МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра інформаційних технологій

«До захисту допущено»
В.о. завідувача кафедри

_____ Світлана ВАЩЕНКО

____ _____ 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня магістр

зі спеціальності 122 «Комп'ютерні науки»,

освітньо-професійної програми «Інформаційні технології проектування»

на тему: "Інформаційна система підтримки обслуговування клієнтів інтернет-магазину одягу та аксесуарів з принтами. Віртуальний асистент"

Здобувача групи ІТ.м-24 Баранніка Дмитра Олександровича
(шифр групи) (прізвище, ім'я, по батькові)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело. _____ _____Дмитро БАРАННІК__
(підпис) (Ім'я та ПРІЗВИЩЕ здобувача)

Керівник к.т.н., доцент Анна НЕНЯ _____
(посада, науковий ступінь, вчене звання, Ім'я та ПРІЗВИЩЕ) (підпис)

Суми – 2023

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра інформаційних технологій
Спеціальність 122 «Комп'ютерні науки»
Освітньо-професійна програма «Інформаційні технології проектування»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри ІТ

Світлана ВАЩЕНКО
«_____» _____ 2023 р.

ЗАВДАННЯ

на кваліфікаційну роботу магістра студентові

Баранніку Дмитру Олександровичу

(прізвище, ім'я, по батькові)

1 Тема кваліфікаційної роботи *Інформаційна система підтримки обслуговування клієнтів інтернет-магазину одягу та аксесуарів з принтами. Віртуальний асистент*

затверджена наказом по університету від «08» листопада 2023 р. № 1249-VI

2 Термін здачі студентом кваліфікаційної роботи «14» грудня 2023 р.

3 Вхідні дані до кваліфікаційної роботи інтернет-магазин одягу та одягу та аксесуарів з принтами

4 Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) Аналіз методології створення віртуального асистента для інформаційної системи підтримки обслуговування клієнтів інтернет-магазину одягу та аксесуарів з принтами, постановка задачі, проектування, програмна реалізація.

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових слайдів презентації) актуальність, постановка задачі, задачі дослідження, огляд платформ для чат-ботів, таблиця порівняння платформ для чат-ботів, вимоги до віртуального асистента, інструменти реалізації, структурно-функціональне моделювання, діаграма варіантів використання, реалізована база даних, практична реалізація, налагодження моделі штучного інтелекту, демонстрація роботи, висновки.

6. Консультанти випускної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

Дата видачі завдання 14.11.2023.

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Аналіз та вибір інструментів реалізації	до 17.11.2023	
2	Розробка модулю чат-бота	до 24.11.2023	
3	Інтеграція чат-бота з інтернет-магазином	до 28.11.2023	
4	Налаштування штучного інтелекту	до 01.12.2023	
5	Тестування віртуального асистента	до 05.12.2023	
6	Оформлення документації	до 14.12.2023	

Магістрант _____ Дмитро БАРАННІК

Керівник роботи _____ к.т.н., доц. Анна НЕНЯ

АНОТАЦІЯ

Тема кваліфікаційної роботи магістра «Інформаційна система підтримки обслуговування клієнтів інтернет-магазину одягу та аксесуарів з принтами. Віртуальний асистент».

Пояснювальна записка складається зі вступу, 4 розділів, висновків, списку використаних джерел із 40 найменувань, додатків. Загальний обсяг роботи − 82 сторінок, у тому числі 57 сторінок основного тексту, 5 сторінок списку використаних джерел, 20 сторінок додатків

Актуальність роботи полягає в забезпеченні більш інтерактивного та зручного досвіду для клієнтів, допомозі в режимі реального часу та наданні персоналізованої рекомендації щодо продуктів, щоб покращити процес покупки, а також підвищенні об'єму продажів інтернет-магазинів та покращенні конкурентоспроможності бізнесу.

Мета роботи: розроблення віртуального асистента для інформаційної система підтримки обслуговування клієнтів інтернет-магазину одягу та аксесуарів з принтами за рахунок створення інтеграції з чат-ботом та впровадження штучного інтелекту.

Результатами роботи є визначення вимог та інструментів реалізації розробки віртуального асистента, аналіз та вибір технологій для його розробки, аналіз та вибір платформи чат-бота як інтерфейсу взаємодії користувача та віртуального асистента, розроблений віртуальний асистент для інформаційної системи підтримки клієнтів інтернет-магазину одягу та аксесуарів з принтами. Крім того, було впроваджено штучний інтелект і проведено тестування роботи віртуального асистента на базі тестових запитів від користувача . Розроблена система дозволяє значно покращити взаємодію з клієнтами, надаючи персоналізовану допомогу, спрощуючи взаємодію та пропонуючи цінну інформацію. Результати представлені у вигляді програмного коду для реалізації роботи віртуального асистента.

Ключові слова: віртуальний асистент, чат-бот, веб-застосунки, штучний інтелект, підтримка клієнтів.

ЗМІСТ

ВСТУП 6
1 АНАЛІЗ МЕТОДОЛОГІЇ СТВОРЕННЯ ВІРТУАЛЬНОГО АСИСТЕНТА ДЛЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ ПІДТРИМКИ ОБСЛУГОВУВАННЯ КЛІЄНТІВ ІНТЕРНЕТ-МАГАЗИНУ ОДЯГУ ТА АКСЕСУАРІВ З ПРИНТАМИ 8
1.1 Загальна характеристика віртуальних асистентів для підтримки клієнтів інтернет-магазину 8
1.2 Опис методів покращення функціоналу інформаційної системи підтримки клієнтів інтернет-магазинів 11
1.3 Аналіз компонентів сучасного віртуального асистента 14
2 ПОСТАНОВКА ЗАДАЧІ 19
2.1 Мета та задачі дослідження 19
2.2 Вимоги до віртуального асистента 20
2.3 Інструменти реалізації 22
3 ПРОЕКТУВАННЯ 25
3.1 Структурно-функціональне моделювання 25
3.2 Моделювання варіантів використання 27
3.3 Проектування моделі бази даних 29
4 ПРОГРАМНА РЕАЛІЗАЦІЯ 32
4.1 Розробка модуля чат-бота 32
4.2 Інтеграція чат-бота у інтернет-магазин 37
4.3 Створення базового функціоналу 42
4.4 Налаштування роботи штучного інтелекту 47
ВИСНОВКИ 55
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 57
ДОДАТОК А. Планування робіт 62
ДОДАТОК Б. Лістинг програмного коду 72

ВСТУП

На сьогоднішній день більшість веб-сайтів залежать від меню, головним чином заснованого на навігації та панелі пошуку для надання даних користувачу. Проте веб-сайти з величезною кількістю вмісту, матеріалів і погано структурована навігація можуть ускладнити пошук товарів. Зазвичай портал онлайн-покупок має величезний каталог продуктів. Перегляд товарів може бути важким і займати багато часу, враховуючи різні аспекти продукту. Також інколи щоб отримати додаткову інформацію про продукт, люди запитують замість того, щоб вивчати його опис. Віртуальні асистенти для споживачів стають все більш відомими завдяки клієнтоорієнтованому бізнесу.

Чат-боти відносяться до механізмів спілкування, розроблених великими підприємствами для власного використання, щоб покращити якість обслуговування і зменшити загальний бюджет обслуговування клієнтів. За допомогою чат-бота компанії можуть без особливих зусиль надавати величезну допомогу та приймати рішення у будь-який час доби та для величезної кількості клієнтів одночасно [1]. Потреба в чат-ботах на основі штучного інтелекту, як величезному компоненті еволюції підприємства, буде тільки продовжувати збільшуватись.

Щодо інтернет-магазинів, здебільшого клієнти віддають перевагу чеканні в черзі, щоб отримати відповідь працівника, що є невигідним, оскільки займає багато часу та потребує людський ресурс.

Було проведено аналіз різних методології створення віртуальних асистентів для підтримки клієнтів інтернет-магазинів. Можна зробити висновок, що розробка віртуального асистента є доволі трудоємним процесом. Створення сучасного віртуального асистента пов’язане з різними проблемами, але складність може залежати від таких факторів, як складність функцій асистента, необхідний рівень налаштування та інтеграція з існуючими системами. Хоча розробка може бути складною, існують також готові рішення та платформи, які забезпечують основу для створення таких асистентів. Ці платформи часто постачаються з попередньо

навченими моделями та інструментами для налаштування, що робить процес розробки більш доступним. Однак налаштування та інтеграція з конкретними бізнес-вимогами все одно можуть вимагати досвіду та ретельного планування.

Однією з таких платформ може виступати один із сучасних месенджерів з можливістю підтримки чат-ботів. Таку платформу можна інтегрувати у свій віртуальний асистент для підтримки клієнтів інтернет-магазину. Також необхідно задіяти штучний інтелект. OpenAI надає API, який дозволяє розробникам отримувати доступ та інтегрувати можливості ChatGPT у свої програми, продукти чи послуги. Цей API дозволяє інтегрувати функції розуміння природної мови та генерувати властивості у віртуальний асистент.

Інтегрувавши існуючі платформи зі штучним інтелектом в інформаційну систему підтримки клієнтів можна значно покращити можливості генерування відповідей, а також забезпечити більш інтерактивний та зручний досвід для клієнтів, які взаємодіють з інтернет-магазином. Тому пропонується провести аналіз існуючих методів створення віртуального асистента з метою подальшої розробки віртуального асистента на базі чат-бота, що використовує найсучасніші системи штучного інтелекту.

