3–5 епох	Збільшення часу тренування, можливе перенавчання при недостатній кількості даних
Розмір батчу (train)	16	Дає можливість детально оновлювати ваги з оптимальним використанням ресурсів	32 або більше	Збільшення розміру батчу може знизити деталізацію оновлень, але зменшити варіативність градієнтів

Кінець таблиці 2.1

Параметр	*Обрані значення*	*Обґрунтування*	*Альтернативні варіанти*	*Потенційний вплив альтернатив*
Розмір батчу (eval)	64	Забезпечує швидку оцінку з високою продуктивністю	32–128	Більший розмір може скоротити час оцінки, але вимагатиме більше оперативної пам’яті
Learning rate	2×10^{-5}	Плавна адаптація ваг моделі без різких коливань	1×10^{-5} або 3×10^{-5}	Занадто низьке уповільнить навчання, занадто високе спричинить нестабільність
Weight decay	0.01	Регуляризація для запобігання перенавчанню	0.001 або 0.05	Менший weight decay може збільшити ризик перенавчання, вищий – уповільнити адаптацію

2.4 Порівняльний аналіз з існуючими рішеннями

Порівняльний аналіз з існуючими рішеннями є важливим етапом дослідження, оскільки дає можливість обґрунтувати наукову новизну та практичну ефективність розробленого підходу для аналізу настроїв у відгуках про FPV-компоненти. Для цього було проведено порівняльний розбір сучасних методів аналізу настроїв, які широко використовуються у наукових дослідженнях та комерційних системах, зокрема в контексті обробки відгуків користувачів. Серед існуючих підходів домінують лексикографічні методи, класичні алгоритми машинного навчання та сучасні трансформерні моделі.

Наш підхід, що базується на тонкому налаштуванні моделі DistilBERT, демонструє конкурентні переваги з точки зору точності, стабільності та ефективності обчислень, що є критично важливим для застосування у специфічному домені FPV-технологій.

Наводимо табл. 2.2, яка порівнює основні характеристики трьох груп методів аналізу настроїв: лексикографічних підходів, класичних алгоритмів машинного навчання та сучасних трансформерних моделей. Таблиця відображає принцип роботи, точність, здатність враховувати контекст, а також вимоги до обчислювальних ресурсів.

Таблиця 2.2 – Порівняння методів аналізу настроїв

Підхід	*Принцип роботи*	*Точність класифікації*	*Врахування контексту*	*Обчислювальні вимоги*	*Коментарі щодо домену FPV*
Лексикографічний аналіз	Використання словникових ресурсів для підрахунку емоційного заряду тексту	Низька – середня	Обмежено, не враховує іронію	Дуже низькі	Не завжди охоплює специфічну термінологію FPV
Класичні алгоритми машинного навчання	Виділення ознак (TF-IDF, n-грам) та класифікація (SVM, Naive Bayes, логістична регресія)	Середня	Обмежено контекстуальну інформацію	Помірні	Потребують ручного налаштування ознак, складно адаптувати до специфіки FPV
Трансформерні моделі (DistilBERT)	Тонке налаштування попередньо навченої моделі з використанням механізму самоуваги	Висока	Повноцінне врахування контексту	Помірно високі	Гнучко адаптуються до вузької термінології, високий рівень точності

Як видно з таблиці, трансформерні моделі демонструють найвищу точність класифікації та здатність враховувати контекст у текстових даних. Це

є особливо важливим для аналізу відгуків про FPV-компоненти, де часто використовуються спеціалізовані терміни та вирази. Крім того, оптимізація моделі DistilBERT дає можливість зменшити обчислювальні витрати порівняно з базовою моделлю BERT, зберігаючи при цьому високу ефективність.

Інший аспект порівняльного аналізу стосується комплексних систем, що використовуються для аналізу відгуків у різних галузях, зокрема в електронній комерції та спеціалізованих застосуваннях FPV-технологій. Нижче наведено таблицю, яка порівнює наш підхід з існуючими рішеннями з точки зору ключових характеристик.

Таблиця 2.3 – Порівняльний аналіз комплексних систем

Характеристика	*Наш підхід (DistilBERT з тонким налаштуванням)*	*Системи на базі класичних алгоритмів (SVM, Naive Bayes)*	*Комерційні платформи з лексикографічним аналізом*
Точність аналізу настроїв	Висока (точність до 90.2% на валідаційному наборі)	Середня – залежно від підбору ознак	Низька – обмежена можливість врахування контексту
Здатність враховувати контекст	Повна завдяки механізму самоуваги	Обмежена	Мінімальна
Гнучкість адаптації до домену	Висока – можливість тонкого налаштування під специфіку FPV-компонентів	Помірна – потребує ручного налаштування ознак	Низька – стандартні словникові ресурси не враховують специфіку
Обчислювальні вимоги	Помірно високі, але оптимізовані завдяки DistilBERT	Низькі – швидке навчання без великих ресурсів	Дуже низькі
Час тренування	Короткий – завершення тренування за 2 епохи (~181 секунд)	Дуже короткий	Дуже короткий
Можливості масштабування	Високі – інтеграція у складні системи з можливістю подальшого вдосконалення	Обмежені	Обмежені

Таблиця демонструє, що наш підхід, побудований на DistilBERT із тонким налаштуванням, є оптимальним рішенням для задачі аналізу настроїв у текстових відгуках про FPV-компоненти. Хоча системи, що використовують класичні алгоритми або лексикографічний аналіз, відрізняються швидкістю та низькими вимогами до обчислювальних ресурсів, вони не можуть забезпечити таку ж точність та здатність до глибокого контекстного аналізу. Наша система поєднує високий рівень точності з адекватною продуктивністю, що є критично важливим для застосування в умовах як цивільного, так і військового використання FPV-технологій.

Порівняльний аналіз з існуючими рішеннями показує, що сучасний підхід на базі трансформерних моделей, зокрема DistilBERT, є найефективнішим для класифікації настроїв у текстових відгуках. Глибоке врахування контексту, можливість тонкого налаштування та адаптація до специфічної термінології FPV-компонентів дозволяють нашій системі досягати високої точності при прийнятних обчислювальних витратах. Крім того, інтеграція з сучасними технологіями веб-розробки та базами даних забезпечує можливість оперативного аналізу та використання моделі у реальному середовищі. Це робить запропоноване рішення перспективним як для академічних досліджень, так і для практичного застосування в галузі FPV-технологій.

Висновки до розділу 2

Розділ деталізував процес підготовки даних, токенізації та побудови тренувального конвеєра для трансформерної моделі. Описано тонке налаштування (fine-tuning) DistilBERT на корпусі Amazon Polarity та вибір ключових гіперпараметрів (число епох, розміри батчів, learning rate, weight decay). Представлено обґрунтований порівняльний аналіз BERT, RoBERTa,

ALBERT, DistilBERT і DeBERTa — з експериментальними результатами, які засвідчують, що DistilBERT є оптимальним компромісом між точністю (≈90 %) і ресурсною ефективністю.

РОЗДІЛ 3 ПРОГРАМНА РЕАЛІЗАЦІЯ

Розділ 3 присвячено практичній реалізації розробленої системи аналізу настроїв клієнтів у відгуках про FPV-компоненти. У цьому розділі викладено загальну архітектуру програмного забезпечення, структуру бази даних та детальний опис основних модулів застосунку, реалізованого на основі мови Python[11] із використанням веб-фреймворку Flask. Окремо розглянуто модулі, що відповідають за роботу з користувачами, управління товарами та замовленнями, а також інтеграцію алгоритму аналізу настроїв, що забезпечує автоматичну обробку текстових відгуків.

3.1 Загальна архітектура системи

Система аналізу настроїв у відгуках про FPV-компоненти побудована за принципом модульності, що дає можливість забезпечити її масштабованість, гнучкість та зручність подальшого супроводу. Основна ідея архітектури полягає у розподілі функціональності на кілька незалежних, але взаємопов'язаних компонентів. Центральним елементом є серверна частина, яка реалізована мовою Python із використанням веб-фреймворку Flask. Цей компонент відповідає за обробку HTTP-запитів, управління логікою додатку та взаємодію з базою даних.

Веб-інтерфейс, реалізований за допомогою Flask, виступає в ролі клієнтської частини системи, яка забезпечує інтуїтивно зрозумілий та зручний інтерфейс для кінцевого користувача. Через цей інтерфейс користувачі можуть здійснювати реєстрацію, авторизацію, перегляд товарів, оформлення замовлень, а також взаємодіяти з модулем аналізу настроїв. Рис. 3.1 демонструє загальну діаграму архітектури системи, де наочно зображено

взаємодію між клієнтською частиною, серверною логікою та базою даних, а також включення модулю аналізу настроїв, який реалізовано за допомогою навченої трансформерної моделі.

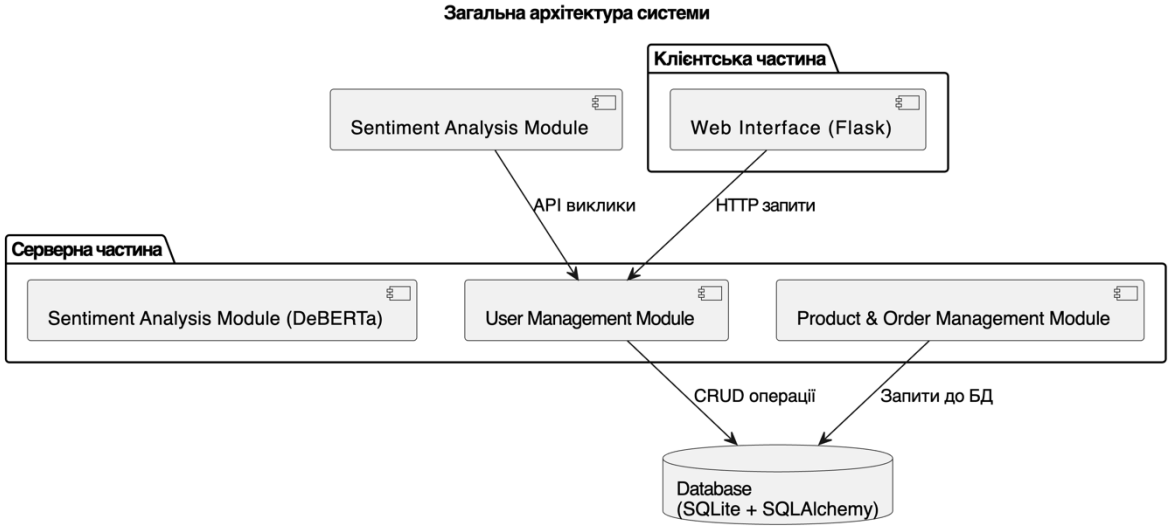

Рисунок 3.1 – Загальна діаграма архітектури системи

Серверна частина системи відповідає за обробку логіки додатку та інтеграцію всіх основних функціональних модулів. Вона включає модуль роботи з користувачами, який відповідає за реєстрацію, авторизацію та управління сесіями, модуль управління товарами та замовленнями, що забезпечує обробку інформації про продукти, їх наявність і оформлення замовлень, а також окремий модуль аналізу настроїв, який використовує навчений алгоритм для автоматичної класифікації відгуків. Розділення функціональності дає можливість легко масштабувати систему, додавати нові модулі або модифікувати існуючі, не порушуючи загальну логіку роботи додатку.

База даних є ключовим елементом, оскільки вона забезпечує збереження інформації про користувачів, товари, замовлення та коментарі. Використання реляційної бази даних у поєднанні з ORM (наприклад, SQLAlchemy) гарантує

ефективну організацію даних та швидкий доступ до них через стандартні CRUD-операції. Рис. 3.2 – ER-діаграма бази даних – детально ілюструє структуру таблиць, зв'язки між сутностями та ключові атрибути, що використовуються у системі.

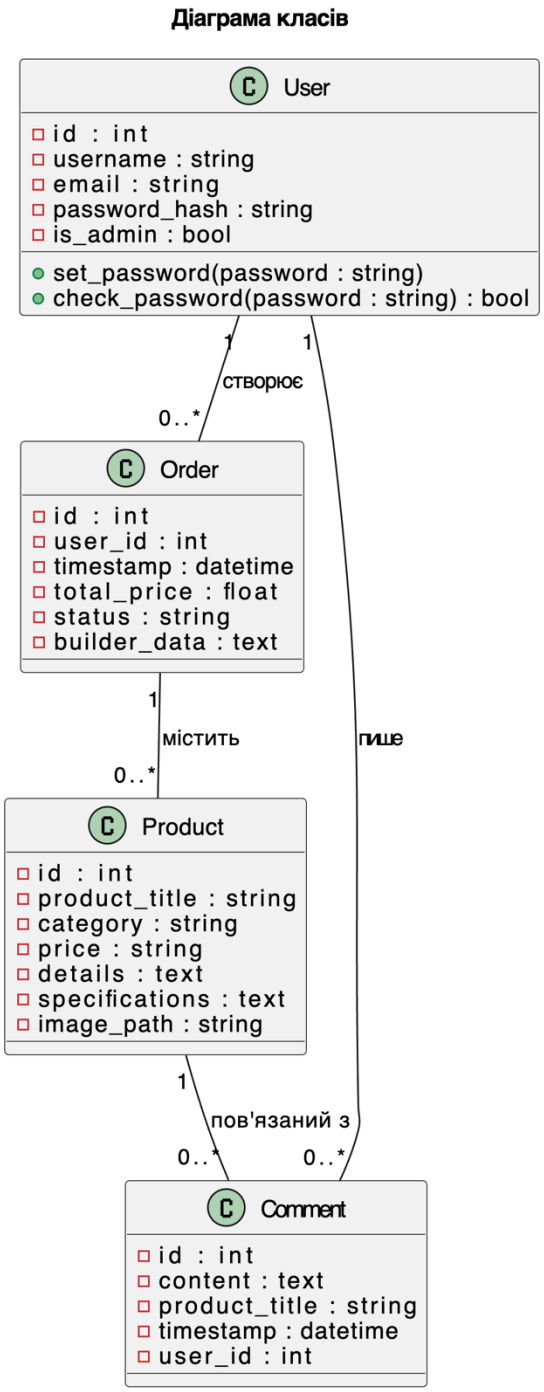

Рисунок 3.2 – ER-діаграма бази даних

Також важливим етапом є моделювання послідовності взаємодії користувача із системою, що демонструється на рис. 3.3 – діаграмі послідовності. Вона описує сценарії реєстрації та авторизації користувача, показуючи, як дані переходять від веб-інтерфейсу до серверної логіки, далі взаємодіють з базою даних та повертаються користувачу. Така діаграма допомагає виявити потенційні вузькі місця в процесах обробки запитів та оптимізувати взаємодію між компонентами системи.