Метою роботи є розроблення віртуального асистента для інформаційної система підтримки обслуговування клієнтів інтернет-магазину одягу та аксесуарів з принтами за рахунок створення інтеграції з чат-ботом та впровадження штучного інтелекту.

Задачами дослідження є аналіз предметної області, вибір технологій для розробки віртуального асистента та впровадження штучного інтелекту в віртуальний асистент; аналіз платформ чат-бота для вибору однієї з них в якості інтерфейсу; моделювання технології; розроблення віртуального асистента.

1 АНАЛІЗ МЕТОДОЛОГІЇ СТВОРЕННЯ ВІРТУАЛЬНОГО АСИСТЕНТА ДЛЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ ПІДТРИМКИ ОБСЛУГОВУВАННЯ КЛІЄНТІВ ІНТЕРНЕТ-МАГАЗИНУ ОДЯГУ ТА АКСЕСУАРІВ З ПРИНТАМИ

1.1 Загальна характеристика віртуальних асистентів для підтримки клієнтів інтернет-магазину

Віртуальні асистенти для підтримки клієнтів інтернет-магазинів, також відомі як віртуальні асистенти e-commerce, – це інструменти, призначені для допомоги користувачам у різних аспектах їхнього досвіду онлайн-покупок [1]. Ці віртуальні асистенти використовують різноманітні технології, щоб розуміти запити користувачів, надавати інформацію та покращувати загальну навігацію та підтримку клієнта. Наведемо деякі загальні характеристики та функціональні можливості віртуальних асистентів інтернет-магазину [2]:

- Підтримка клієнтів
    - Допомога в режимі реального часу. Віртуальні асистенти можуть надати миттєву допомогу користувачам, відповідаючи на запитання, пов’язані з продуктами, замовленнями, доставкою та поверненням.
    - Інтерфейси чату. Багато віртуальних асистентів використовують інтерфейси чату, щоб вести інтерактивні розмови з користувачами, імітуючи спілкування з представником служби підтримки клієнтів.
- Рекомендації щодо продукту
    - Персоналізація. Аналізуючи поведінку та вподобання користувачів, віртуальні асистенти можуть пропонувати персоналізовані рекомендації щодо продуктів, щоб покращити процес покупки.
    - Перехресні продажі та додаткові продажі. Віртуальні асистенти можуть пропонувати додаткові або більш цінні товари на основі інтересів користувача та історії покупок.
- Пошук і навігація

- Пошук природною мовою. Користувачі можуть шукати продукти за допомогою запитів природною мовою, а віртуальний асистент інтерпретує та повертає відповідні результати.
- Керована навігація. Віртуальні асистенти можуть допомогти користувачам переміщатися між категоріями продуктів, застосовувати фільтри та уточнювати результати пошуку.

- Відстеження замовлень
  - Оновлення статусу. Користувачі можуть запитувати про статус своїх замовлень, отримувати інформацію про відстеження та отримувати оновлення щодо доставки та доставки.
- Допомога з кошиком
  - Нагадування. Віртуальні асистенти можуть нагадувати користувачам про товари, що залишилися в їхніх кошиках для покупок, і заохочувати їх завершити покупку.
  - Знижки та заохочення. Вони можуть пропонувати знижки або заохочення, щоб мотивувати користувачів завершити свої операції.
- Управління наявності товару
  - Наявність продукту. Віртуальні асистенти можуть надавати інформацію про наявність продукту в режимі реального часу, сповіщати користувачів про поповнення запасів або інформувати їх про обмежений запас.
- Акції та знижки
  - Оголошення. Віртуальні асистенти можуть ділитися інформацією про поточні акції, знижки та спеціальні пропозиції.
  - Застосування знижок. Вони можуть допомогти користувачам застосовувати відповідні акції до їхніх замовлень.
- Відгуки та огляди
  - Збір відгуків. Віртуальні асистенти можуть запитувати та збирати відгуки користувачів про продукти та загальний досвід покупки.
  - Підсумки відгуків. Вони можуть надавати підсумки оглядів продуктів, щоб допомогти користувачам приймати обґрунтовані рішення.

- Інтеграція з іншими системами
    - Інтеграція з платформою e-commerce. Віртуальні асистенти легко інтегруються з базовою платформою e-commerce, системами CRM та іншими базами даних для доступу та отримання відповідної інформації.
- Безперервне навчання
    - Віртуальні асистенти можуть використовувати методи машинного навчання, щоб постійно навчатися на основі взаємодії користувачів і з часом покращувати свою продуктивність [3].

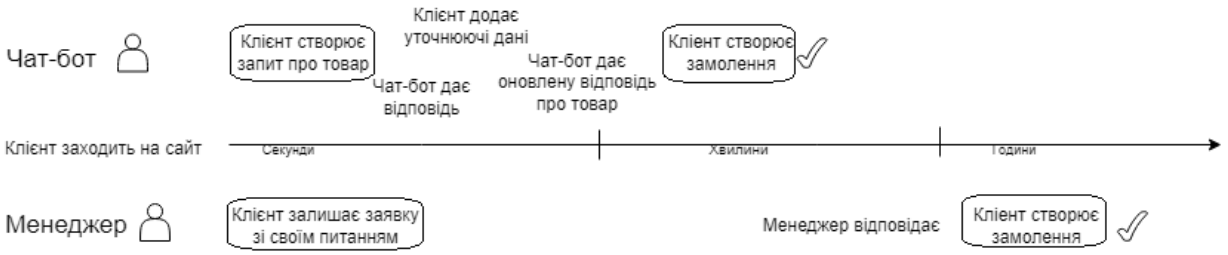

Рисунок 1.1 – Переваги використання віртуальних асистентів.

Джерело: розроблено автором

Мета впровадження віртуального асистента полягає в тому, щоб надати користувачам більш персоналізований, інтерактивний та ефективний досвід покупок, імітуючи допомогу, яку можна отримати у звичайному магазині [3]. На рисунку 1.1 зображено переваги в економії часу при користуванні чат-ботом як віртуальним асистентом при запиті інформації про товар та створені замовлення.

1.2 Опис методів покращення функціоналу інформаційної системи підтримки клієнтів інтернет-магазинів

Удосконалення функціональності інформаційної системи підтримки клієнтів передбачає поєднання оптимізації наявних функцій, додавання нових можливостей і покращення загального досвіду користувача [4]. Приведемо кілька стратегій, які можна розглянути, щоб покращити функціональність інтернет-магазину.

Треба переконатися, що інтернет-магазин повністю оптимізований для мобільних пристроїв. Більшість користувачів переглядають і роблять покупки за допомогою смартфонів, тому дизайн, зручний для мобільних пристроїв, має вирішальне значення для бездоганної роботи [5]. Також не мало важливо оптимізувати процес навігації за допомогою чітких категорій, фільтрів і функцій пошуку, спростити користувачам пошук продуктів.

Крім вище згаданих методів, не менш важливими є й інші методи:

- Швидкий час завантаження. Користувачі очікують швидких і чутливих веб-сайтів, а швидший час завантаження може значно підвищити задоволеність користувачів і знизити показники відмов.
- Персоналізація. Бажане застосування персоналізовані рекомендації на основі вподобань користувача, історії покупок і поведінки веб-переглядача. Персоналізація може покращити враження від покупок і збільшити ймовірність переходів на товари.
- Розширені функції пошуку - покращення функції пошуку за допомогою таких функцій, як автозаповнення, перевірка орфографії та обробка природної мови. Надійна пошукова система допомагає користувачам швидко знаходити продукти.
- Високоякісні зображення продукту - використання зображення високої роздільної здатності для своїх продуктів. Чіткі та привабливі візуальні елементи створюють позитивне враження та допомагають користувачам приймати зважені рішення про покупку.

- Відгуки та рейтинги клієнтів - показ відгуків клієнтів і оцінок продуктів. Позитивні відгуки зміцнюють довіру, а надання платформи для відгуків клієнтів може вплинути на потенційних покупців.
- Кілька варіантів оплати. Бажано пропонувати різноманітні варіанти оплати, щоб задовольнити різні вподобання клієнтів, додавати такі популярні методи, як кредитні картки, цифрові гаманці та інші безпечні платіжні системи [6].
- Безпечний процес оформлення замовлення - створення безпечного і простого процесу оплати, зведення до мінімуму кількість кроків, необхідних для завершення покупки, і надання чітких інструкцій, щоб зменшити рівень покидання кошика клієнтом[7].
- Створення віртуального асистента. Інтеграція з віртуальними асистентами в онлайн-магазинах може значно покращити взаємодію з клієнтами, надаючи персоналізовану допомогу, спрощуючи взаємодію та пропонуючи цінну інформацію. Надаючи інтелектуальну допомогу з урахуванням контексту, віртуальні асистенти сприяють більш зручному, ефективному та персоналізованому досвіду онлайн-покупок, зрештою підвищуючи задоволеність і лояльність клієнтів [2].

Штучний інтелект (ШІ) у чат-ботах змінює спосіб взаємодії онлайн-магазинів зі своїми клієнтами, роблячи процес більш ефективним, персоналізованим і зручним для користувача. Ця технологія продовжує розвиватися, пропонуючи нові можливості для покращення взаємодії з клієнтами в електронній комерції [8].

В джерелі [9] детально проаналізовано вибір між чат-ботом на основі правил і чат-ботом на основі ШІ, що залежить від конкретних потреб, контексту та ресурсів бізнесу чи системи. Кожен вид має свої переваги і недоліки. Для початку розглянемо чат-боти на основі правил. Їм притаманні такі якості:

1. Обмежений обсяг: вони можуть обробляти лише запити, які були передбачені та запрограмовані, що призводить до обмежень у обробці складних або несподіваних питань.
2. Передбачуваність: вони дотримуються заздалегідь визначених правил і сценаріїв, забезпечуючи передбачувані та послідовні відповіді.