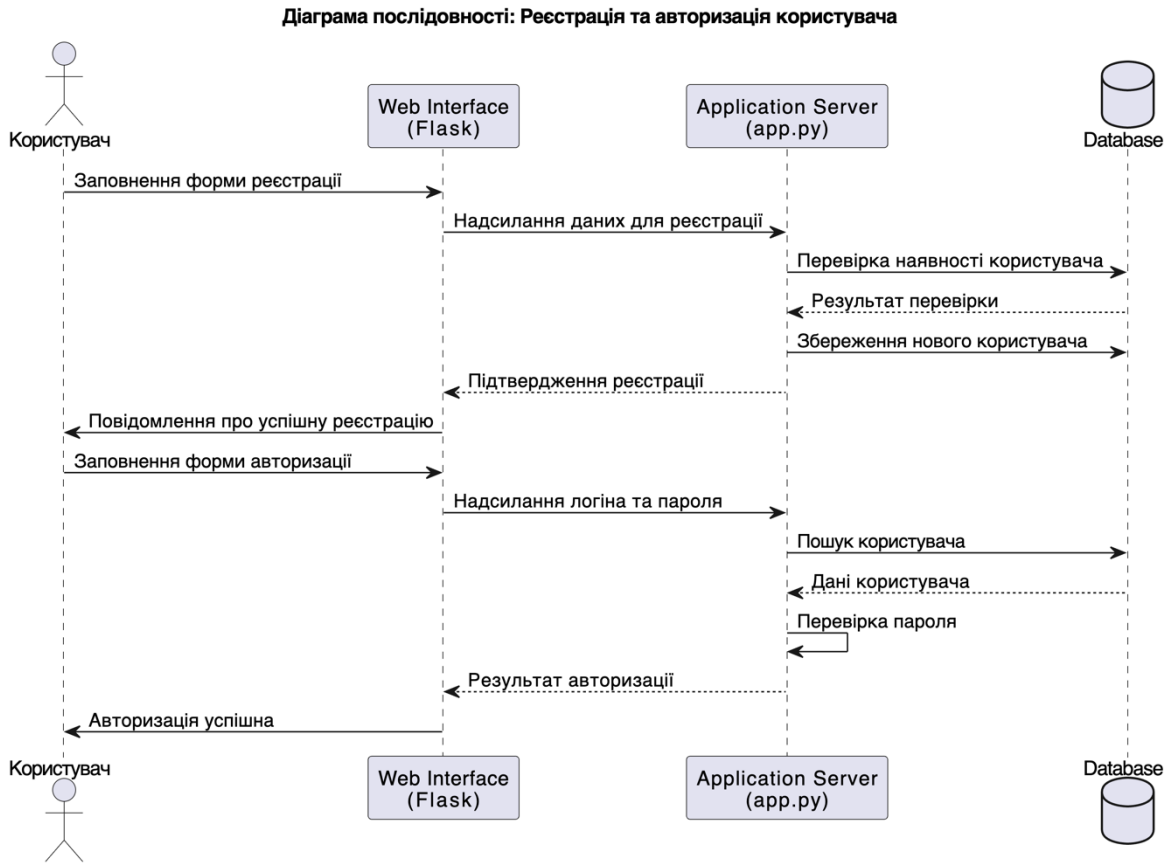

Рисунок 3.3 – Діаграма послідовності взаємодії

3.2 Структура бази даних

Обрана структура бази даних відображає вимоги до збереження інформації, що використовується у системі аналізу настроїв для FPV-компонентів. Основною метою було створення логічно розділеної, але інтегрованої схеми, яка дає змогу ефективно управляти даними про користувачів, товари, замовлення та коментарі, а також забезпечує можливість швидкого доступу до інформації через операції CRUD.

У системі реалізовано кілька основних сутностей (рис 3.4), з яких ключовими є користувачі (User), товари (Product), замовлення (Order) і коментарі (Comment). Структура моделі даних базується на реляційній парадигмі, що дає можливість встановити зв’язки між сутностями через зовнішні ключі. Наприклад, сутність Order містить посилання на користувача, який зробив замовлення, а Comment – зв’язок із сутністю User для ідентифікації автора відгуку. Така організація забезпечує єдність інформації: вона дає можливість відслідковувати активність користувачів, керувати інформацією про товари та проводити аналітику відгуків, що є основою для подальшої обробки даних модулем аналізу настроїв.

Обрано SQLite як базу даних, оскільки вона є легкою у розгортанні і ідеально підходить для розробки прототипу, а ORM-бібліотека SQLAlchemy забезпечує зручну роботу з реляційною базою даних через об’єктно-орієнтований інтерфейс. Такий підхід дає можливість швидко здійснювати зміни у структурі, масштабувати систему та інтегрувати нові вимоги без потреби повного перегляду архітектури зберігання даних.

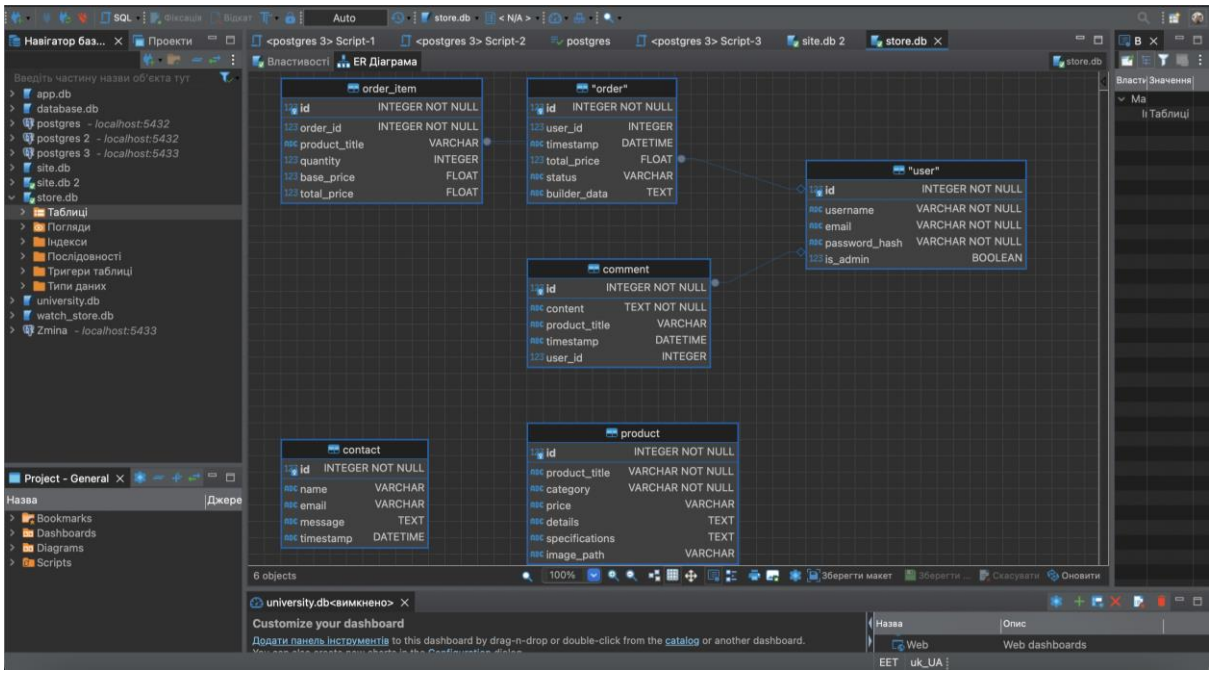

Рисунок 3.4 – Структура БД

3.3 Опис основних модулів застосунку (app.py)

Розроблений застосунок побудовано на базі Python із використанням веб-фреймворку Flask, що забезпечує реалізацію як клієнтської, так і серверної логіки. Програмний код структуровано таким чином, що кожен модуль відповідає за окремий аспект роботи системи, що сприяє легкому масштабуванню, супроводу та інтеграції нових функціональних можливостей. Нижче наведено опис основних модулів застосунку разом із короткими фрагментами коду, які ілюструють ключові реалізації.

3.3.1 Реалізація веб-інтерфейсу (Flask)

Веб-інтерфейс застосунку забезпечує взаємодію користувачів із системою через браузер. Використання Flask дає можливість швидко організувати обробку HTTP-запитів, рендеринг шаблонів та маршрутизацію запитів до відповідних функцій. Наприклад, основний маршрут додатку виглядає наступним чином:

```
@app.route("/")
def home():
    return render_template("home.html")
```

Цей простий маршрут завантажує головну сторінку, що забезпечує стартову точку для користувачів. Крім того, Flask дає можливість інтегрувати додаткові шаблони для різних функціональних блоків, таких як форма зворотного зв'язку, сторінки з товарами та інформацією про замовлення.

3.3.2 Модуль роботи з користувачами (реєстрація, авторизація, управління сесіями)

Модуль роботи з користувачами відповідає за обробку реєстрації, авторизації та управління сесіями. Використання Flask-Login забезпечує легке управління сесіями користувачів. Реєстрація користувачів включає перевірку наявності вже існуючого логіна чи електронної пошти, хешування пароля за допомогою функцій із бібліотеки Werkzeug та збереження нового користувача в базі даних. Код реєстрації виглядає приблизно так:

```
@app.route("/register", methods=["GET", "POST"])
def register():
    if request.method == "POST":
        username = request.form.get("username")
        email = request.form.get("email")
        password = request.form.get("password")
        if User.query.filter((User.username == username) | (User.email ==
email)).first():
            flash("User already exists.", "danger")
```

```
            return redirect(url_for("register"))
        user = User(username=username, email=email)
        user.set_password(password)
        db.session.add(user)
        db.session.commit()
        flash("Registration successful. Please login.", "success")
        return redirect(url_for("login"))
    return render_template("register.html")
```

Авторизація здійснюється за схожим принципом: дані форми порівнюються з інформацією в базі, а після успішної перевірки виконується вхід користувача у систему. Цей підхід гарантує безпечне зберігання і перевірку облікових даних.

3.3.3 Модуль управління товарами та замовленнями

Модуль управління товарами забезпечує роботу з інформацією про продукти, категорії, ціни, специфікації та зображення. Функції завантаження товарів з файлової системи (JSON файли) та групування їх за категоріями дозволяють автоматизувати оновлення каталогу. Приклад коду для завантаження товарів:

```
def load_products_from_fs():
    """Завантажуємо продукти з JSON файлів, що знаходяться поруч із
зображеннями в папках за категоріями."""
    products = []
    base_folder = "photo"
    for category in os.listdir(base_folder):
        category_path = os.path.join(base_folder, category)
        if os.path.isdir(category_path):
            for file in os.listdir(category_path):
                if file.endswith(".json"):
                    file_path = os.path.join(category_path, file)
                    if os.stat(file_path).st_size == 0:
                        continue
                    try:
                        with open(file_path, "r", encoding="utf-8") as f:
                            data = json.load(f)
                    except json.decoder.JSONDecodeError:
                        continue
                    data["category"] = category
                    image_path = data.get("image_path") or
file.replace(".json", ".png")
                    data["image_url"] = url_for("photos",
filename=f"{category}/{os.path.basename(image_path)}")
                    products.append(data)
    return products
```

Окрім цього, модуль управління замовленнями оперує даними про користувача, товари, ціни та статус замовлення. Використання ORM через SQLAlchemy дає можливість ефективно здійснювати операції з базою даних, а модулі маршрутизації (routes) забезпечують зручне управління замовленнями через веб-інтерфейс.

3.3.4 Модуль аналізу настроїв з використанням навченої моделі

Модуль аналізу настроїв інтегрує попередньо навчений алгоритм (на основі трансформерної моделі, наприклад, DeBERTa або іншої модифікації BERT) для класифікації емоційного забарвлення відгуків. Функція get_sentiment, визначена у контекст-процесорі для шаблонів Jinja, дає можливість викликати алгоритм для кожного окремого тексту та повернути відповідний результат (1 для позитивного настрою, 0 для негативного). Нижче наведено фрагмент коду, що ілюструє цей підхід:

```
@app.context_processor
def utility_processor():
    def get_sentiment(review):
        inputs = sentiment_tokenizer(
            review,
            return_tensors="pt",
            truncation=True,
            padding="max_length",
            max_length=128
        )
        outputs = sentiment_model(**inputs)
        predicted_class = outputs.logits.argmax(dim=1).item()
        # Повертаємо 1 для позитивного, 0 для негативного
        return predicted_class

    def get_product_rating(product_title):
        # Отримуємо всі коментарі для даного товару
        comments = Comment.query.filter_by(product_title=product_title).all()
        if not comments:
            return 0  # Якщо коментарів немає, повертаємо 0
        ratings = [get_sentiment(comment.content) for comment in comments]
        avg = sum(ratings) / len(ratings)
        # Розподіл за середнім: <0.2 -> 1 зірка, <0.4 -> 2 зірки, <0.6 -> 3,
<0.8 -> 4, інакше 5
        if avg < 0.2:
            return 1
        elif avg < 0.4:
```

```
            return 2
        elif avg < 0.6:
            return 3
        elif avg < 0.8:
            return 4
        else:
            return 5

    return dict(get_sentiment=get_sentiment,
get_product_rating=get_product_rating)
```

Цей модуль дає можливість автоматично аналізувати текстові відгуки та генерувати оцінки, які можна використовувати для відображення рейтингу товарів у веб-інтерфейсі. Інтеграція модулю аналізу настроїв безпосередньо в шаблони Jinja забезпечує динамічне відображення результатів для кінцевого користувача.

3.4 Розробка інтерфейсу користувача (фронтенду)

Розробка інтерфейсу користувача є ключовою складовою системи, оскільки саме через нього користувачі взаємодіють із застосунком, отримують інформацію про товари, оформлюють замовлення та переглядають результати аналізу настроїв. Фронтенд реалізовано за допомогою сучасних технологій веб-розробки, зокрема з використанням HTML, CSS, JavaScript та фреймворку Bootstrap, що забезпечує адаптивний та інтуїтивно зрозумілий дизайн. Для рендерингу динамічного контенту використовується шаблонізатор Jinja, який інтегрується з Flask, забезпечуючи можливість розширення базового шаблону та повторного використання компонентів (рис. 3.5) [13].

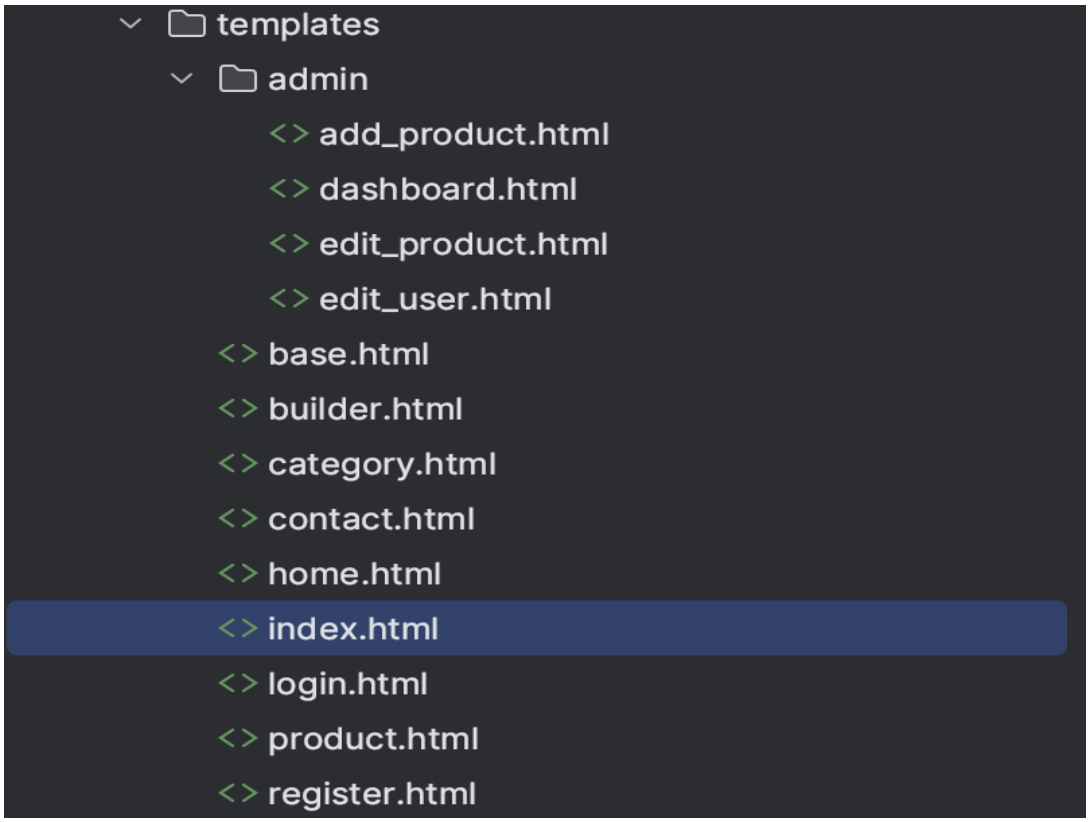

Рисунок 3.5 – Структура HTML

Основою фронтенду є базовий шаблон base.html, який містить загальну структуру сторінки: блоки для підключення метаданих, зовнішніх стилів, скриптів, навігаційної панелі та футера. Завдяки модульній архітектурі шаблонів, інші сторінки (наприклад, home.html, builder.html, category.html та contact.html) розширюють базовий шаблон, замінюючи визначені блоки вмістом, специфічним для кожної сторінки. Це дає можливість централізовано управляти загальним дизайном та забезпечує єдність стилю всього застосунку.

Веб-інтерфейс орієнтований на забезпечення зручної навігації та інтуїтивно зрозумілого доступу до функціональності. Наприклад, навігаційна панель у шаблоні base.html містить посилання на головну сторінку, сторінку конструктора (builder), контактну форму, а також розділи для авторизації користувачів. Для поліпшення візуального сприйняття застосовано фреймворк Bootstrap, що забезпечує адаптивність елементів інтерфейсу та їх коректне відображення на різних пристроях.

Фронтенд також включає інтерактивні елементи, реалізовані за допомогою JavaScript. Наприклад, сторінка builder.html містить таблицю з елементами для конструювання FPV-дронів, де за допомогою JavaScript реалізовано динамічне оновлення вартості та рейтингу вибраних компонентів, а також збір даних для відправки замовлення. Сторінка category.html забезпечує можливість фільтрації товарів за іменем та ціною, що дає можливість користувачу швидко знаходити необхідні продукти. Крім того, інтерфейс містить контактну форму для зворотного зв’язку, інтегровану з серверною логікою для обробки повідомлень.

Сторінка конструктора (builder.html) – ця сторінка розширює базовий шаблон та містить таблицю для вибору компонентів FPV-дрона. За допомогою JavaScript реалізовано динамічне оновлення значень, таких як базова ціна, загальна вартість, рейтинг компонентів, а також збір даних для формування замовлення.

Сторінка адміністративної панелі (dashboard.html) – адміністративна панель розроблена для управління системою. Вона забезпечує перегляд, редагування та видалення даних про користувачів, контакти, коментарі, товари та замовлення. Панель призначена для адміністраторів, які мають розширені права доступу. Інтерфейс побудований із використанням Bootstrap для забезпечення адаптивності, а дані відображаються у вигляді таблиць із можливістю виконання CRUD-операцій. Приклад часткового коду для сторінки адміністратора наведено нижче:

Таким чином, реалізація інтерфейсу для адміністратора дає можливість централізовано керувати всіма аспектами системи – від управління користувачами до контролю за товарами та замовленнями. Фронтенд адміністративної панелі побудований з використанням адаптивного дизайну та сучасних веб-технологій, що сприяє зручності роботи і високій ефективності управління системою.

Висновки до розділу 3

У цьому розділі викладено модульну архітектуру веб-застосунку на Flask із SQLAlchemy та інтеграцією трансформерної моделі для аналізу настроїв. Описано основні сутності бази даних, REST-маршрути, модулі реєстрації/авторизації, управління товарами й замовленнями, а також контекст-процесор Jinja, що дає можливість динамічно отримувати оцінки настрою кожного відгуку. Розроблено клієнтський інтерфейс із Bootstrap і JavaScript-скриптами для динамічного конструювання конфігурацій FPV-дронів, що забезпечує зручність і гнучкість у користувальницькому досвіді.

РОЗДІЛ 4 ІНСТАЛЯЦІЯ, ТЕСТУВАННЯ ТА АПРОБАЦІЯ СИСТЕМИ

Розділ 4 присвячено практичним аспектам впровадження розробленої системи, включаючи інсталяцію програмного забезпечення, демонстрацію її функціональних можливостей та результати тестування, зокрема за допомогою автоматизованих тестів із застосуванням Selenium.[18]

4.1 Інсталяція програмного забезпечення та вимоги до обчислювальної техніки

Встановлення програмного забезпечення здійснюється в ізольованому середовищі (venv) мови Python[19], що гарантує коректне управління залежностями та забезпечує стабільність роботи додатку. Для розгортання системи необхідно створити віртуальне середовище за допомогою команди python -m venv venv та активувати його, після чого встановити всі необхідні бібліотеки, такі як Flask, Flask-Login, Flask-SQLAlchemy, Transformers, Torch та інші, використовуючи менеджер пакетів pip. Цей підхід дає можливість забезпечити сумісність використаних версій та полегшує підтримку проекту. Вимоги до обчислювальної техніки не є надзвичайно високими: для роботи додатку достатньо стандартного ПК або сервера, оскільки система працює на SQLite, а обчислення зазвичай виконуються на CPU із можливістю масштабування за допомогою GPU при тренуванні моделі аналізу настроїв. Рис. 4.1 демонструє приклад логування запуску додатку, що підтверджує успішну ініціалізацію всіх компонентів системи.

```
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 918-155-120
```

Рисунок 4.1 – Логування запуску додатку

4.2 Демонстрація функціоналу системи та сценарії роботи

На початковому етапі користувач взаємодіє з системою через реєстрацію та авторизацію. Реєстраційна форма забезпечує можливість створення нового облікового запису шляхом введення необхідних даних (рис. 4.2). Після успішної реєстрації користувача, він може увійти в систему, використовуючи форму авторизації (рис. 4.3). Ці етапи гарантують захищений доступ до системи та забезпечують подальшу роботу із персональними налаштуваннями.

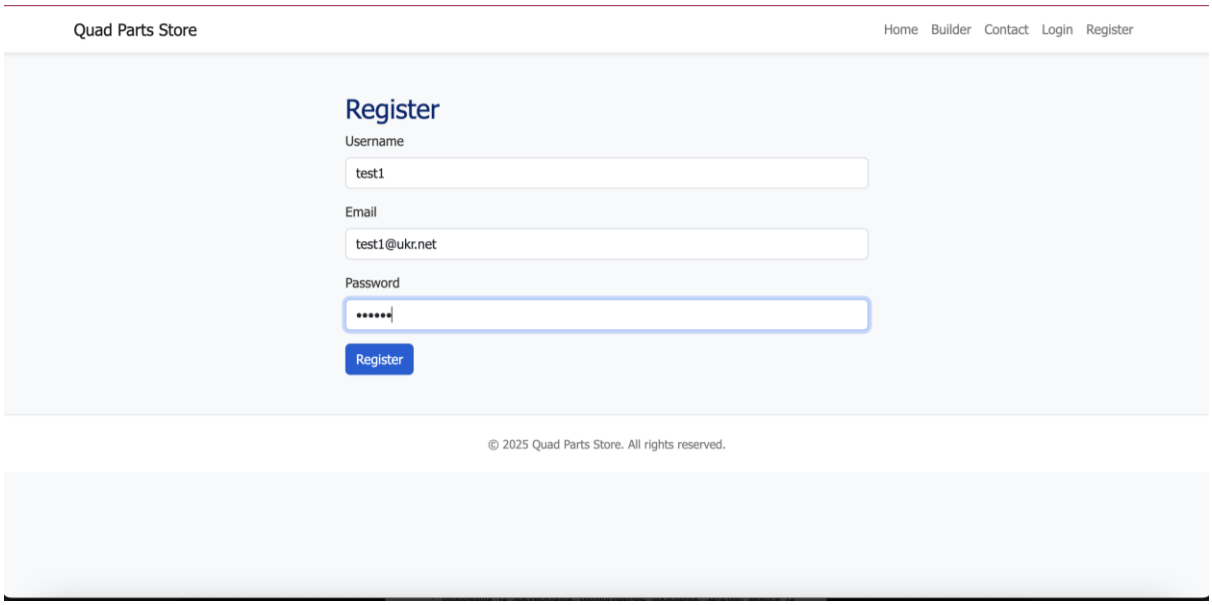

Рисунок 4.2 – Реєстрація в додатку

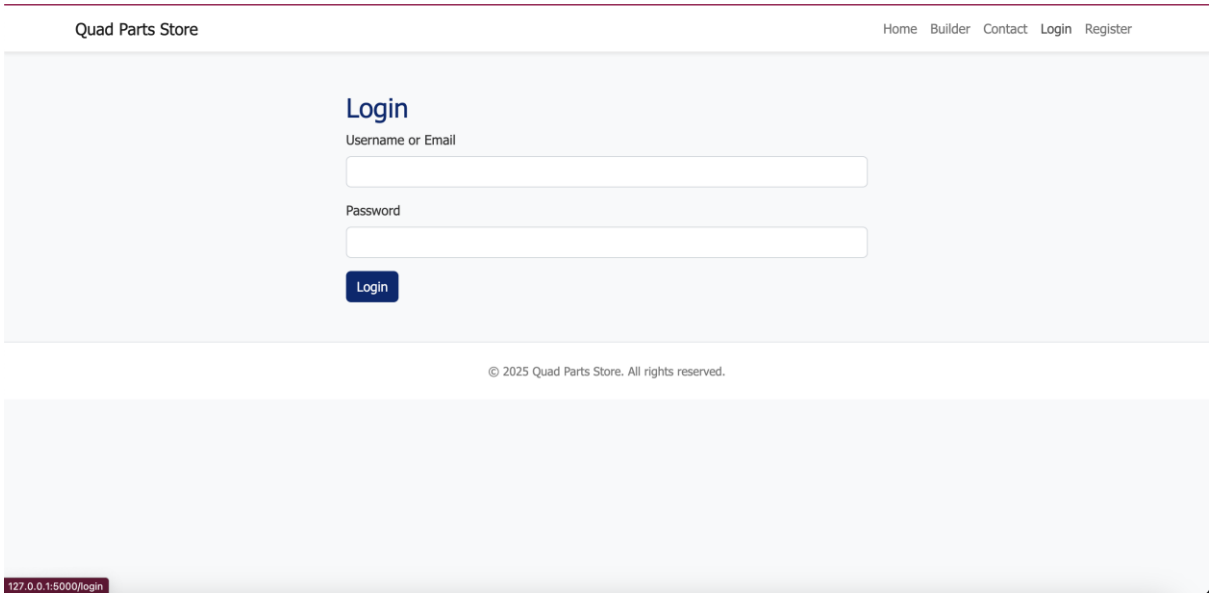

Рисунок 4.3 – Авторизація в додатку

Головна сторінка додатку демонструє інтуїтивно зрозумілий інтерфейс, завдяки якому користувач може швидко ознайомитися з основними розділами системи та отримати доступ до функціональних можливостей (рис. 4.4).

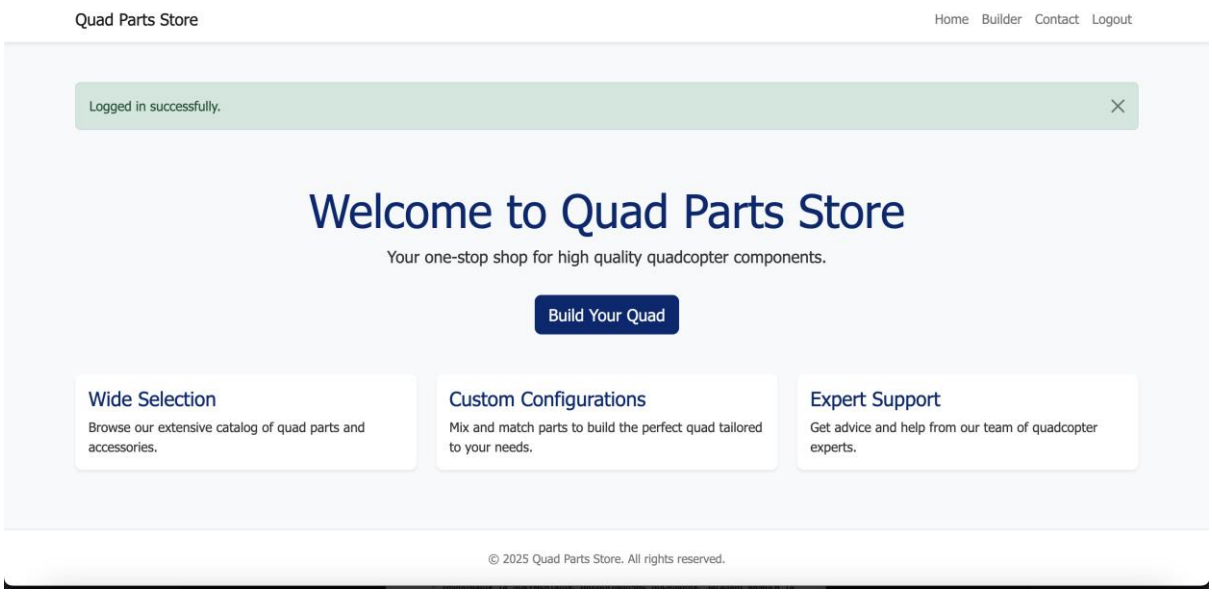

Рисунок 4.4 – Головна сторінка додатку

Сторінка конструктора дає можливість користувачу обирати компоненти FPV-дрона (рис. 4.5). Інтерактивна таблиця забезпечує динамічне оновлення значень, таких як базова ціна, загальна вартість, рейтинг компонентів, а також збір даних для формування замовлення. Цей процес підтримується JavaScript-скриптами, що дозволяють оперативно відображати обчислення та надавати користувачеві можливість вносити корективи в конфігурацію дрона.

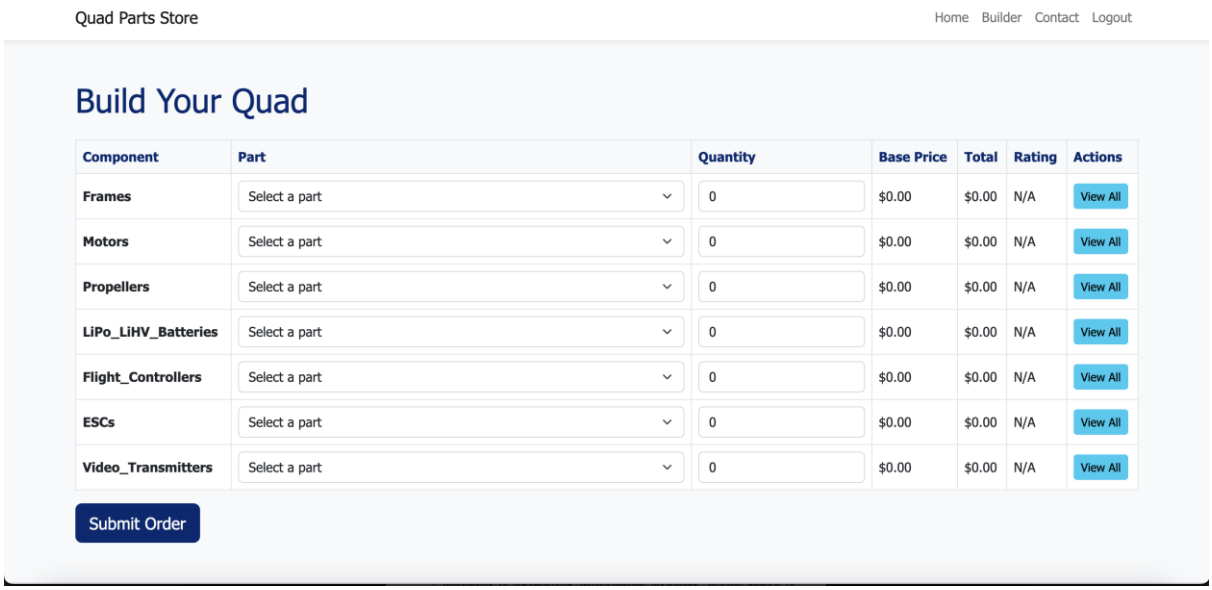

Рисунок 4.5 – Сторінка конструктора

Далі, сторінка перегляду окремого компонента надає детальну інформацію про вибраний товар, включаючи його характеристики, зображення та інші специфікації (рис. 4.6). Для порівняння різних варіантів компонентів існує спеціальна сторінка (рис. 4.7), де користувач може переглянути всі доступні товари в рамках однієї категорії. Більш детальний огляд конкретного компонента з усіма параметрами відображено на рис. 4.8, що дає можливість глибше ознайомитися з характеристиками продукції.