3. Надійність у конкретних сценаріях: добре працює для конкретних завдань, коли запити клієнтів є простими та передбачуваними.
4. Простота: легше розробляти та впроваджувати, що робить їх придатними для підприємств з обмеженими технічними ресурсами.
5. Економічна ефективність: загалом впровадження та обслуговування є більш рентабельним порівняно з чат-ботами на основі ШІ.

Щодо чат-ботів на основі ШІ, їм були визначені такі якості:

1. Гнучкість: можуть обробляти широкий спектр запитів, у тому числі складних і непередбачуваних, розуміючи контекст і наміри.
2. Обробка природної мови (NLP): здатність обробляти природну мову та реагувати на неї, що забезпечує більш схожу на людську взаємодію.
3. Навчання та адаптація: може вчитися на взаємодії та вдосконалюватися з часом, надаючи все ефективніші відповіді.
4. Персоналізація: можливість пропонувати персоналізовані відповіді та рекомендації, аналізуючи дані та поведінку користувачів.
5. Ресурсовитратні: вимагають більше ресурсів для розробки, навчання та обслуговування.
6. Складність: складніша у створенні та може вимагати спеціальних навичок у сфері ШІ та машинного навчання.

Для вибору необхідного типу чат-бота необхідно врахувати призначення та складність завдань, які він буде виконувати. Для простих і зрозумілих завдань достатньо чат-ботів на основі правил. Для складних, різноманітних взаємодій більше підходять чат-боти на основі ШІ. Також необхідно враховувати масштабованість системи. Чат-боти на основі ШІ краще масштабуються зі збільшенням складності та обсягу взаємодій.

На основі правил ідеально підходить для невеликих або спеціальних програм, де запити передбачувані, а ресурси обмежені. ШІ є кращим для динамічних та великомасштабних програм, які вимагають складної взаємодії, персоналізації та здатності до навчання та адаптації. Але також на практиці часто використовується

гібридний підхід, коли система, заснована на правилах, обробляє стандартні запити, а штучний інтелект виконує складні взаємодії [9].

1.3 Аналіз компонентів сучасного віртуального асистента

Віртуальний асистент зазвичай складається з кількох ключових компонентів, кожен з яких сприяє його загальній функціональності та можливостям. Ці компоненти працюють разом, щоб розуміти введені дані користувача, виконувати завдання та надавати відповідну інформацію. Ось основні компоненти віртуального асистенту: управління діалогом, виконання завдання, інтерфейс користувача, моделі машинного навчання, база знань, персоналізація, механізм зворотного зв'язку, багатоканальна підтримка [10].

Управління діалогом стосується процесу контролю потоку розмови між віртуальним асистентом і користувачем. Це включає в себе керування зворотним обміном інформацією, розуміння намірів користувача та забезпечення узгодженої взаємодії з урахуванням контексту [2, 3].

Виконання завдання в контексті віртуального асистента відноситься до процесу виконання певних дій або операцій у відповідь на запити або запити користувача. Коли користувач взаємодіє з віртуальним асистентом, він часто має на увазі завдання або цілі, а віртуальний асистент призначений для виконання або сприяння виконанню цих завдань.

Необхідно проаналізувати наявні месенджери з платформами чат-ботів та обрати одну з них для подальшого проектування віртуального асистента.

Месенджер – це платформа, розроблена для відправлення миттєвих повідомлень (текстових та медіа) і часто містить додаткові функції, такі як голосові та відео дзвінки, чат-боти, можливості онлайн-покупок і ігри, при цьому використовується широкою аудиторією.

У рамках цієї роботи передбачено провести аналіз та вибрати один із сучасних месенджерів, який підтримує платформу чат-ботів. Для дослідження вибрані такі месенджери, як Viber, Telegram та Facebook.

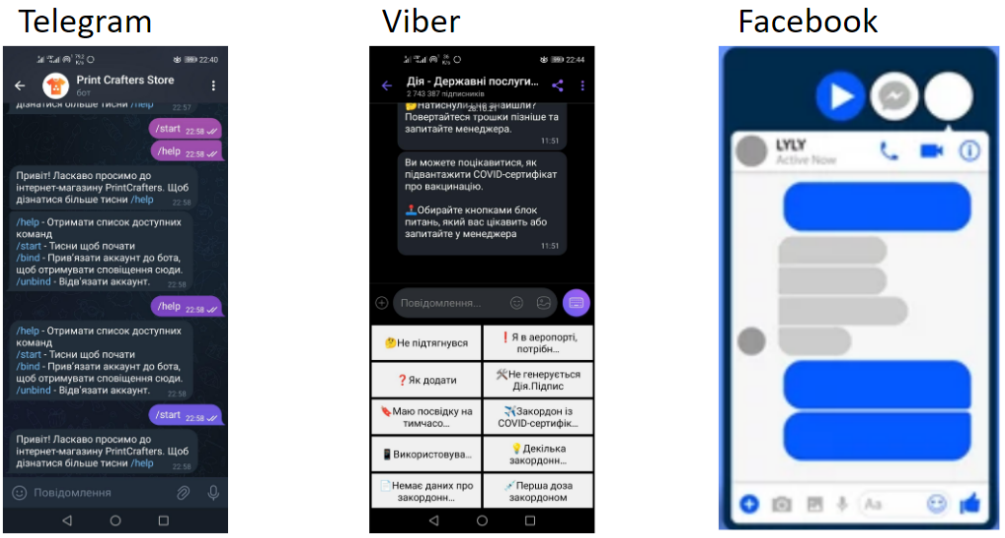

Рисунок 1.2 – Популярні месенджери.

Джерело: розроблено автором

Критерії аналізу включають: протоколи передачі даних, форма використання, інтерфейси, доступи, формати передачі даних, механізми зв’язку, основні функціональні можливості (такі, як опитування, листування, передача файлів, інтегрований магазин покупок, ігри).

Інтерфейс користувача (UI) відноситься до засобів, за допомогою яких користувачі взаємодіють із асистентом і отримують від нього інформацію. Це візуальне або звукове представлення можливостей і відповідей віртуального асистента. Метою добре розробленого інтерфейсу користувача є сприяння плавній та інтуїтивно зрозумілій взаємодії між користувачами та віртуальним асистентом. Дизайн графічного інтерфейсу повинен бути створений, використовуючи принципи коректного впровадження UI (User Interface) та UX (User Experience). Термін "Користувацький досвід" відноситься до взаємодії людини з продуктом, програмою чи операційною системою. Таким чином, розробка користувацького досвіду, або,

іншими словами, створення UX, означає визначення способу функціонування продукту та його відповідності потребам користувача. Очевидно, що UX повинен бути чітким, зручним для користувача, правильно розроблений, детальний і продуманий [11].

Таблиця 1.1 містить порівняльну інформацію про популярні месенджери, що підтримують платформу чат-ботів.

Таблиця 1.1 Порівняння платформ чат-ботів.

Характеристика	Viber	Facebook	Telegram
Доступи	Підписка; вбудований в діалог	Підписка; вбудований в діалог	Додавання до групи; підписка; вбудований в діалог
Форма використання	Умовно безкоштовно	Умовно безкоштовно	Безкоштовно
Функціональні можливості	Листування; магазин покупок; передача файлів; ігри	Листування; опитування; магазин покупок; передача файлів; ігри	Листування; опитування; магазин покупок; передача файлів; ігри
Формати передачі даних	JSON, multipart/formdata	JSON, multipart/formdata	JSON, URL query, multipart/formdata
Механізми зв’язку	webhook	webhook	webhook, polling
Інтерфейси користувача	Текстовий, кнопковий, голосовий	Текстовий, кнопковий, голосовий	Текстовий, кнопковий, голосовий
Протоколи передачі даних	https	https	https

Джерело: розроблено автором

База знань — це сховище структурованої інформації, до якої асистент може отримати доступ, щоб надавати точні та відповідні відповіді на запити користувачів. Це сховище зазвичай містить дані про продукти, послуги, політики, поширені запитання (FAQ) та іншу відповідну інформацію, про яку можуть запитувати користувачі. База знань служить основним ресурсом для віртуального асистента, який може використовуватися під час взаємодії з користувачами [12].

Персоналізація у віртуальних асистентах стосується здатності адаптувати роботу користувача на основі індивідуальних уподобань, поведінки та історичних взаємодій. Це передбачає налаштування вмісту, рекомендацій і відповідей, які надає віртуальний асистент, щоб підвищити задоволеність і залученість користувачів. Персоналізація є ключовим аспектом створення більш індивідуальної та відповідної взаємодії між користувачами та віртуальними асистентами.

Механізм зворотного зв’язку — це система, яка дозволяє користувачам надавати вхідні дані, коментарі чи оцінки щодо їх взаємодії з віртуальним асистентом. Цей цикл зворотного зв’язку має вирішальне значення для покращення продуктивності, зручності використання та ефективності віртуального асистента з часом. Механізм зворотного зв’язку збирає інформацію від користувачів у формі коментарів, оцінок або конкретних відповідей. Користувачам може бути запропоновано поділитися своїми думками щодо продуктивності віртуального асистента або надати відгук про певні взаємодії [12, 7]. Віртуальні асистенти часто включають системи оцінювання, які дозволяють користувачам призначати бали або давати відгуки про якість отриманої допомоги. Це може варіюватися від простих оцінок «подобається» або «не подобається» до більш детальних шкал оцінювання. Зазвичай користувачам надається можливість надати конкретні коментарі, пропозиції чи додаткові відомості про свій досвід. Цей якісний відгук може запропонувати цінну інформацію про сфери, які потрібно вдосконалити. Аналіз настроїв можна застосувати до відгуків користувачів, щоб оцінити загальні настрої користувачів. Розуміння настроїв користувачів допомагає визначити сфери задоволення та сфери, які потребують уваги.