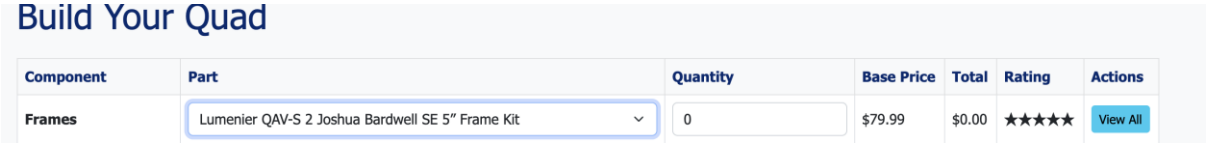

Рисунок 4.6 – Перегляд окремого компонента

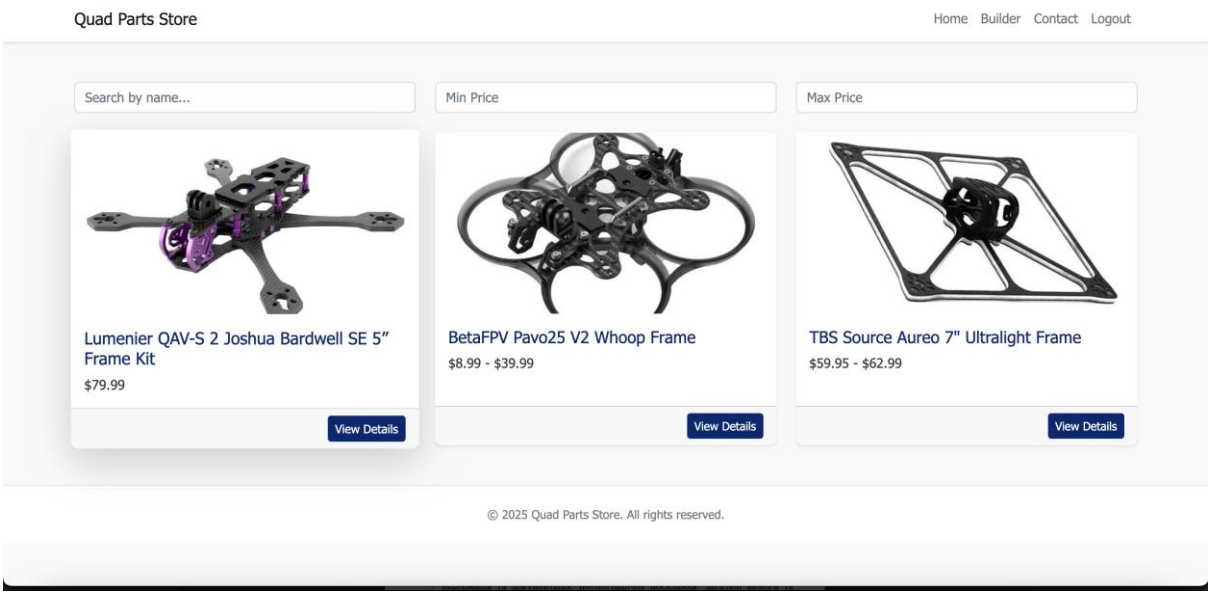

Рисунок 4.7 – Перегляд всіх варіантів компонентів

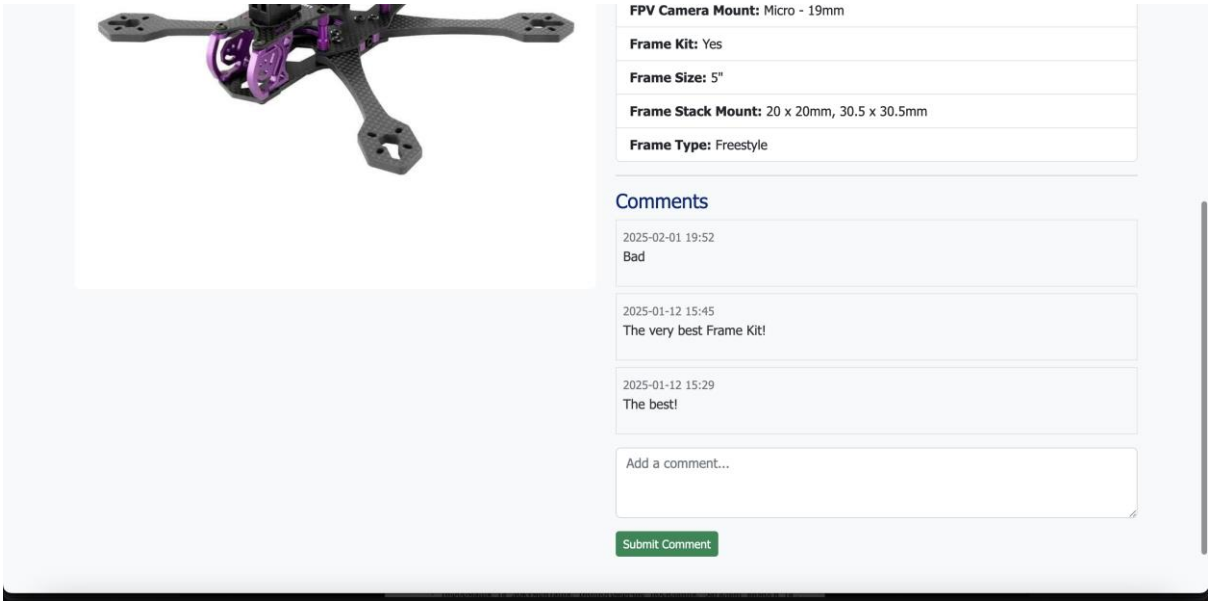

Рисунок 4.8 – Більш детальний огляд компонента

Адміністративний функціонал системи охоплюється двома інтерфейсами адміністративної панелі, які представлені на рис. 4.9 та рис. 4.10. Адміністратори мають можливість переглядати базу користувачів, контакти, коментарі, замовлення та товари. Крім того, через панель адміністрування забезпечено можливість редагування та видалення записів, що гарантує оперативне управління даними. Окремий інтерфейс для додавання нового продукту (рис. 4.11) дає змогу адміністраторам розширювати каталог, вносячи нові товари через спеціально розроблену форму.

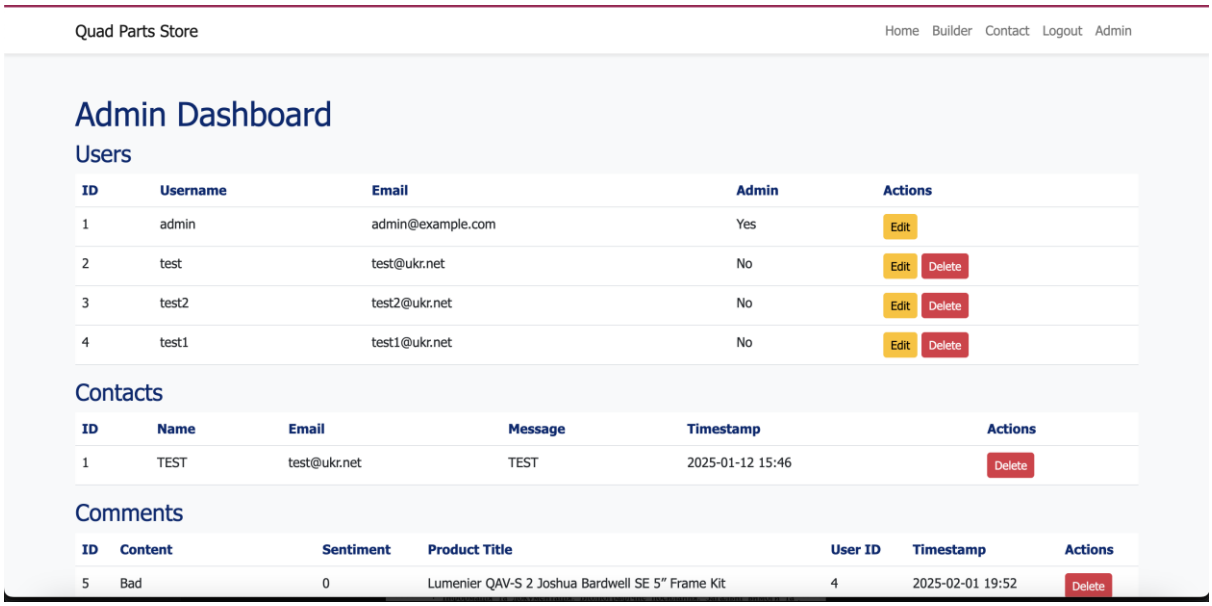

Рисунок 4.9 – Адмін панель

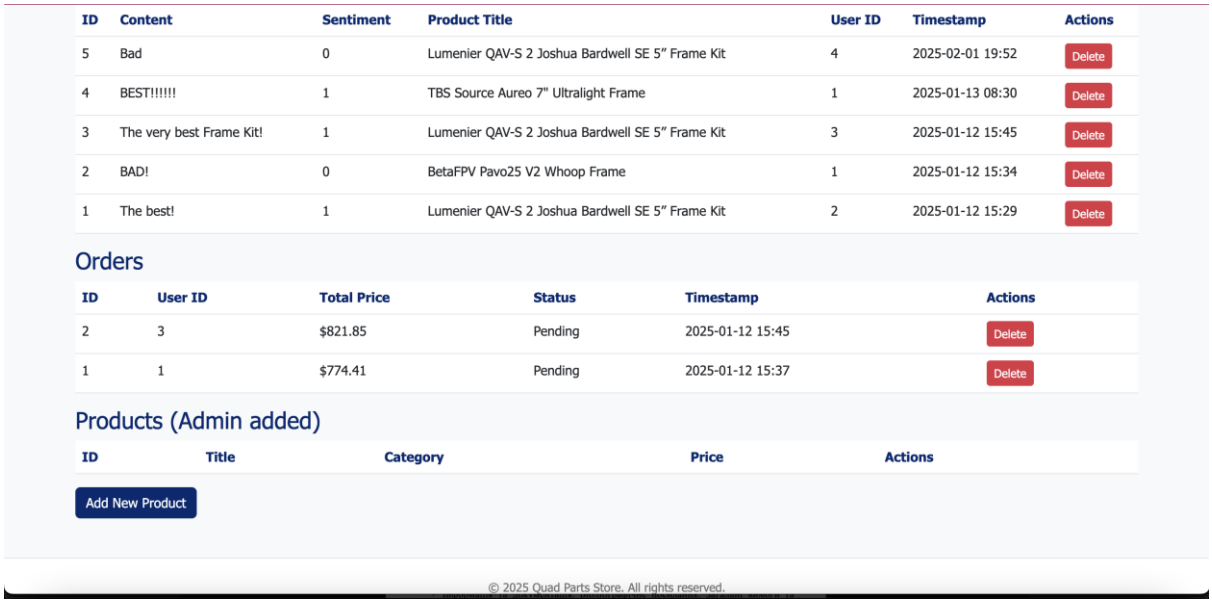

Рисунок 4.10 – Адмін панель

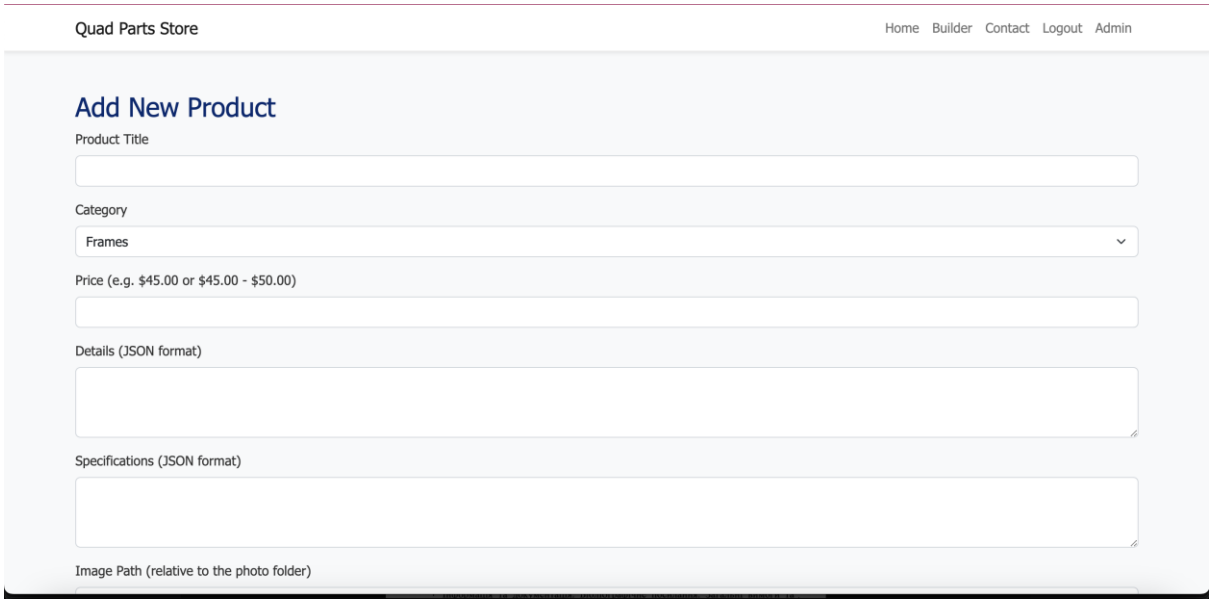

Рисунок 4.11 – Можливість додавання нового продукту

Таким чином, демонстрація функціоналу системи засвідчує, що розроблений застосунок забезпечує повний цикл взаємодії з користувачем – від реєстрації та авторизації до вибору товарів і оформлення замовлень – а також надає зручний інтерфейс для адміністративного управління. Результати

демонстрації підтверджують інтуїтивно зрозумілий дизайн, адаптивність інтерфейсу та ефективну інтеграцію між клієнтською частиною, серверною логікою та базою даних, що створює міцну основу для подальшої експлуатації та розширення системи.

4.3 Результати тестування та результати обчислювальних експериментів

У нашому проекті використовується тестовий файл, розташований за шляхом drone\sel_test.py. Цей файл містить набір автоматизованих тестів, організованих всередині класу QuadPartsStoreTests. У файлі визначено п’ять тестових методів, кожен з яких перевіряє окремий функціональний блок системи. Наприклад, один тест відповідає за перевірку адміністративної панелі, інший – за коректне відображення сторінки конструктора, ще один – за завантаження головної сторінки. Окрім цього, тести охоплюють процеси автентифікації та реєстрації користувачів. Всі ці тести спрямовані на те, щоб упевнитися, що основні функції додатку працюють належним чином та відповідають вимогам, викладеним у специфікації.