Ці компоненти разом дозволяють віртуальному асистенту розуміти потреби користувачів, виконувати завдання та забезпечувати персоналізований та ефективний досвід роботи з різними програмами та галузями, включаючи онлайн-магазини.

2 ПОСТАНОВКА ЗАДАЧІ

2.1 Мета та задачі дослідження

Покращення функціональності онлайн-магазинів має вирішальне значення з кількох причин, оскільки воно безпосередньо впливає на взаємодію з користувачем, задоволеність клієнтів і загальний успіх e-commerce. Постійно вдосконалюючи функціональні можливості онлайн-магазинів, підприємства можуть створювати позитивний і безперебійний досвід покупок, підвищуючи лояльність клієнтів, заохочуючи повторні операції та залишаючись конкурентоспроможними в динамічному середовищі e-commerce. Одним з методів покращення функціоналу є створення віртуального асистента на основі штучного інтелекту. Розробка віртуального асистента для онлайн-магазинів пропонує кілька переваг, які сприяють покращенню взаємодії з клієнтами, підвищенню ефективності та покращенню бізнес-результатів [3, 7].

- покращене залучення клієнтів;
- цілодобова доступність;
- покращена підтримка клієнтів;
- ефективна обробка замовлень;
- економія часу та коштів.

Таким чином, розробка віртуального асистента для системи підтримки клієнтів інтернет-магазинів важлива для збереження конкурентоспроможності, підвищення задоволеності клієнтів і оптимізації бізнес-процесів. Він змінює досвід онлайн-покупок, надаючи персоналізовану допомогу, покращуючи ефективність і сприяючи загальному успіху бізнесу.

Мета даної роботи – розробка віртуального асистента для інформаційної система підтримки обслуговування клієнтів інтернет-магазину одягу та принтів для збереження конкурентоспроможності, підвищення задоволеності клієнтів і оптимізації бізнес-процесів. За результатами аналізу характеристик віртуальних

асистентів, методів покращення функціоналу та аналізу компонентів сучасного віртуального асистента для даної роботи поставлені наступні задачі:

1. Визначення вимог та інструментів реалізації.
2. Вибір технологій для розробки віртуального асистента.
3. Аналіз та вибір платформи чат-бота як інтерфейсу віртуального асистента.
4. Розробка та реалізація віртуального асистента для інформаційної системи підтримки клієнтів інтернет-магазину одягу та аксесуарів з принтами.
5. Впровадження штучного інтелекту у віртуальний асистент.

2.2 Вимоги до віртуального асистента

Визначаючи вимоги до віртуального асистента інформаційної системи підтримки клієнтів, важливо враховувати як функціональні, так і нефункціональні аспекти. Функціональні вимоги описують, що повинна робити система, тоді як нефункціональні вимоги визначають, наскільки добре система повинна виконувати певні функції [13]. Наведемо основні функціональні і нефункціональні вимоги до віртуального асистента в інформаційній системі підтримки клієнтів інтернет-магазина.

Функціональні вимоги:

- Розуміння природної мови (NLU). Віртуальний асистент повинен бути здатний розуміти запити користувача, виражені природною мовою, включаючи здатність розпізнавати наміри, сутності та контекст.
- Отримання інформації про продукт. Асистент повинен отримувати детальну інформацію про продукти, включаючи описи, специфікації, ціни, наявність і відгуки клієнтів.
- Персоналізовані рекомендації. Асистент повинен надавати персоналізовані рекомендації щодо продукту на основі вподобань користувача, історії покупок і поведінки веб-переглядача.

- Підтримка клієнтів і поширені запитання. Віртуальний асистент має пропонувати підтримку клієнтів, відповідаючи на поширені запитання (FAQ), вирішуючи типові проблеми та надаючи допомогу в режимі реального часу.

Нефункціональні вимоги:

- Продуктивність. Віртуальний асистент повинен оперативно реагувати на запити користувача, забезпечуючи низьку затримку та високу швидкість реагування.
- Масштабованість. Система повинна бути масштабованою, щоб обслуговувати все більшу кількість користувачів і взаємодій без пропорційного зниження продуктивності.
- Надійність і доступність. Асистент має бути надійним, зводити до мінімуму простої та забезпечувати високу доступність для користувачів, особливо під час пікових періодів покупок.
- Зручність використання та доступність. Інтерфейс користувача має бути інтуїтивно зрозумілим і доступним для різноманітної бази користувачів, забезпечуючи позитивний досвід користувача для всіх.
- Інтеграція зі сторонніми службами. Система повинна підтримувати інтеграцію зі сторонніми службами, такими як платіжні шлюзи, постачальники послуг доставки та зовнішні API, щоб покращити функціональність.

Встановлюючи як функціональні, так і нефункціональні вимоги, розробники та зацікавлені сторони можуть переконатися, що віртуальний асистент онлайн-магазину відповідає очікуванням користувачів, працює надійно та відповідає бізнес-цілям онлайн-магазину.

2.3 Інструменти реалізації

Для реалізації інформаційної системи обрано монолітну архітектуру, це спрощує розробку, тестування та розгортання порівняно з розподіленою системою і забезпечує ефективність, оскільки відсутній зайвий наклад, пов'язаний із мережевою взаємодією між компонентами. Монолітна архітектура представляє собою структурний підхід до створення програмного забезпечення, при якому весь додаток або система формується як єдиний, нероздільний блок. У цьому архітектурному підході всі компоненти додатку, такі як користувацький інтерфейс, база даних, логіка додатку і т.д., знаходяться в одному кодовому базисі та запускаються в одному процесі. Монолітні застосунки традиційно розглядаються як рішення "все-в-одному".

Основні риси монолітної архітектури:

- Єдиний блок: Усі компоненти та функції додатку об'єднані в єдиному коді.
- Простота розробки та розгортання: Розробка, тестування та розгортання здійснюються для всього додатку як для єдиного цілого.
- Простота моніторингу та налагодження: Усі компоненти працюють в одному процесі, що полегшує моніторинг та відлагодження.
- Простота масштабування: В деяких випадках масштабування може бути простішим, оскільки всі компоненти розташовані в одному місці.
- Легкість управління: Управління додатком може бути простішим, оскільки весь код об'єднаний в одному проекті.

Монолітна архітектура має свої переваги, але також обмеження, особливо при створенні великих та складних систем. У таких випадках розглядається архітектурна перебудова, наприклад, в напрямку мікросервісної архітектури.

Система управління базою даних MySQL обрана через її високу продуктивність, швидкість операцій та легку масштабованість. Вибір MySQL також обумовлений його відповідністю стандартам мови SQL, що полегшує використання для користувачів, які вже мають досвід роботи з базами даних.

Реляційна база даних MySQL дозволяє зберігати дані різних типів, що дає можливість зберегти фрагменти документів в структурі реляційної бази даних і відзначається низкою ключових переваг, що сприяють її популярності у використанні в різних застосунках:

1. Організована структура: MySQL пропонує зберігання даних у впорядкованому, структурованому форматі. Це означає, що дані розміщуються в таблицях з рядками та стовпцями, забезпечуючи легкий доступ та організацію інформації.
2. Забезпечення цілісності даних: Реляційні бази даних підкреслюють важливість точності та надійності даних через встановлення строгих правил і обмежень, що забезпечують послідовність і достовірність даних.
3. Розширені можливості для запитів та аналізу: SQL у MySQL надає можливість виконувати складні запити, оптимізуючи обробку та аналіз великих масивів даних.

Завдяки цим перевагам, MySQL є відмінним варіантом для широкого спектру проектів – від особистих ініціатив до великих корпоративних систем. [14].

В якості штучного інтелекту буде використано ChatGPT від OpenAI. Його особливою характеристикою є здатність вести діалоги з користувачами, використовуючи природну мову, та імітувати людське спілкування для надання допомоги. Віртуальний асистент на базі ChatGPT буде мати ряд переваг. По-перше, після навчання ChatGPT на потрібних даних, він може ефективно покращити обслуговування клієнтів, надаючи швидкі та точні відповіді на їхні запитання, що сприяє зростанню рівня задоволення клієнтів. Віртуальний помічник також може забезпечити більш персоналізоване спілкування та показати індивідуальний підхід до кожного клієнта. По-друге, його щодобова робота дозволяє клієнтам відправляти запити у будь-який час, забезпечуючи їм зручність та можливість отримувати відповіді без очікування закінчення робочого дня служби підтримки, уникаючи при цьому затримок у відповідях [18].

ChatGPT інтегрує технології обробки природної мови (NLP) та машинного навчання (ML) для створення текстів, схожих на людське письмо. NLP вивчає взаємодію між комп'ютерами та людською мовою, а ML фокусується на розробці

систем, які можуть вчитися з даних. Поєднання цих технологій дозволяє ChatGPT точніше реагувати на запити, аналізуючи та ідентифікуючи закономірності у даних.

У процесі розробки віртуального асистента використані такі технології як VS Code, PHP 8.2, Laravel, Docker, ChatGPT API, та MySQL для реалізації серверної та клієнтської частини.

3 ПРОЕКТУВАННЯ

3.1 Структурно-функціональне моделювання

IDEF0 представляє собою методологію для функціонального моделювання та використовує графічну нотацію для структурованого опису бізнес-процесів.

Характерною рисою IDEF0 є її зосередженість на ієрархічній взаємодії об'єктів. У функціональній моделі IDEF0 основні компоненти представлені у вигляді блоків, які функціонують як "чорні скриньки" з певними входами, виходами, механізмами управління та виконавчими елементами. Ці блоки можуть бути детально розкладені до потрібного рівня деталізації. Найважливіша функція моделі розташовується в лівому верхньому куті. Блоки з'єднуються між собою за допомогою стрілок, які мають своє специфічне значення і описують зв'язки між функціональними елементами. Ця модель дозволяє описувати різні типи процесів, включаючи адміністративні та організаційні [15]. Види стрілок у цій моделі включають:

- Вхідні (подають завдання),
- Вихідні (показують результати діяльності),
- Контрольні (здійснюють управління, наприклад, через положення та інструкції),
- Механізми (використовуються для виконання завдань).

На рисунку 3.1 зображено контекстну діаграму в нотації IDEF0 для функціоналу віртуального асистента для інформаційної системи підтримки клієнтів інтернет-магазину одягу.

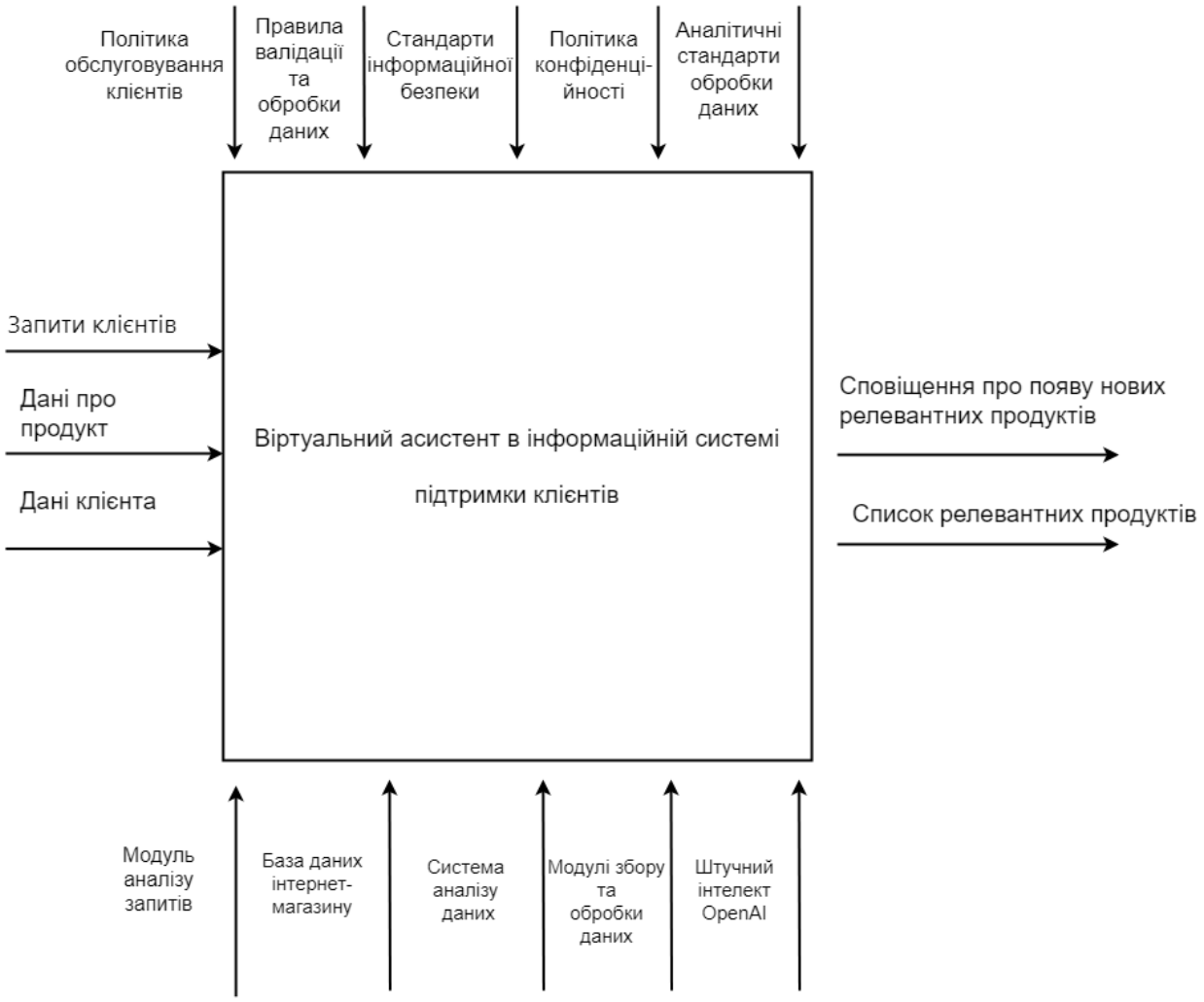

Рисунок 3.1 - Контекстна діаграма в нотації IDEF0 для функціоналу віртуального асистента в інформаційній системі підтримки клієнтів інтернет-магазину одягу.
Джерело: розроблено автором

Функціональна декомпозиція є технікою аналізу, яка розкладає складний процес на більш прості та зрозумілі компоненти. Вона широко застосовується у бізнесі для спрощення управління та розуміння об'ємних і складних процесів.

Цей метод є важливим інструментом для вирішення проблем і відіграє значну роль у розвитку різних областей, включаючи бізнес-операції, програмування, машинне навчання та інші сфери[16].

На рисунку 3.2 представлено декомпозицію IDEF0-діаграми для функціоналу віртуального асистента в інформаційній системі підтримки клієнтів інтернет-магазину одягу.

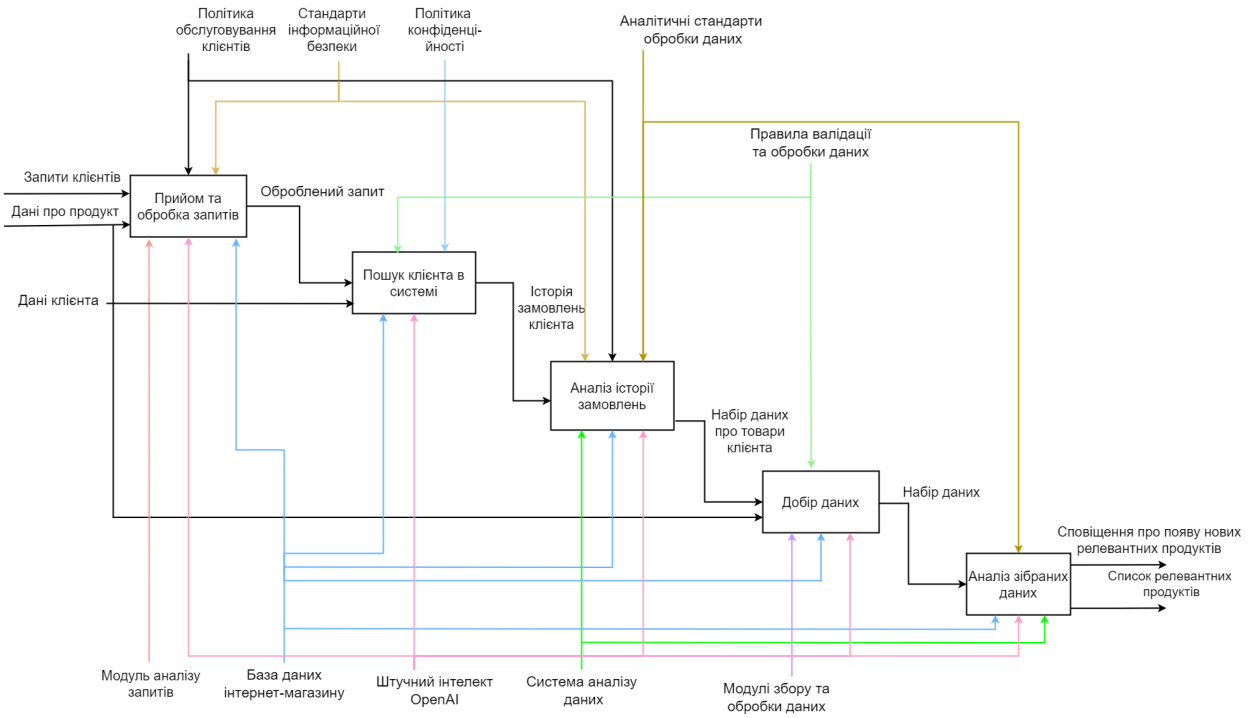

Рисунок 3.2 – Декомпозиція IDEF0-діаграми для віртуального асистента в інформаційній системі підтримки клієнтів інтернет-магазину одягу.
Джерело: розроблено автором

3.2 Моделювання варіантів використання

UML діаграма варіантів використання відображає дії, що відбуваються усередині системи, є прикладом динамічної діаграми. Вона відрізняється від статичних діаграм, оскільки демонструє, як поведінка користувачів або системних компонентів впливає на потенційні взаємодії чи зміни в процесах [17].

Основна користь цієї діаграми полягає в її спроможності допомагати розробникам і компаніям планувати процеси, виходячи з перспективи користувача. Це сприяє створенню більш ефективних систем, які краще відповідають потребам користувачів.