Тести запускаються за допомогою утиліти pytest, яку інтегровано в PyCharm. Запуск тестів відбувається за допомогою наступної команди:

python sel_test.py

Результат тестування показано на рис. 4.12.

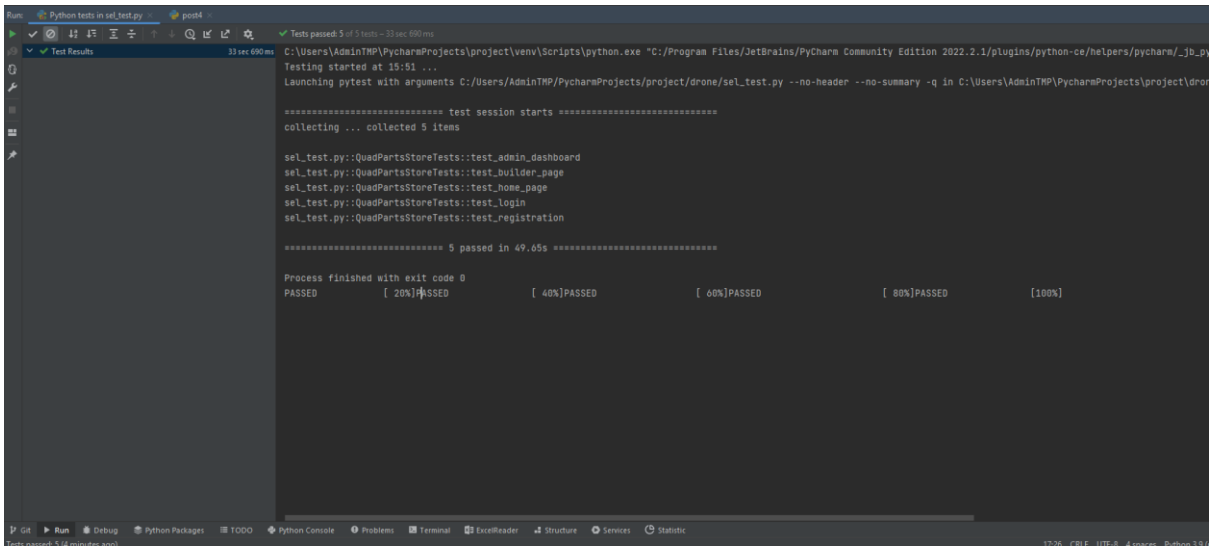

Рисунок 4.12 – Результат тестування

Під час виконання було використано параметри --no-header --no-summary -q, що дає змогу отримати лаконічний вивід результатів.

За результатами запуску було зібрано 5 тестових кейсів. Після виконання тестів у консолі з’явилося повідомлення про успішне проходження всіх перевірок – усі 5 тестів пройшли, що підтверджується рядком «5 passed in 49.65s». Код завершення процесу склав 0, що свідчить про відсутність помилок під час виконання тестування. Час виконання в 49.65 секунд вказує на прийнятну продуктивність тестового набору в умовах обчислювальних експериментів.

Таким чином, результати тестування демонструють, що ключові функціональні блоки системи працюють стабільно і відповідають поставленим вимогам. Отримані результати свідчать про коректну реалізацію функцій адміністративної панелі, конструктора, головної сторінки, а також процесів автентифікації та реєстрації. Успішне проходження всіх тестів дає змогу з упевненістю стверджувати, що поточна версія системи готова до подальшого використання та інтеграції.

Висновки до розділу 4

Розділ охопив практичні аспекти розгортання системи: створення ізольованого віртуального середовища, встановлення залежностей, налаштування SQLite-бази даних. Продемонстровано ключові сценарії роботи через інтерфейс (реєстрація, авторизація, конструювання, перегляд продуктів) та адміністративну панель. Автоматизовані Selenium-тести (5 кейсів) пройшли успішно, підтвердивши стабільність роботи основних функціональних блоків. Система готова до експлуатації у реальному середовищі.

РОЗДІЛ 5 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

У цьому розділі буде проведено оцінювання основних характеристик для майбутнього вбудованого програмного продукту.

Також у цьому дослідженні показано різні варіанти реалізації для забезпечення найбільш коректної та оптимальної стратегії вибору, що має вплив на економічні фактори та сумісність з майбутнім програмним продуктом. Для цього застосовувався апарат функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) передбачає собою технологію, що дає змогу оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії. ФВА проводиться з метою виявлення резервів зниження витрат за рахунок ефективніших варіантів виробництва, кращого співвідношення між споживчою вартістю виробу та витратами на його виготовлення. Для проведення аналізу використовується економічна, технічна та конструкторська інформація.

Алгоритм функціонально-вартісного аналізу включає в себе визначення послідовності етапів розробки продукту, визначення повних витрат (річних) та кількості робочих часів, визначення джерел витрат та кінцевий розрахунок вартості програмного продукту.

5.1 Постановка задачі проектування

У роботі застосовується метод ФВА для проведення техніко-економічного аналізу розробки системи прогнозу стійкості фінансових показників. Оскільки рішення стосовно проектування та реалізації

компонентів, що розробляється, впливають на всю систему, кожна окрема підсистема має її задовольняти. Тому фактичний аналіз представляє собою аналіз функцій програмного продукту, призначеного для збору, обробки та проведення аналізу даних по компанії.

Технічні вимоги до програмного продукту є наступні:

- функціонування на персональних комп'ютерах із стандартним набором компонентів;
- зручність та зрозумілість для користувача;
- швидкість обробки даних та доступ до інформації в реальному часі;
- можливість зручного масштабування та обслуговування;
- мінімальні витрати на впровадження програмного продукту.

5.2 Обґрунтування функцій програмного продукту

Головна функція F_0 – розробка можливого програмного продукту, яка дає змогу аналізувати різні характеристики, що безпосередньо впливають на стійкість підприємства. Беручи за основу цю функцію, можна виділити наступні:

F_1 – вибір типу мікроконтролера,

F_2 – вибір інтегрованого середовища розробки,

F_3 – вибір датчика акселерометра для приладу.

Кожна з цих функцій має декілька варіантів реалізації:

Функція F_1:

а) PYTHON;

б) FLASK;

в) Html.

Функція F_2:

а) Visual Studio;

б) PYTHON IDE.

Функція F_3:

а) основний;

б) в составі повноцінної системи.

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 5.1).

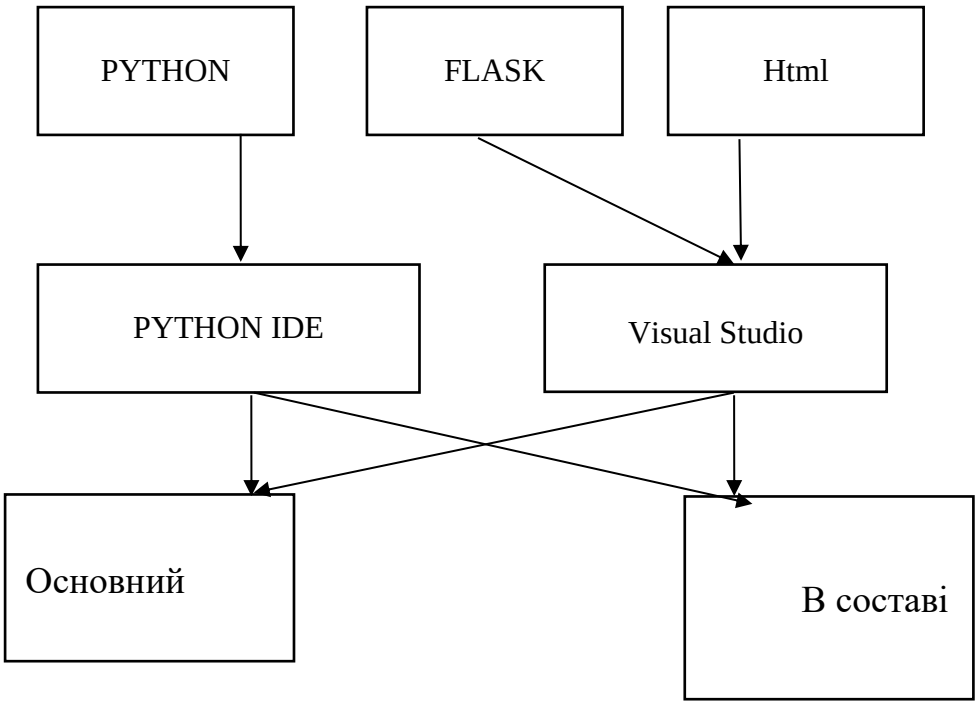

Рисунок 5.1 – Морфологічна карта

Морфологічна карта відображає множину всіх можливих варіантиів основних функцій. Позитивно-негативна матриця показана в табл. 5.1.

Таблиця 5.1 - Позитивно-негативна матриця

Функція	*Варіант*	*Переваги*	*Недоліки*
F_1	А – Python	– Широке застосування у Data Science і NLP – Велика кількість готових бібліотек (Transformers, SQLAlchemy) – Крос-платформеність	– Потребує окремого веб-фреймворку для HTTP
	Б – Flask	– Легкий мікрофреймворк для REST API – Мінімалістичний і розширюваний	– Не є повноцінною мовою: потребує доповнення Python-кодом
	В – HTML	– «з коробки» для фронтенду – Простий для верстки інтерфейсів	– Статичний: без логіки аналізу і збереження даних
F_2	А – Visual Studio	– Потужні інструменти налагодження – Інтеграція з .NET, C/C++	– Переважно Windows-орієнтоване – Важке для Python-проектів
	Б – Python IDE	– Спеціалізовані плагіни для Django/Flask – Крос-платформенність (PyCharm, VS Code)	– Менше вбудованих DevOps-інструментів ніж у VS Studio
F_3	А – Основний модуль	– Швидкий запуск, мінімальні залежності – Легко оновлювати та тестувати	– Обмежений набір фіч, потребує окремого деплою
	Б – У складі повноцінної системи	– Повний набір аналітики, адміністрування і звітів – Єдина точка доступу для всіх користувачів	– Складніший деплой, довша підтримка і супровід

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому, що вони не відповідають поставленим перед програмним продуктом задачам. Ці варіанти відзначені у морфологічній карті.

Функція F_1. Вибір технології реалізації серверної логіки та NLP-модуля

Перевагу віддаємо класичному варіанту – Python. Незважаючи на те, що сам по собі Python не є веб-фреймворком і потребує додаткової оболонки для прийому HTTP-запитів, він відкриває доступ до величезної екосистеми бібліотек для обробки природної мови (Transformers, spaCy, NLTK) та роботи з базами даних (SQLAlchemy). У нашому проєкті для взаємодії з клієнтом ми

використовуємо окремий модуль DT-06, який забезпечує готовий HTTP-сервер і Web-інтерфейс налаштування — це дає змогу швидко стартувати без необхідності писати низькорівневий код. Якби ми обрали Flask як мовний варіант, довелося б вручну писати більшість обгорток для токенізації, обміну даними та масштабування, натомість з Python ми зосереджуємося на самому алгоритмі аналізу та логіці роботи. Варіант HTML (статичні сторінки) не розглядається через відсутність динамічної логіки й неможливість безпосередньо інтегрувати модель NLP. Обраний варіант: А – Python.

Функція F_2. Вибір середовища розробки. Проєкт реалізується на GNU/Linux, тому відмовилися від Visual Studio (хоча воно й потужне, але орієнтоване переважно на Windows і .NET). Натомість обрали спеціалізоване Python IDE (наприклад, PyCharm або VS Code із розширеннями для Python/Flask). Це середовище забезпечує вбудовані засоби налагодження, підтримку віртуальних оточень, інтеграцію з Git та Docker, автоматичний рефакторинг коду і підсвітку синтаксису для Jinja-шаблонів. Завдяки цьому розробка й тестування як бекенду, так і фронтенду йде значно швидше, а синхронізація з CI/CD-пайплайнами налаштована «з коробки».

Обраний варіант: Б – Python IDE.

Функція F_3. Вибір режиму інтеграції аналітичного модуля

У проєкті є два підходи до публікації модуля аналізу настроїв: як окремий модуль (легко розгортати, тестувати й оновлювати) або як частину повноцінної системи з єдиним інтерфейсом. Другий варіант відкриває більше можливостей для адміністрування, аудиту результатів та розширених звітів, але ускладнює деплой і потребує складнішої інфраструктури. Оскільки наша мета – швидко прототипувати й отримувати якісні результати аналізу без довгого збирання контейнерів, ми обираємо «Основний модуль», який запускається самостійно та взаємодіє з фронтенд-частиною через REST API.

Обраний варіант: А – Основний модуль F_1а – F_2б – F_3а.

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

5.3 Обґрунтування системи параметрів програмного продукту

На основі даних, розглянутих вище, визначаються основні параметри вибору, які будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

- *X1* – потенційний об’єм програмного коду
- *X2* – об’єм RAM-пам’яті що потребує програма;
- *X3* – об’єм FLASH-пам’яті що займає програма;
- *X4* – час що займає один цикл програми

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію програмного продукту, як показано у табл. 5.2.

Таблиця 5.2 - Основні параметри програмного продукту

Назва Параметра	*Умовні позна-чення*	*Одиниці виміру*	*Значення параметра*		
			гірші	*середні*	*кращі*
Точність рекомендацій моделі (%)	X1	%	75	85	92
Час обробки одного запиту	X2	мс	250	180	120
Обсяг оперативної пам’яті, що споживає модель під час запиту	X3	МБ	800	600	400
Розмір дискового образу моделі	X4	МБ	1000	700	500

За даними табл. 5.2 будуються графічні характеристики параметрів – рис. 5.2 – рис. 5.5.

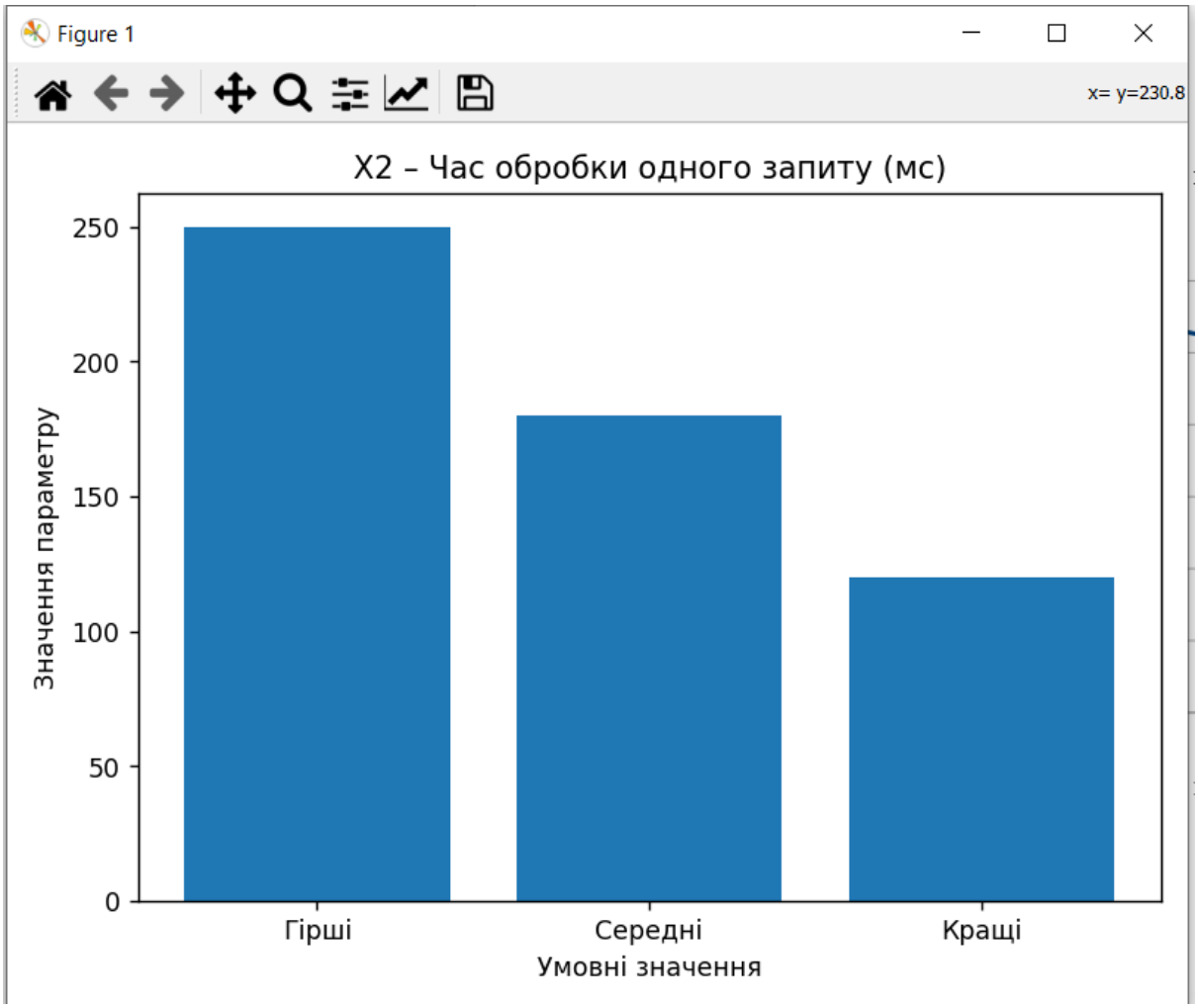

Рисунок 5.2 – X1 Точність рекомендацій моделі (%)

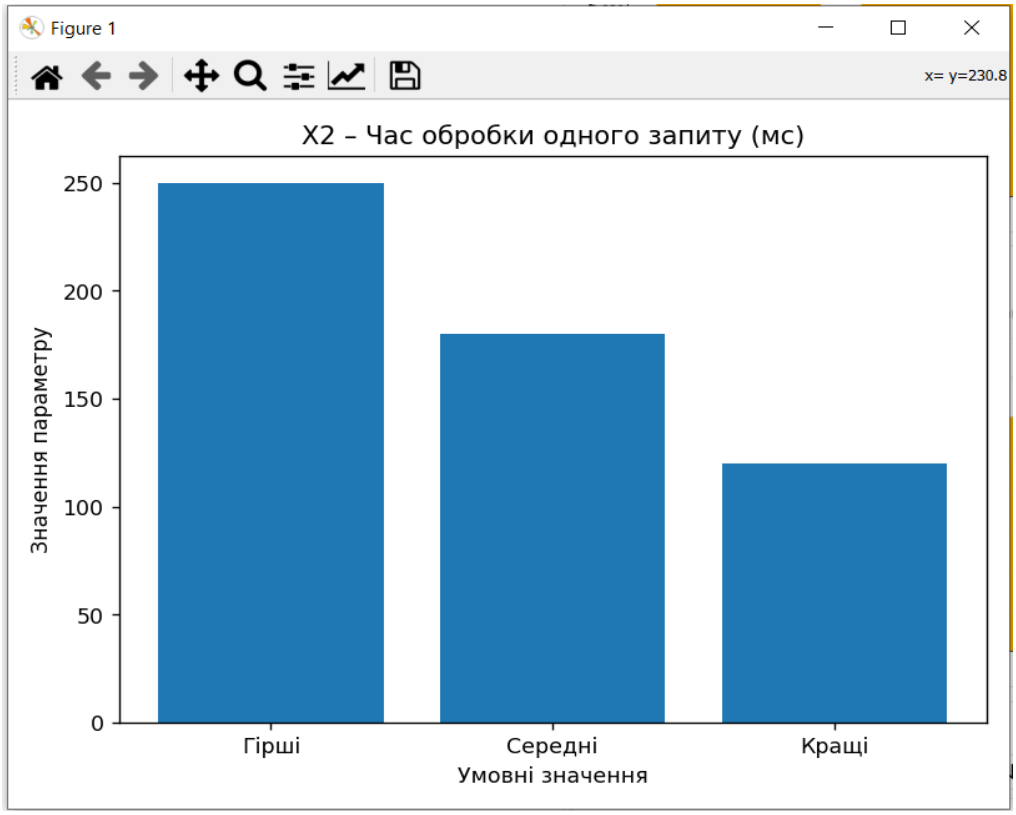

Рисунок 5.3 – X2 Час обробки одного запиту (мс)

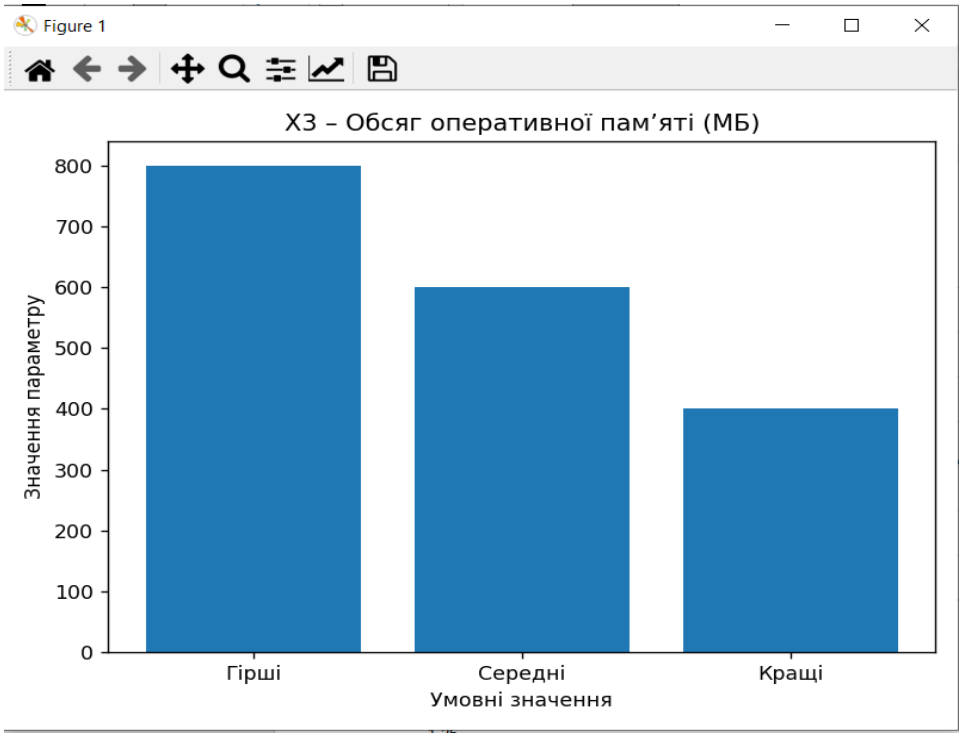

Рисунок 5.4 – X3 Обсяг оперативної пам’яті (МБ)

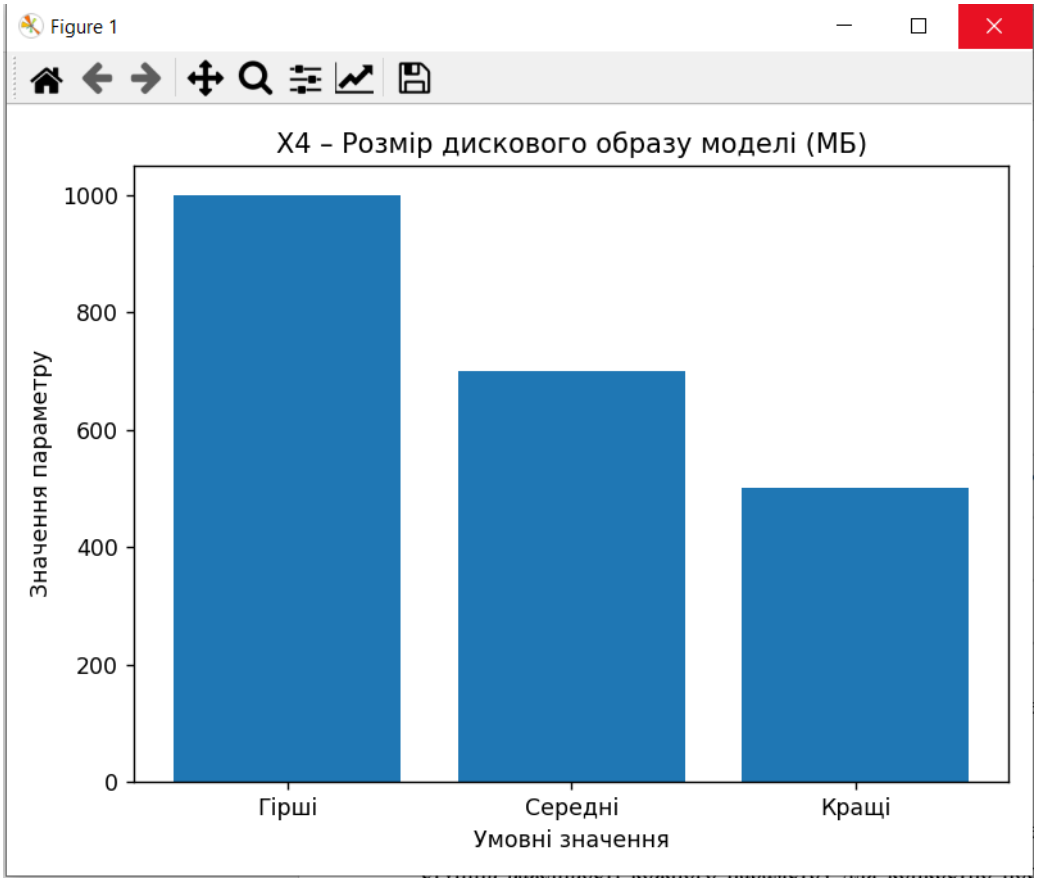

Рисунок 5.5 – X4 Розмір дискового образу моделі (МБ)

5.4 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка програмного продукту, який дає найбільш точні результати при знаходженні параметрів моделей адаптивного прогнозування і обчислення прогнозних значень.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із семи осіб. Визначення коефіцієнтів значущості передбачає:

- визначення рівня значимості параметра шляхом присвоєння різних рангів;
- перевірку придатності експертних оцінок для подальшого використання;
- визначення оцінки попарного пріоритету параметрів;
- обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 5.3.

Таблиця 5.3 - Результати ранжування параметрів

Позначення параметра	*Назва параметра*	*Одиниці виміру*	*Ранг параметра за оцінкою експерта*							*Сума рангів* R_i	*Відхилення* Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
X1	Точність рекомендацій моделі (%)	%	2	3	2	2	2	3	2	16	–1,5	2,25
X2	Час обробки одного запиту (мс)	мс	4	2	4	3	4	2	4	23	+5,5	30,25
X3	Обсяг оперативної пам’яті під запит (МБ)	МБ	4	3	4	3	4	3	24	+6,5	42,25	3
X4		Розмір дискового образу моделі (МБ)	3	4	3	4	3	4	3	24	+6,5	42,25
	Разом		10	10	10	10	10	10	10	70	0	185

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів (формула 5.1):

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 70, \quad (5.1)$$

де N – число експертів,

n – кількість параметрів;

б) середня сума рангів (формула 5.2):

$$T = \frac{1}{n} R_{ij} = 17{,}5; \quad (5.2)$$

в) відхилення суми рангів кожного параметра від середньої суми рангів (формула 5.3):

$$\Delta_i = R_i - T, \quad (5.3)$$

сума відхилень по всім параметрам повинна дорівнювати 0;

г) загальна сума квадратів відхилення (формула 5.4):

$$S = \sum_{i=1}^{N} \Delta_i^2 = 185. \quad (5.4)$$

Порахуємо коефіцієнт узгодженості (формула 5.5):

$$W = \frac{12S}{N^2(n^3-n)} = \frac{12 \cdot 185}{7^2(4^3-4)} = 0{,}755 > W_k = 0{,}698. \quad (5.5)$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у табл. 5.4.

Таблиця 5.4 – Попарне порівняння параметрів

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	<	<	<	<	<	<	<	<	3
X1 і X3	<	>	>	<	>	>	>	>	0,5
X1 і X4	<	<	<	<	<	<	<	<	0,5
X2 і X3	>	>	>	>	>	>	>	>	1,5
X2 і X4	<	>	<	<	>	<	<	>	0,5
X3 і X4	<	<	<	<	<	<	<	<	0,5

Числове значення, що визначає ступінь переваги *i*–го параметра над *j*–тим, a_{ij} визначається за формулою 5.6:

$$a_{ij} = \{1.5 \text{при} X_i > X_j 1.0 \text{при} X_i = X_j 0.5 \text{при} X_i < X_j. \quad (5.6)$$

З отриманих числових оцінок переваги складемо матрицю А=$\| a_{ij} \|$.

Для кожного параметра зробимо розрахунок вагомості $K_{ві}$ за наступними формулами 5.7 – 5.8:

$$K_{\text{ві}} = \frac{b_i}{\sum_{i=1}^{n} b_i}, \quad (5.7)$$

$$b_i = \sum_{i=1}^{N} a_{ij}. \quad (5.8)$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%). На другому і наступних кроках відносні оцінки розраховуються за наступними формулами 5.9 – 5.10:

$$K_{\text{ві}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'}, \quad (5.9)$$

$$b_i' = \sum_{i=1}^{N} a_{ij} b_{j.} \quad (5.10)$$

Як видно з табл. 5.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 5.5 – Розрахунок вагомості параметрів

Параметри x_i	Параметри x_j				Перша ітер.		Друга ітер.		Третя ітер.	
	X1	X2	X3	X4	b_i	$K_{вi}$	b_i^1	$K_{вi}^1$	b_i^2	$K_{вi}^2$
X1	1.0	1.5	1.5	1.5	5.50	0.344	21.25	0.360	77.875	0.360
X2	0.5	1.0	1.5	1.5	4.50	0.281	16.25	0.275	59.125	0.274
X3	0.5	0.5	1.0	1.	3.50	0.219	12.25	0.207	44.875	0.208
X4	0.5	0.5	0.5	1.0	2.50	0.156	9.25	0.157	34.125	0.158
Всього:					16.00	1.000	59.00	1.000	216.00	1.000

5.5 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів *X2* (Об’єм RAM-пам’яті що потребує програма), *X3* (Об’єм FLASH-пам’яті що займає програма) та *X4* (Час що займає один цикл програми) відповідають технічним вимогам умов функціонування даного ПП. Абсолютне значення параметра *X1* (Потенційний об’єм програмного коду) обрано не найгіршим.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (табл. 5.6):

$$K_K(j) = \sum_{i=1}^{n} K_{вi,j} B_{i,j} \quad (5.11)$$

визначаємо рівень якості кожного з двох варіантів:

де n – кількість параметрів;

$K_{вi}$– коефіцієнт вагомості i–го параметра;

B_i – оцінка i–го параметра в балах.