Діаграма варіантів використання віртуального асистента інтернет-магазину одягу представлена на рисунку 3.3.

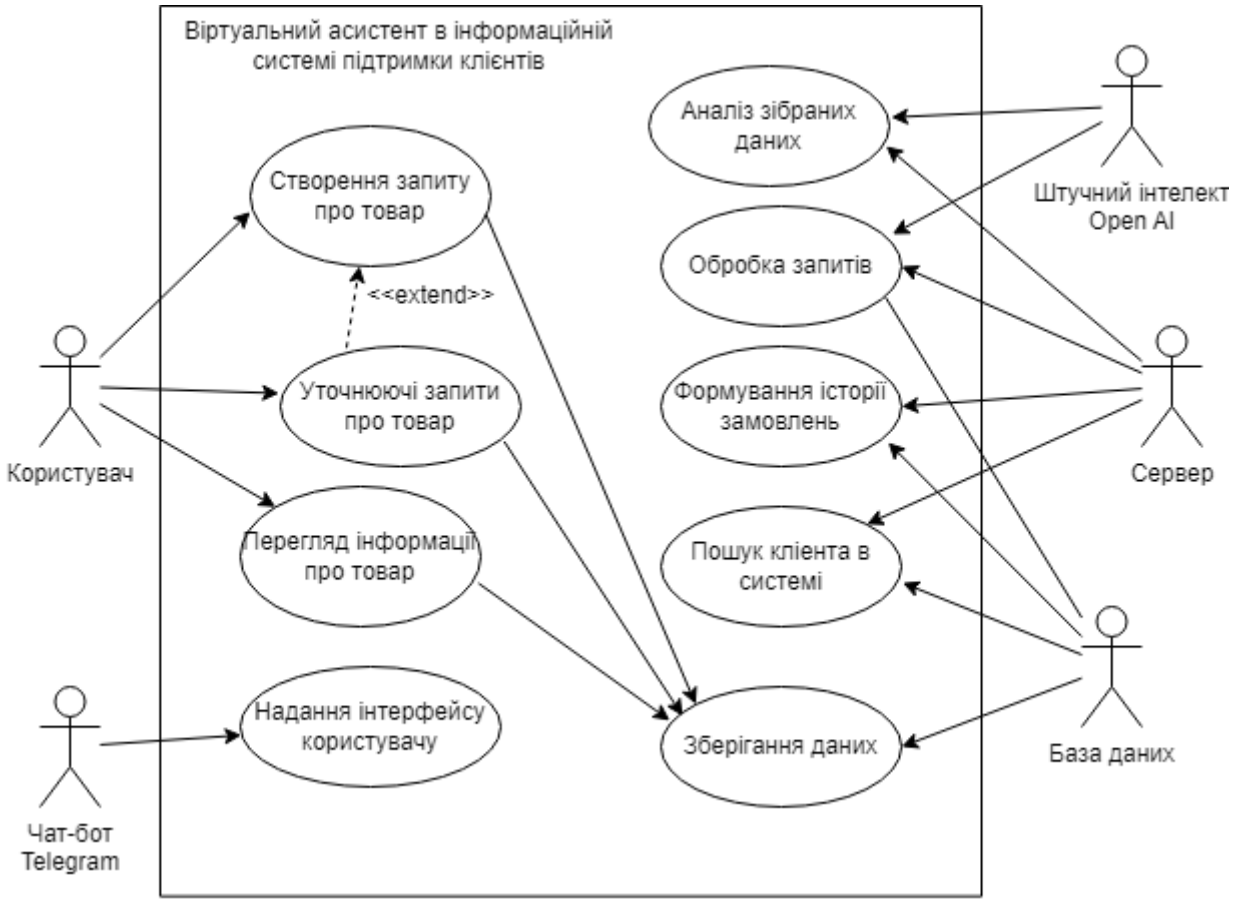

Рисунок 3.3 – Діаграма варіантів використання віртуального асистента інформаційної системи підтримки клієнтів інтернет-магазину одягу.

Джерело: розроблено автором

3.3 Проектування моделі бази даних

Розробка бази даних є процесом вибудовування структурованого плану, який вказує на те, як будуть організовані та доступні дані. Цей процес включає створення схеми бази даних, яка визначає її структуру і взаємозв'язки між різними даними, а також вибір найбільш відповідних типів даних та способів їх зберігання.

На рисунку 3.4 представлений приклад збереження товару користувача у json форматі.


```
{
"productID": "004596",
 "name": "Футболка червона",
 "description": "Test description",
 "price": 320,
 "quantity": 25,
 "category": "Одяг",
 "SKU": "Унікальний ідентифікатор товару",
 "imageURL": "",
 "dateAdded": "2023-01-01T12:00:00"
 }
```


Рисунок 3.4 – Інформація про товар користувача у форматі json.

Джерело: розроблено автором

Даний документ являє собою набір пар ключ-значення. В документі productID це ключ, а 004596 – значення.

На рисунках 3.5 – 3.7 представлено фрагменти фізичної моделі бази даних.

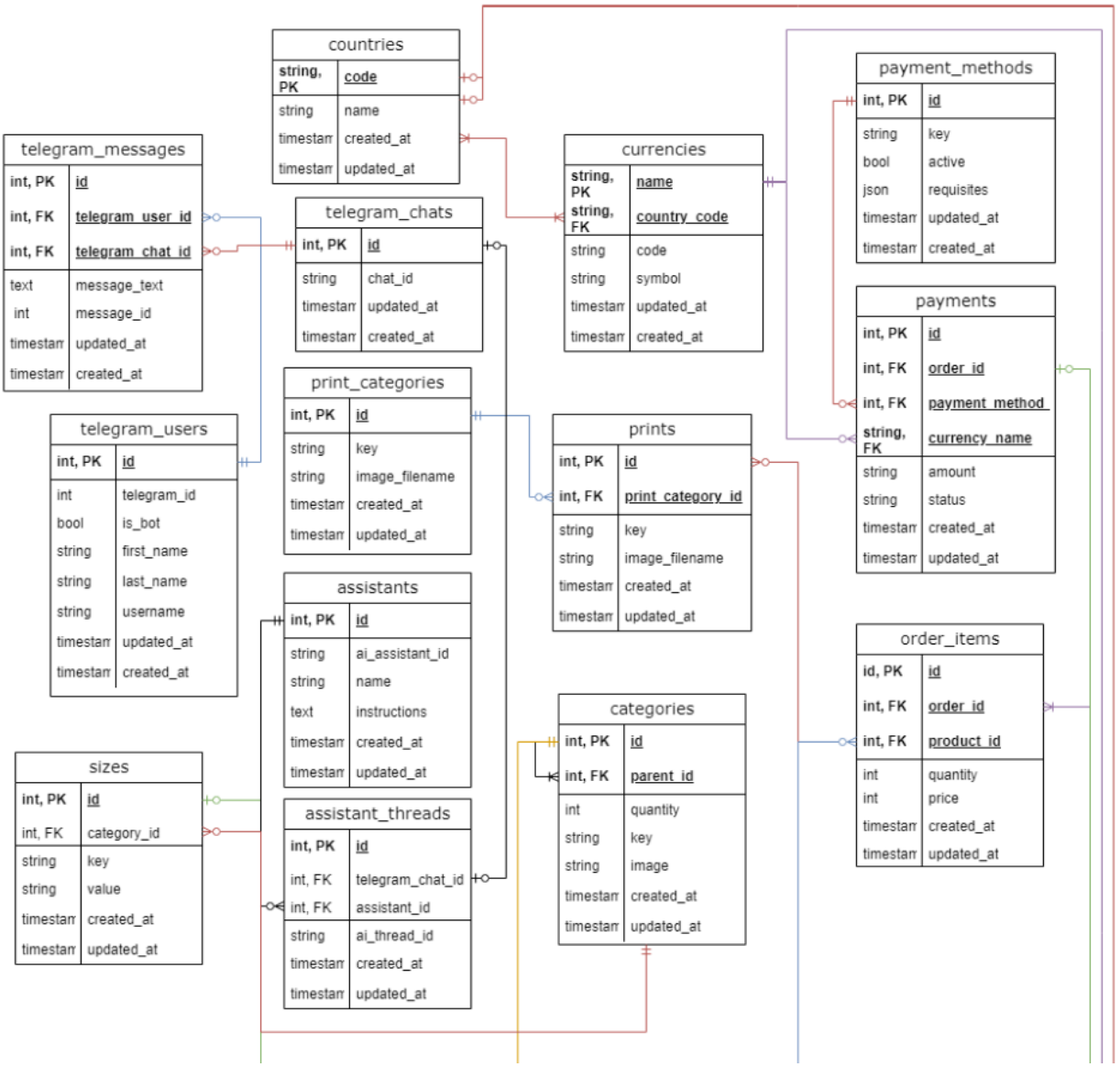

Рисунок 3.5 – Фрагмент ER-діаграми бази даних. Частина 1

Джерело: розроблено автором

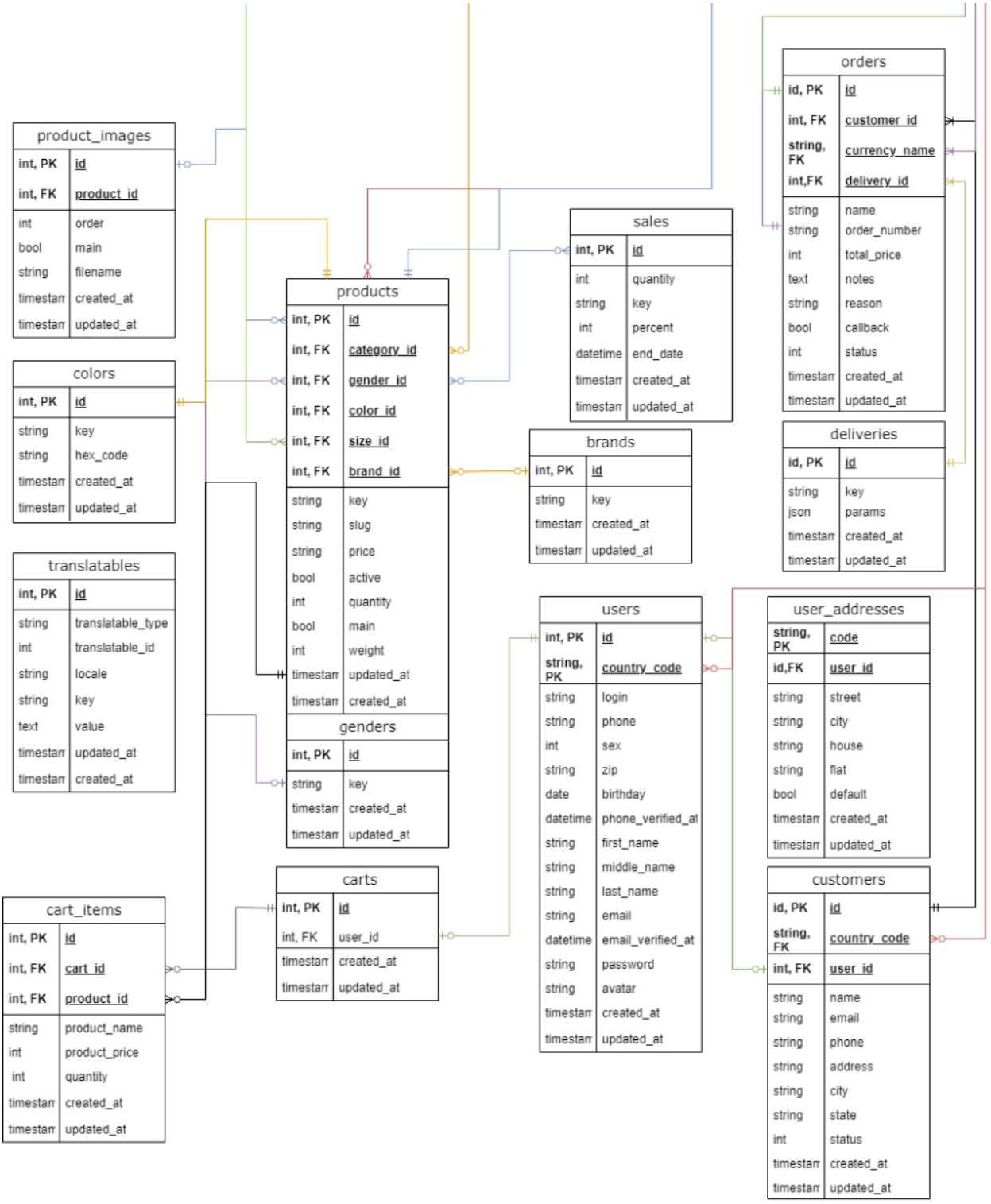

Рисунок 3.6 – Фрагмент ER-діаграми бази даних. Частина 2

Джерело: розроблено автором

4 ПРОГРАМНА РЕАЛІЗАЦІЯ

4.1 Розробка модуля чат-бота

Спершу необхідно встановити Telegram Bot SDK для Laravel. Для цього використовуємо команду: composer require irazasyed/telegram-bot-sdk

Ця команда дозволяє встановити пакет irazasyed/telegram-bot-sdk, який є популярним вибором для інтеграції ботів Telegram із Laravel. Цей пакет є дуже популярним по декільком причинам:

- Простота у використанні: Цей пакет пропонує легкий у освоєнні інтерфейс для взаємодії з API Telegram, спрощуючи процес створення і керування Telegram ботами, особливо для тих, хто вже обізнаний з Laravel.
- Розширений набір можливостей: Пакет включає великий асортимент функцій Telegram API, таких як відправлення повідомлень, зображень, клавіатур, обробку команд та багато іншого, що дає розробникам можливість створювати функціонально насичених ботів.
- Постійне оновлення: Постійні оновлення пакету забезпечують його сумісність з найновішими версіями Laravel та Telegram API, гарантуючи актуальність та безпеку використаних рішень.

Наступним кроком треба налаштувати конфігурацію Laravel проекта. Для цього ми додаємо токен бота Telegram до файлу .env.

Додавання токену бота Telegram до файлу .env у проекті, особливо під час роботи з такими фреймворками, як Laravel, важливо з кількох причин:

- Безпека: токен бота є конфіденційною інформацією. Він діє як ключ для керування ботом Telegram. Зберігаючи його у файлі .env, ми захищаємо його від загальнодоступної кодової бази, особливо якщо код розміщено в системах контролю версій, таких як Git. Це запобігає несанкціонованому доступу до бота.

- Керування конфігурацією: файл .env використовується для налаштувань конфігурації, які різняться в різних середовищах (розробка, постановка, виробництво тощо). Зберігання маркера бота в .env дозволяє легко керувати різними налаштуваннями для різних середовищ і перемикатися між ними, не змінюючи код ядра.
- Простота доступу: якщо зберегти токен бота у файлі .env, до нього можна легко отримати доступ у всій програмі за допомогою методів доступу до змінних середовища, наданих фреймворком. Це централізує конфігурацію, роблячи код чистішим і зручнішим для обслуговування. Отже, зберігання конфігураційних даних, таких як токени API, облікові дані бази даних тощо, окремо від логіки додатків, є найкращою практикою розробки програмного забезпечення. Це покращує модульність і зручність обслуговування коду.
- Гнучкість для оновлень і змін: якщо потрібно оновити токен бота або перейти на іншого бота, можна зробити це легко, просто змінивши файл .env без необхідності занурюватися в фактичну кодову базу.

Таким чином, збереження токена бота Telegram у файлі .env має вирішальне значення для безпеки, ефективного керування конфігурацією та дотримання найкращих практик у розробці програмного забезпечення. Додавання токена Телеграм-бота показано на рис. 4.1.

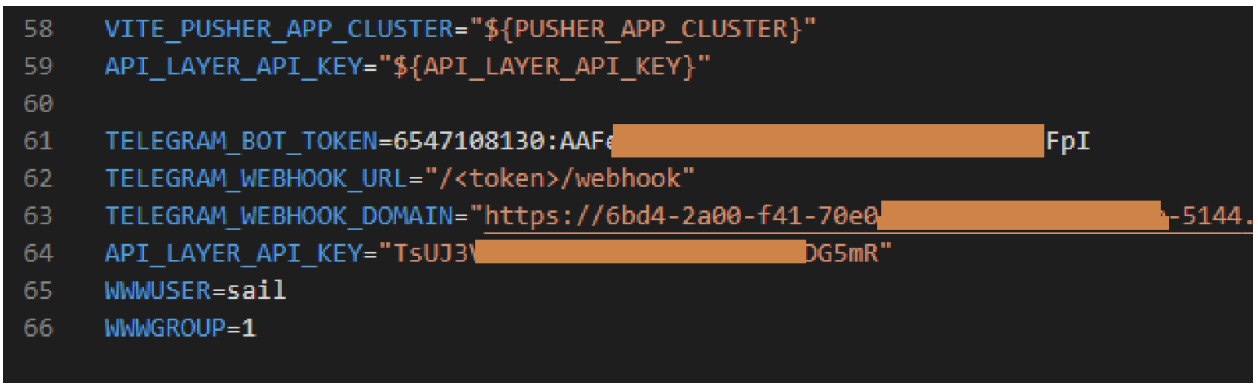

Рисунок 4.1 – Збереження токена бота у файлі .env.

Джерело: розроблено автором

Далі необхідно створити контролер Telegram Bot. Для цього скористаємося командою artisan, щоб створити новий контролер: php artisan make:controller TelegramBotController

В структурі Laravel, команда artisan відноситься до команди, яка виконується за допомогою інструмента командного рядка Artisan. Artisan — це назва інтерфейсу командного рядка, що входить до складу Laravel. Laravel поставляється з низкою вбудованих команд Artisan, які можуть виконувати широкий спектр завдань, включаючи міграцію баз даних, тестування та виконання завдань. Розробники також можуть створювати власні команди Artisan відповідно до конкретних потреб своєї програми. Це особливо корисно для автоматизації повторюваних завдань. Команди Artisan зазвичай мають структуру php artisan command:name, де command:name — це конкретна команда, яку треба виконати [19].

Для подальшої роботи над створенням бота необхідно буде використовувати такі команди, як:

- Очищення кешу (php artisan cache:clear)
- Запуск міграції бази даних (php artisan migrate)
- Створення шаблонного коду для нових контролерів, моделей тощо (php artisan make:controller, php artisan make:model)
- Запуск локального сервера розробки (php artisan service)

Також Artisan можна використовувати для планування завдань у програмі Laravel. Команда php artisan schedule:run використовується в завданні cron для виконання запланованих завдань. Artisan надає інтерфейс командного рядка для взаємодії з додатком Laravel, що дозволяє виконувати такі дії, як заповнення бази даних, публікація активів пакетів тощо. Тому ці команди є потужною функцією для виконання та керування різноманітними завданнями у проектах Laravel, пропонуючи як набір попередньо створених команд, так і гнучкість для визначення власних [20].