Таблиця 5.6 – Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	*Варіант реалізації*	*Параметр*	*Абсолютне значення*	*Бальна оцінка*	*Коефіцієнт вагомості*	*Коефіцієнт рівня якості*
F_1	А	X_1 – точність рекомендацій моделі (%)	85	3	0,344	1,032
F_2	Б	X_2 – час обробки одного запиту (мс)	180	3	0,281	0,843
		X_3 – обсяг оперативної пам’яті (МБ)	600	3	0,219	0,657
F_3	А	X_4 – розмір дискового образу моделі (МБ)	700	3	0,156	0,468

За даними з табл. 5.5 за формулою 5.12:

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \ldots + K_{\text{ТУ}}[F_{zk}] \qquad (5.12)$$

визначаємо рівень якості кожного з варіантів:

$$K_{K1} = 4 + 8{,}12 + 8{,}28 = 20{,}4 ;$$
$$K_{K2} = 4 + 3{,}8 + 8{,}28 = 16{,}08 .$$

Як видно з розрахунків, кращим є 2 варіант, для якого коефіцієнт технічного рівня має найбільше значення.

5.6 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти охоплюють два окремих завдання.

1. Розробка вбудованого програмного забезпечення.

2. Розробка допоміжної програми для візуалізації.

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 3; а в завданні 2 – до групи 1.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Загальна трудомісткість обчислюється як:

$$T_О = T_Р \cdot К_П \cdot К_{СК} \cdot К_М \cdot К_{СТ} \cdot К_{СТ.М}, \quad (5.13)$$

де $T_Р$ – трудомісткість розробки ПП;

$К_П$ – поправочний коефіцієнт;

$К_{СК}$ – коефіцієнт на складність вхідної інформації;

$К_М$ – коефіцієнт рівня мови програмування;

$К_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;

$К_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: $T_Р$ =45 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $К_П$ = 1.8. Поправочний коефіцієнт, який враховує складність контролю вхідної

та вихідної інформації для всіх семи завдань рівний 1: $К_{СК} = 1$. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $К_{СТ} = 0.9$. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$Т_1 = 45 \cdot 1.8 \cdot 0.9 = 72,9 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $Т_Р = 25$ людино-днів, $К_П = 0.9$, $К_{СК} = 1$, $К_{СТ} = 0.8$:

$$Т_2 = 25 \cdot 0.9 \cdot 0.8 = 18 \text{ людино-днів.}$$

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$$Т_I = (72,9+18,0+4,8+20,88) \cdot 8 = 932,64 \text{ людино-годин.}$$

$$Т_{II} = (72,9+18,0+6,91+20,88) \cdot 8 = 949,52 \text{людино-годин.}$$

Найбільш високу трудомісткість має варіант II.

В розробці бере участь один програміст з окладом 12000 грн. Визначаємо середню зарплату за годину за формулою 5.14:

$$СЧ = \frac{М}{T_m \cdot t} \text{ грн,} \qquad (5.14)$$

де М – місячний оклад працівників;

T_m – кількість робочих днів тиждень;

t – кількість робочих годин в день.

$$С_ч = \frac{12000}{3 \cdot 21 \cdot 8} = 23,81 грн. \quad (5.15)$$

Тоді, розраховуємо заробітну плату за формулою:

$$С_{зп} = С_ч \cdot Т_i \cdot КД, \quad (5.16)$$

де $С_ч$– величина погодинної оплати праці програміста;

$Т_i$ – трудомісткість відповідного завдання;

$К_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $С_{ЗП} = 23,81 \times 932,64 \cdot 1.2 = 26667,2$ грн.

II. $С_{ЗП} = 23,81 \times 949,52 \cdot 1.2 = 27131,0$ грн.

Відрахування на єдиний соціальний внесок становить 22%:

I. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 26667,2 \cdot 0.22 = 5867$грн.

II. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 27131,0 \cdot 0.22 = 5969$грн.

Тепер визначимо витрати на оплату однієї машино-години. ($С_М$)

Так як одна ЕОМ обслуговує одного програміста з окладом 40000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 12000 \cdot 0,2 = 28800$ грн.

З урахуванням додаткової заробітної плати:

$$С_{ЗП} = С_Г \cdot (1 + К_З) = 28800 \cdot (1 + 0.2) = 34560 \text{ грн.}$$

Відрахування на соціальний внесок:

$$С_{ВІД} = С_{ЗП} \cdot 0.22 = 34560 \cdot 0,22 = 7603,20 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 16000 грн.

$$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.4 \cdot 0.25 \cdot 16000 = 5600 \text{ грн.,}$$

де $К_{ТМ}$– коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

$К_А$– річна норма амортизації;

$Ц_{ПР}$– договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.4 \cdot 16000 \cdot 0.08 = 1792 \text{ грн.,}$$

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$Т_{ЕФ} = (Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 104 – 12 – 16) \cdot 8 \cdot 0.40 = 745,6 \text{ години,}$$

де $Д_К$ – календарна кількість днів у році;

$Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів;

$Д_Р$ – кількість днів планових ремонтів устаткування;

t –кількість робочих годин в день;

$К_В$– коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 745,6 \cdot 0,4 \cdot 0,2 \cdot 9,43 = 562,48 \text{ грн.},$$

де $N_С$ – середньо-споживча потужність приладу;

$К_З$– коефіцієнтом зайнятості приладу;

$Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$С_Н = Ц_{ПР} \cdot 0.67 = 16000 \cdot 0,67 = 10720 \text{ грн.}$$

Тоді, річні експлуатаційні витрати будуть(формула 5.17):

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н \qquad (5.17)$$

$$С_{ЕКС} = 34560+7603,20+5600+1792+562,48+10720 = 60837.68 \text{ грн}$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$С_{М\text{-}Г} = С_{ЕКС} / Т_{ЕФ} = 60837.68 / 745,6 = 81.57 \text{ грн/год.}$$

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає(формула 5.18):

$$С_М = С_{М-Г} \cdot T \qquad (5.18)$$

I. $С_М = 81.57 \times 932,64 \approx 76068.3$ грн.

II. $С_М = 81.57 \times 949,52 \approx 77464.7$ грн.

Накладні витрати складають 67% від заробітної плати (формула 5.19):

$$С_Н = С_{ЗП} \cdot 0,67 \quad (5.19)$$

I. $С_Н$ = 26667,2×0,67= 17867.8грн.

II. $С_Н$ = 27131,0×0,67= 18197.8грн.

Отже, вартість розробки ПП за варіантами становить(формула 5.20):

$$С_{ПП} = С_{ЗП} + С_{ВІД} + С_М + С_Н \quad (5.20)$$

I. $С_{ПП}$ = 26667.2+5867+76068.3+17867.8≈126470.3грн.

II. $С_{ПП}$ = 27131.0+5969+77464.7+18197.8≈128762.5 грн.

5.7 Вибір кращого варіанту ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою 5.21:

$$К_{ТЕР j} = К_{К j} / С_{Ф j} \quad (5.21)$$

$К_{ТЕР1}$ = 20,4 / 126470.3=0,000161,

$К_{ТЕР2}$ = 16,08 / 128762.5= 0,000125.

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1}$= 0,000161.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є перший варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості $К_{ТЕР}$ = 0,000161.

Цей варіант виконання програмного комплексу гарантує швидку та не ресурсомістку реалізацію програмного забезпечення для проекту.

Висновки до розділу 5

У цій частині дипломної роботи було проведено докладний аналіз функціональних та вартісних аспектів програмного продукту. Також було оцінено основні функції цього програмного продукту.

В результаті аналізу функціональності та вартості програмного комплексу, який розробляється, було визначено та проаналізовано його ключові функції, а також ідентифіковано параметри, що характеризують його. На основі проведеного аналізу був обраний варіант реалізації програмного продукту.

ВИСНОВКИ

На основі аналізу підходів, методів та алгоритмів розв’язання поставленої задачі обґрунтовано застосування сучасних трансформерних моделей для обробки природної мови. Проведений огляд дозволив встановити, що моделі типу BERT, RoBERTa, ALBERT та DistilBERT мають високий потенціал для класифікації тональності текстових відгуків. Однак експериментальні дослідження свідчать про те, що завдяки механізмам розділення уваги та глибокому контекстуальному аналізу модель DistilBERT демонструє найвищу точність, що робить її оптимальним вибором для даної задачі.

Аналіз програмних засобів, що використовуються у системі, обґрунтовує застосування мови Python, середовища розробки PyCharm, веб-фреймворку Flask, бази даних SQLite у поєднанні з ORM SQLAlchemy, а також бібліотек для роботи з трансформерами. Такий набір технологій дає змогу забезпечити ефективну реалізацію серверної логіки, зручну інтеграцію з фронтендом та масштабованість системи. Всі обрані засоби мають високий рівень підтримки та документованості, що сприяє швидкому впровадженню та подальшому супроводу розробленого застосунку.

Розроблено програмну систему «Quad Parts Store» для аналізу настроїв клієнтів у відгуках про складові FPV-систем. Система інтегрує веб-інтерфейс для взаємодії з користувачами, модуль роботи з користувачами (реєстрація, авторизація, управління сесіями), модуль управління товарами та замовленнями, а також модуль аналізу настроїв, що використовує попередньо навчений алгоритм (з використанням моделі DeBERTa). Такий підхід дає змогу автоматично оцінювати емоційне забарвлення відгуків, що є важливим для прийняття рішень щодо оптимізації продукції та покращення якості обслуговування.

На основі тестування розробленої системи доведено коректність її роботи. Експериментальні результати свідчать про високий рівень точності

класифікації текстових відгуків та ефективність обробки даних як на фронтенді, так і на серверній частині. Інтеграція алгоритму аналізу настроїв з базою даних і веб-інтерфейсом дає змогу оперативно реагувати на зміни у відгуках клієнтів, що підтверджує практичну користь розробленого рішення.

Отримані результати засвідчують високий практичної значущість проведеного дослідження. Система має потенціал для подальшого вдосконалення, зокрема через оптимізацію алгоритмів аналізу, розширення функціональних можливостей веб-інтерфейсу та інтеграцію додаткових модулів для підтримки нових бізнес-процесів. Запропонований підхід сприятиме підвищенню якості обслуговування клієнтів та може бути адаптований до інших доменів застосування FPV-технологій.

ПЕРЕЛІК ПОСИЛАНЬ

1. Devlin, J., Chang, M.-W., Lee, K., Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding // arXiv. – Режим доступу: https://arxiv.org/abs/1810.04805 (дата звернення: 10.01.2025).

2. Hugging Face. Transformers Documentation. – Режим доступу: https://huggingface.co/transformers (дата звернення: 15.01.2025).

3. Hugging Face. DistilBERT. – Режим доступу: https://huggingface.co/distilbert-base-uncased (дата звернення: 15.01.2025).

4. SentiWordNet 3.0 // SentiWordNet. – Режим доступу: http://sentiwordnet.isti.cnr.it (дата звернення: 15.01.2025).

5. Amazon Polarity Dataset // Hugging Face Datasets. – Режим доступу: https://huggingface.co/datasets/amazon_polarity (дата звернення: 15.01.2025).

6. Pedregosa, F. et al. Scikit-learn: Machine Learning in Python // Journal of Machine Learning Research, 2011. – Режим доступу: https://scikit-learn.org (дата звернення: 18.01.2025).

7. Devlin J., Chang M.-W., Lee K., Toutanova K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding // arXiv [Електронний ресурс]. Режим доступу: https://arxiv.org/abs/1810.04805 (дата звернення: 18.01.2025).

8. Sanh V., Debut L., Chaumond J., Wolf T. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter // arXiv [Електронний ресурс]. Режим доступу: https://arxiv.org/abs/1910.01108 (дата звернення: 30.01.2025).

9. Liu Y., Ott M., Goyal N. et al. RoBERTa: A Robustly Optimized BERT Pretraining Approach // arXiv [Електронний ресурс]. Режим доступу: https://arxiv.org/abs/1907.11692 (дата звернення: 30.01.2025).

10. Lan Z., Chen M., Goodman S., Gimpel K., Sharma P., Soricut R. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations

// arXiv [Електронний ресурс]. Режим доступу: https://arxiv.org/abs/1909.11942 (дата звернення: 30.01.2025).

11. Python Software Foundation. Python Language Reference // Python [Електронний ресурс]. Режим доступу: https://www.python.org (дата звернення: 31.01.2025).

12. JetBrains. PyCharm // JetBrains [Електронний ресурс]. Режим доступу: https://www.jetbrains.com/pycharm (дата звернення: 31.01.2025).

13. World Wide Web Consortium (W3C). HTML5 Specification // W3C [Електронний ресурс]. Режим доступу: https://www.w3.org/TR/html5/ (дата звернення: 31.01.2025).

14. Bootstrap. Bootstrap Documentation // Bootstrap [Електронний ресурс]. Режим доступу: https://getbootstrap.com (дата звернення: 31.01.2025).

15. Mozilla Developer Network. JavaScript Guide // MDN [Електронний ресурс]. Режим доступу: https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата звернення: 01.02.2025).

16. SQLite Consortium. SQLite Home Page // SQLite [Електронний ресурс]. Режим доступу: https://www.sqlite.org (дата звернення: 01.02.2025).

17. Pallets Projects. Flask Documentation // Flask [Електронний ресурс]. Режим доступу: https://flask.palletsprojects.com (дата звернення: 01.02.2025).

18. SeleniumHQ. Selenium Documentation // Selenium [Електронний ресурс]. Режим доступу: https://www.selenium.dev (дата звернення: 05.02.2025).

19. Python Software Foundation. venv — Creation of Virtual Environments // Python [Електронний ресурс]. Режим доступу: https://docs.python.org/3/library/venv.html (дата звернення: 05.02.2025).

ДОДАТОК А ЛІСТИНГ ПРОГРАМИ

app.py

```
import os
import json
from datetime import datetime
from flask import Flask, render_template, url_for, send_from_directory,
abort, request, redirect, flash
from flask_sqlalchemy import SQLAlchemy
from flask_login import LoginManager, UserMixin, login_user, logout_user,
login_required, current_user
from werkzeug.security import generate_password_hash, check_password_hash
import torch
from transformers import AutoTokenizer, AutoModelForSequenceClassification

app = Flask(__name__)
app.config['SECRET_KEY'] = 'your_secret_key_here'
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///store.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

db = SQLAlchemy(app)
login_manager = LoginManager(app)
login_manager.login_view = "login"


################################################################################
# MODELS
################################################################################

class User(db.Model, UserMixin):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(80), unique=True, nullable=False)
    email = db.Column(db.String(120), unique=True, nullable=False)
    password_hash = db.Column(db.String(128), nullable=False)
    is_admin = db.Column(db.Boolean, default=False)

    def set_password(self, password):
        self.password_hash = generate_password_hash(password)

    def check_password(self, password):
        return check_password_hash(self.password_hash, password)


class Contact(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(120))
    email = db.Column(db.String(120))
    message = db.Column(db.Text)
    timestamp = db.Column(db.DateTime, default=datetime.utcnow)


class Comment(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    content = db.Column(db.Text, nullable=False)
    product_title = db.Column(db.String(200))  # Використовуємо назву товару
    timestamp = db.Column(db.DateTime, default=datetime.utcnow)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'))
```

```
class Product(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    product_title = db.Column(db.String(200), nullable=False)
    category = db.Column(db.String(100), nullable=False)
    price = db.Column(db.String(50))
    details = db.Column(db.Text)  # Зберігаємо JSON як рядок (якщо додається
адміністратором)
    specifications = db.Column(db.Text)
    image_path = db.Column(db.String(200))  # Шлях до зображення


class Order(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=True)
    timestamp = db.Column(db.DateTime, default=datetime.utcnow)
    total_price = db.Column(db.Float, default=0.0)
    status = db.Column(db.String(50), default="Pending")
    builder_data = db.Column(db.Text)  # JSON рядок із даними замовлення


class OrderItem(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    order_id = db.Column(db.Integer, db.ForeignKey("order.id"),
nullable=False)
    product_title = db.Column(db.String(200))
    quantity = db.Column(db.Integer)
    base_price = db.Column(db.Float)
    total_price = db.Column(db.Float)


###############################################################################
# SENTIMENT ANALYSIS MODEL LOAD
###############################################################################

sentiment_model_path = "sentiment_model"
sentiment_tokenizer = AutoTokenizer.from_pretrained(sentiment_model_path)
sentiment_model =
AutoModelForSequenceClassification.from_pretrained(sentiment_model_path)


# Контекст-процесор для Jinja, який додає функцію get_sentiment, що повертає
смайлик залежно від настрою
@app.context_processor
def utility_processor():
    def get_sentiment(review):
        inputs = sentiment_tokenizer(
            review,
            return_tensors="pt",
            truncation=True,
            padding="max_length",
            max_length=128
        )
        outputs = sentiment_model(**inputs)
        predicted_class = outputs.logits.argmax(dim=1).item()
        # Повертаємо 1 для позитивного, 0 для негативного
        return predicted_class

    def get_product_rating(product_title):
        # Отримуємо всі коментарі для даного товару
        comments = Comment.query.filter_by(product_title=product_title).all()
        if not comments:
            return 0  # Якщо коментарів немає, повертаємо 0
        ratings = [get_sentiment(comment.content) for comment in comments]
```

```
        avg = sum(ratings) / len(ratings)
        # Розподіл за середнім: <0.2 -> 1 зірка, <0.4 -> 2 зірки, <0.6 -> 3,
<0.8 -> 4, інакше 5
        if avg < 0.2:
            return 1
        elif avg < 0.4:
            return 2
        elif avg < 0.6:
            return 3
        elif avg < 0.8:
            return 4
        else:
            return 5

    return dict(get_sentiment=get_sentiment,
get_product_rating=get_product_rating)



################################################################################
# LOGIN MANAGER
################################################################################

@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))


################################################################################
# ФУНКЦІЇ ЗАВАНТАЖЕННЯ ПРОДУКТІВ З ФАЙЛОВОЇ СИСТЕМИ
################################################################################

def load_products_from_fs():
    """Завантажуємо продукти з JSON файлів, що знаходяться поруч із
зображеннями в папках за категоріями."""
    products = []
    base_folder = "photo"
    for category in os.listdir(base_folder):
        category_path = os.path.join(base_folder, category)
        if os.path.isdir(category_path):
            for file in os.listdir(category_path):
                if file.endswith(".json"):
                    file_path = os.path.join(category_path, file)
                    if os.stat(file_path).st_size == 0:
                        continue
                    try:
                        with open(file_path, "r", encoding="utf-8") as f:
                            data = json.load(f)
                    except json.decoder.JSONDecodeError:
                        continue
                    data["category"] = category
                    image_path = data.get("image_path") or
file.replace(".json", ".png")
                    data["image_url"] = url_for("photos",
filename=f"{category}/{os.path.basename(image_path)}")
                    products.append(data)
    return products


def group_products_by_category(products):
    grouped = {}
    for prod in products:
        cat = prod.get("category", "Uncategorized")
```

```
        grouped.setdefault(cat, []).append(prod)
    return grouped


##############################################################################
# ROUTES ГОЛОВНОГО ДОДАТКУ
##############################################################################

@app.route("/")
def home():
    return render_template("home.html")


@app.route("/builder")
def builder():
    products = load_products_from_fs()
    products_grouped = group_products_by_category(products)
    categories = [
        "Frames",
        "Motors",
        "Propellers",
        "LiPo_LiHV_Batteries",
        "Flight_Controllers",
        "ESCs",
        "Video_Transmitters"
    ]
    return render_template("builder.html", categories=categories,
products_grouped=products_grouped)


@app.route("/contact", methods=["GET", "POST"])
def contact():
    if request.method == "POST":
        name = request.form.get("name")
        email = request.form.get("email")
        message = request.form.get("message")
        if name and email and message:
            contact_entry = Contact(name=name, email=email, message=message)
            db.session.add(contact_entry)
            db.session.commit()
            flash("Your message has been sent!", "success")
            return redirect(url_for("contact"))
        else:
            flash("Please fill all fields.", "danger")
    return render_template("contact.html")


@app.route("/category/<category>")
def category(category):
    products = load_products_from_fs()
    filtered = [p for p in products if p.get("category", "").lower() ==
category.lower()]
    if not filtered:
        abort(404)
    return render_template("category.html", category=category,
products=filtered)


@app.route("/product/<category>/<filename>")
def product_detail(category, filename):
    file_path = os.path.join("photo", category, f"{filename}.json")
    if not os.path.exists(file_path):
        abort(404)
```

```
    try:
        with open(file_path, "r", encoding="utf-8") as f:
            product = json.load(f)
    except json.decoder.JSONDecodeError:
        abort(500)
    product["category"] = category
    product["image_url"] = url_for('photos',
filename=f"/{product.get('image_path', f'{filename}.png')}")
    comments =
Comment.query.filter_by(product_title=product.get("product_title")).order_by(
        Comment.timestamp.desc()).all()
    return render_template("product.html", product=product,
comments=comments)


@app.route("/product/<category>/<filename>/comment", methods=["POST"])
@login_required
def add_comment_fs(category, filename):
    file_path = os.path.join("photo", category, f"{filename}.json")
    if not os.path.exists(file_path):
        abort(404)
    try:
        with open(file_path, "r", encoding="utf-8") as f:
            product = json.load(f)
    except json.decoder.JSONDecodeError:
        abort(500)
    product_title = product.get("product_title", "Unknown")
    content = request.form.get("comment")
    if content:
        comment = Comment(content=content, product_title=product_title,
user_id=current_user.id)
        db.session.add(comment)
        db.session.commit()
        flash("Comment added.", "success")
    else:
        flash("Comment cannot be empty.", "danger")
    return redirect(url_for("product_detail", category=category,
filename=filename))


@app.route("/photos/<path:filename>")
def photos(filename):
    return send_from_directory("photo", filename)


#############################################################################
# КОРЗИНА ТА ЗАМОВЛЕННЯ (Builder Order)
#############################################################################

@app.route("/order/submit", methods=["POST"])
@login_required
def submit_order():
    order_data = request.form.get("order_data")
    if not order_data:
        flash("No order data provided.", "danger")
        return redirect(url_for("builder"))
    try:
        order_items = json.loads(order_data)
    except Exception as e:
        flash("Invalid order data.", "danger")
        return redirect(url_for("builder"))
    total = 0
    for item in order_items:
```

```
        try:
            total += float(item.get("quantity", 0)) *
float(item.get("base_price", 0))
        except Exception:
            continue
    order = Order(user_id=current_user.id, total_price=total,
builder_data=order_data)
    db.session.add(order)
    db.session.commit()
    flash("Order submitted successfully!", "success")
    return redirect(url_for("builder"))


################################################################################
# АДМІНСЬКИЙ ДАШБОАРД та CRUD
################################################################################

@app.route("/admin")
@login_required
def admin_dashboard():
    if not current_user.is_admin:
        abort(403)
    users = User.query.all()
    contacts = Contact.query.order_by(Contact.timestamp.desc()).all()
    comments = Comment.query.order_by(Comment.timestamp.desc()).all()
    products_db = Product.query.order_by(Product.id.desc()).all()
    orders = Order.query.order_by(Order.timestamp.desc()).all()
    return render_template("admin/dashboard.html", users=users,
contacts=contacts, comments=comments,
                           products=products_db, orders=orders)



@app.route("/admin/delete_contact/<int:contact_id>", methods=["POST"])
@login_required
def delete_contact(contact_id):
    if not current_user.is_admin:
        abort(403)
    contact = Contact.query.get_or_404(contact_id)
    db.session.delete(contact)
    db.session.commit()
    flash("Contact deleted.", "success")
    return redirect(url_for("admin_dashboard"))


@app.route("/admin/delete_comment/<int:comment_id>", methods=["POST"])
@login_required
def delete_comment(comment_id):
    if not current_user.is_admin:
        abort(403)
    comment = Comment.query.get_or_404(comment_id)
    db.session.delete(comment)
    db.session.commit()
    flash("Comment deleted.", "success")
    return redirect(url_for("admin_dashboard"))


@app.route("/admin/delete_order/<int:order_id>", methods=["POST"])
@login_required
def delete_order(order_id):
    if not current_user.is_admin:
        abort(403)
    order = Order.query.get_or_404(order_id)
```

```
    db.session.delete(order)
    db.session.commit()
    flash("Order deleted.", "success")
    return redirect(url_for("admin_dashboard"))


@app.route("/admin/delete_user/<int:user_id>", methods=["POST"])
@login_required
def delete_user(user_id):
    if not current_user.is_admin:
        abort(403)
    user = User.query.get_or_404(user_id)
    if user.is_admin:
        flash("Cannot delete admin account.", "danger")
        return redirect(url_for("admin_dashboard"))
    db.session.delete(user)
    db.session.commit()
    flash("User deleted.", "success")
    return redirect(url_for("admin_dashboard"))


@app.route("/admin/edit_user/<int:user_id>", methods=["GET", "POST"])
@login_required
def edit_user(user_id):
    if not current_user.is_admin:
        abort(403)
    user = User.query.get_or_404(user_id)
    if request.method == "POST":
        user.username = request.form.get("username")
        user.email = request.form.get("email")
        db.session.commit()
        flash("User updated.", "success")
        return redirect(url_for("admin_dashboard"))
    return render_template("admin/edit_user.html", user=user)


@app.route("/admin/edit_product/<int:product_id>", methods=["GET", "POST"])
@login_required
def edit_product(product_id):
    if not current_user.is_admin:
        abort(403)
    product = Product.query.get_or_404(product_id)
    if request.method == "POST":
        product.product_title = request.form.get("product_title")
        product.category = request.form.get("category")
        product.price = request.form.get("price")
        product.details = request.form.get("details")
        product.specifications = request.form.get("specifications")
        product.image_path = request.form.get("image_path")
        db.session.commit()
        flash("Product updated.", "success")
        return redirect(url_for("admin_dashboard"))
    return render_template("admin/edit_product.html", product=product)


@app.route("/admin/delete_product/<int:product_id>", methods=["POST"])
@login_required
def delete_product(product_id):
    if not current_user.is_admin:
        abort(403)
    product = Product.query.get_or_404(product_id)
    db.session.delete(product)
    db.session.commit()
```

```
    flash("Product deleted.", "success")
    return redirect(url_for("admin_dashboard"))


@app.route("/admin/add_product", methods=["GET", "POST"])
@login_required
def add_product():
    if not current_user.is_admin:
        abort(403)
    if request.method == "POST":
        product_title = request.form.get("product_title")
        category = request.form.get("category")
        price = request.form.get("price")
        details = request.form.get("details")
        specifications = request.form.get("specifications")
        image_path = request.form.get("image_path")
        if product_title and category:
            product = Product(product_title=product_title, category=category,
                              price=price,
                              details=details,
                              specifications=specifications,
                              image_path=image_path)
            db.session.add(product)
            db.session.commit()
            flash("Product added successfully.", "success")
            return redirect(url_for("admin_dashboard"))
        else:
            flash("Please fill required fields.", "danger")
    return render_template("admin/add_product.html")


@app.route("/admin/orders")
@login_required
def admin_orders():
    if not current_user.is_admin:
        abort(403)
    orders = Order.query.order_by(Order.timestamp.desc()).all()
    return render_template("admin/orders.html", orders=orders)


##########################################################################
# АВТОРИЗАЦІЯ: РЕЄСТРАЦІЯ, ЛОГІН, ЛОГАУТ
##########################################################################

@app.route("/register", methods=["GET", "POST"])
def register():
    if request.method == "POST":
        username = request.form.get("username")
        email = request.form.get("email")
        password = request.form.get("password")
        if User.query.filter((User.username == username) | (User.email ==
email)).first():
            flash("User already exists.", "danger")
            return redirect(url_for("register"))
        user = User(username=username, email=email)
        user.set_password(password)
        db.session.add(user)
        db.session.commit()
        flash("Registration successful. Please login.", "success")
        return redirect(url_for("login"))
    return render_template("register.html")
```

```
@app.route("/login", methods=["GET", "POST"])
def login():
    if request.method == "POST":
        username_or_email = request.form.get("username_or_email")
        password = request.form.get("password")
        user = User.query.filter((User.username == username_or_email) |
(User.email == username_or_email)).first()
        if user and user.check_password(password):
            login_user(user)
            flash("Logged in successfully.", "success")
            return redirect(url_for("home"))
        else:
            flash("Invalid credentials.", "danger")
    return render_template("login.html")


@app.route("/logout")
@login_required
def logout():
    logout_user()
    flash("Logged out.", "success")
    return redirect(url_for("home"))


# ##############################################################################
# ІНІЦІАЛІЗАЦІЯ БД та створення admin-акаунту
# ##############################################################################

with app.app_context():
    db.create_all()
    if not User.query.filter_by(username="admin").first():
        admin = User(username="admin", email="admin@example.com",
is_admin=True)
        admin.set_password("admin")
        db.session.add(admin)
        db.session.commit()
        print("Admin account created: username 'admin', password 'admin'")

if __name__ == "__main__":
    app.run(debug=True)
```

train.py

```
# pip install transformers datasets evaluate

import numpy as np
import evaluate
from transformers import AutoTokenizer, AutoModelForSequenceClassification,
TrainingArguments, Trainer
from datasets import load_dataset  # Видалили load_metric


dataset = load_dataset("amazon_polarity")


print(dataset)


model_checkpoint = "distilbert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(model_checkpoint)
model = AutoModelForSequenceClassification.from_pretrained(model_checkpoint,
```

```
num_labels=2)


def tokenize_function(example):
    return tokenizer(example["content"], truncation=True,
padding="max_length", max_length=128)


tokenized_datasets = dataset.map(tokenize_function, batched=True)


tokenized_datasets = tokenized_datasets.remove_columns(["content"])
tokenized_datasets = tokenized_datasets.rename_column("label", "labels")
tokenized_datasets.set_format("torch")


train_dataset = tokenized_datasets["train"]
test_dataset = tokenized_datasets["test"]


# , використаємо лише 10 000 з 3.6M прикладів для тренування та 2 000 для
тестування
small_train_dataset = train_dataset.select(range(10000))
small_test_dataset = test_dataset.select(range(2000))

#  accuracy
accuracy_metric = evaluate.load("accuracy")

def compute_metrics(eval_pred):
    logits, labels = eval_pred
    predictions = np.argmax(logits, axis=-1)
    return accuracy_metric.compute(predictions=predictions,
references=labels)


training_args = TrainingArguments(
    output_dir="sentiment_model",
    evaluation_strategy="epoch",
    num_train_epochs=2,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=64,
    learning_rate=2e-5,
    weight_decay=0.01,
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=small_train_dataset,  # Використовуємо підмножину
тренувальних даних
    eval_dataset=small_test_dataset,      # Використовуємо підмножину
тестових даних
    compute_metrics=compute_metrics,
)


trainer.train()


eval_results = trainer.evaluate()
print("Evaluation results:", eval_results)
```

```
model.save_pretrained("sentiment_model")
tokenizer.save_pretrained("sentiment_model")
```