Наступним важливим кроком є налаштування Webhook. Для цього визначаємо маршрут у web.php, який вказує на метод контроллера.

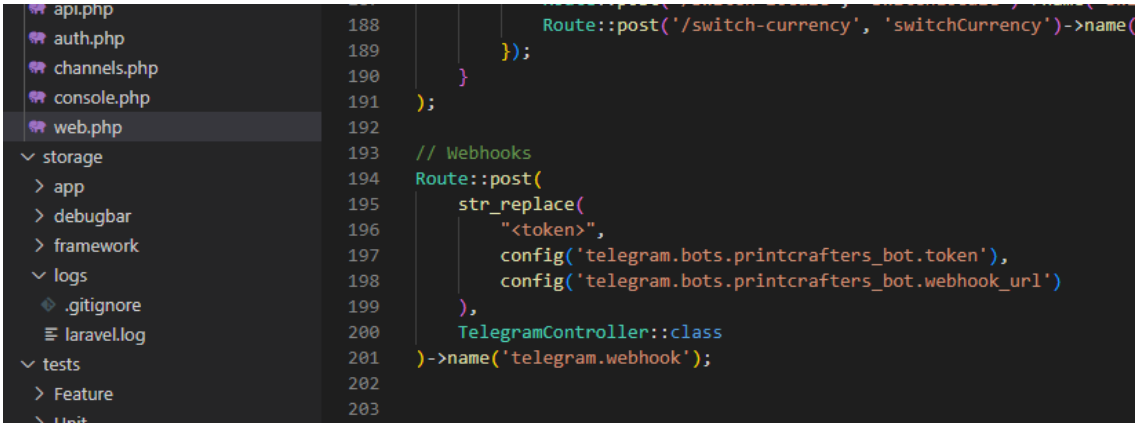

Рисунок 4.2 – Налаштування Webhook у файлі web.php.

Джерело: розроблено автором

```php
    /**
     * Setup Webhook.
     *
     * @throws TelegramSDKException
     */
    private function setupWebhook(): void
    {
        $this->info('Setting up webhook...');
        $this->newLine();

        $webhookUrl = data_get($this->config, 'webhook_url');

        if (! Str::startsWith($webhookUrl, 'https://')) {
            $this->error('Your webhook url must start with https://');

            return;
        }

        $params = ['url' => $webhookUrl];
        $certificatePath = data_get($this->config, 'certificate_path', false);

        if ($certificatePath && 'YOUR-CERTIFICATE-PATH' !== $certificatePath) {
            $params['certificate'] = $certificatePath;
        }

        $response = $this->telegram->setWebhook($params);
        if ($response) {
            $this->info('Success: Your webhook has been set!');

            return;
        }

        $this->error('Your webhook could not be set!');
    }

    /**
```

Рисунок 4.3 – Функція налаштування Webhook у файлі WebhookCommand.php.

Джерело: розроблено автором

Далі використовуємо Telegram API, щоб налаштувати URL-адресу веб-хуку на щойно створений маршрут. Це можна зробити за допомогою Telegram Bot SDK або шляхом прямого виклику API.

Налаштовуємо обробку вхідних повідомлень за допомогою створення логіки в контролері. Для цього у методі handle використовуємо Telegram Bot SDK для обробки вхідних оновлень і відповідної відповіді. Таким чином можна аналізувати команди, текстові повідомлення та інші типи оновлень.

```
46      public function handle(BotsManager $botsManager): void
47      {
48          $this->botsManager = $botsManager;
49          $bot = $this->argument('bot');
50
51          $this->resolveTelegramBot($bot);
52
53          $this->config = $this->botsManager->getBotConfig($bot);
54
55          if ($this->option('setup')) {
56              $this->setupWebhook();
57          }
58
59          if ($this->option('remove')) {
60              $this->removeWebhook();
61          }
62
63          if ($this->option('info')) {
64              $this->getInfo();
65          }
66      }
```

Рисунок 4.4 – Метод handle для налаштування команд Webhook-ів у файлі WebhookCommand.php.

Джерело: розроблено автором

4.2 Інтеграція чат-бота у інтернет-магазин

Інтеграція Telegram чат-бота з онлайн-магазином передбачає кілька етапів, зокрема налаштування чат-бота, підключення його до серверної частини магазину та налаштування для обробки різних взаємодій з клієнтами.

Для налаштування серверної частини інтернет-магазину потрібно переконатися, що інтернет-магазин має API для керування запитами про продукти, замовленнями та даними клієнтів. Також чат-боту знадобиться доступ до бази даних магазину, щоб отримати інформацію про продукт, керувати замовленнями тощо.

Для того, щоб інтегрувати чат-бот, необхідно налагодити взаємодію з базою даних інтернет-магазину. Наприклад, щоб показати продукти або обробити замовлення, ви можете запитати базу даних на основі даних, введених користувачем із Telegram [21].

Інтеграція бота з Інтернет-магазином також передбачає під’єднання бота до API магазину, щоб отримувати та надсилати дані з подальшим використанням веб-хуків для оновлень у реальному часі, як-от підтвердження замовлень, оновлення доставки тощо. Для цього необхідні додаткові таблиці до бази даних інтернет-магазину.

Для зберігання інформації про користувачів, які взаємодіють з ботом необхідно створити таблицю користувача з такими полями:

- telegram_id (id телеграм користувача)
- is_bot (параметр наявності бота у користувача)
- first_name (Ім’я)
- last_name (Прізвище)
- username (нік-ім’я)
- language_code (мова коду)
- created_at (дата створення боту)
- updated_at (дата оновлення інформації)
- id (унікальний id)

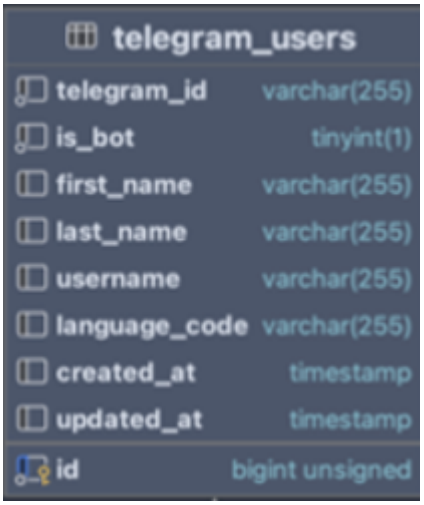

Рисунок 4.5 – Таблиця користувача Telegram бота.

Джерело: розроблено автором

Для зберігання інформації про повідомлення користувачів, які взаємодіють з ботом необхідно створити таблицю користувача з такимим полями:

- message_id (id повідомлення)
- update_id (id редагування повідомлення)
- telegram_chat_id (id телеграм-чату)
- telegram_user_id (id телергам-користувача)
- text (текст повідомлення)
- date (дата повідомлення)
- created_at (дата створення повідомлення)
- updated_at (дата редагування повідомлення)
- id (унікальний id)

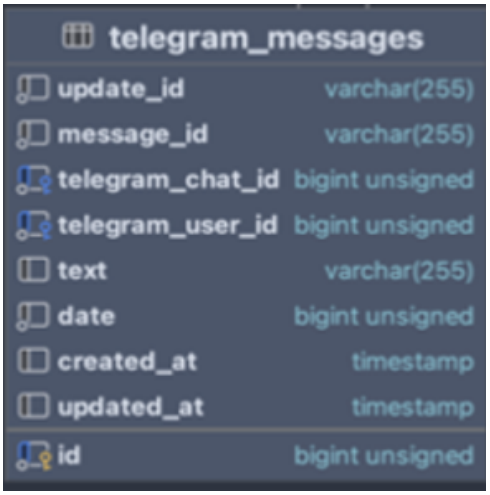

Рисунок 4.6 – Таблиця повідомлень користувача Telegram бота.

Джерело: розроблено автором

Для зберігання інформації про чати користувачів, які взаємодіють з ботом необхідно створити таблицю користувача з такимим полями:

- chat_id (id телеграм-чату)
- first_name (Ім’я)
- last_name (Прізвище)
- username (нік-ім’я)
- type (тип чату)
- title (заготовок чату)
- created_at (дата створення чату)
- updated_at (дата оновлення інформації)
- id (унікальний id)

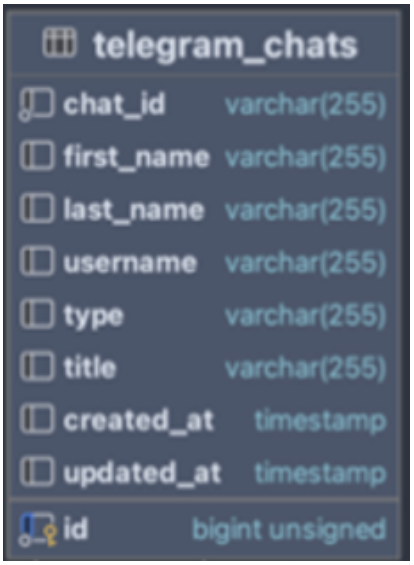

Рисунок 4.7 – Таблиця чатів користувача Telegram бота.

Джерело: розроблено автором

Для зберігання інформації про адміністраторів телеграм чатів, які взаємодіють з ботом необхідно створити таблицю з такими полями:

- admin_user_id (id адміністратора)
- telegram_chat_id (id телеграм-чату)
- created_at (дата створення чату адміністратора)
- updated_at (дата оновлення інформації чату адміністратора)
- id (унікальний id)

Рисунок 4.8 – Таблиця адміністраторів чатів Telegram бота.

Джерело: розроблено автором

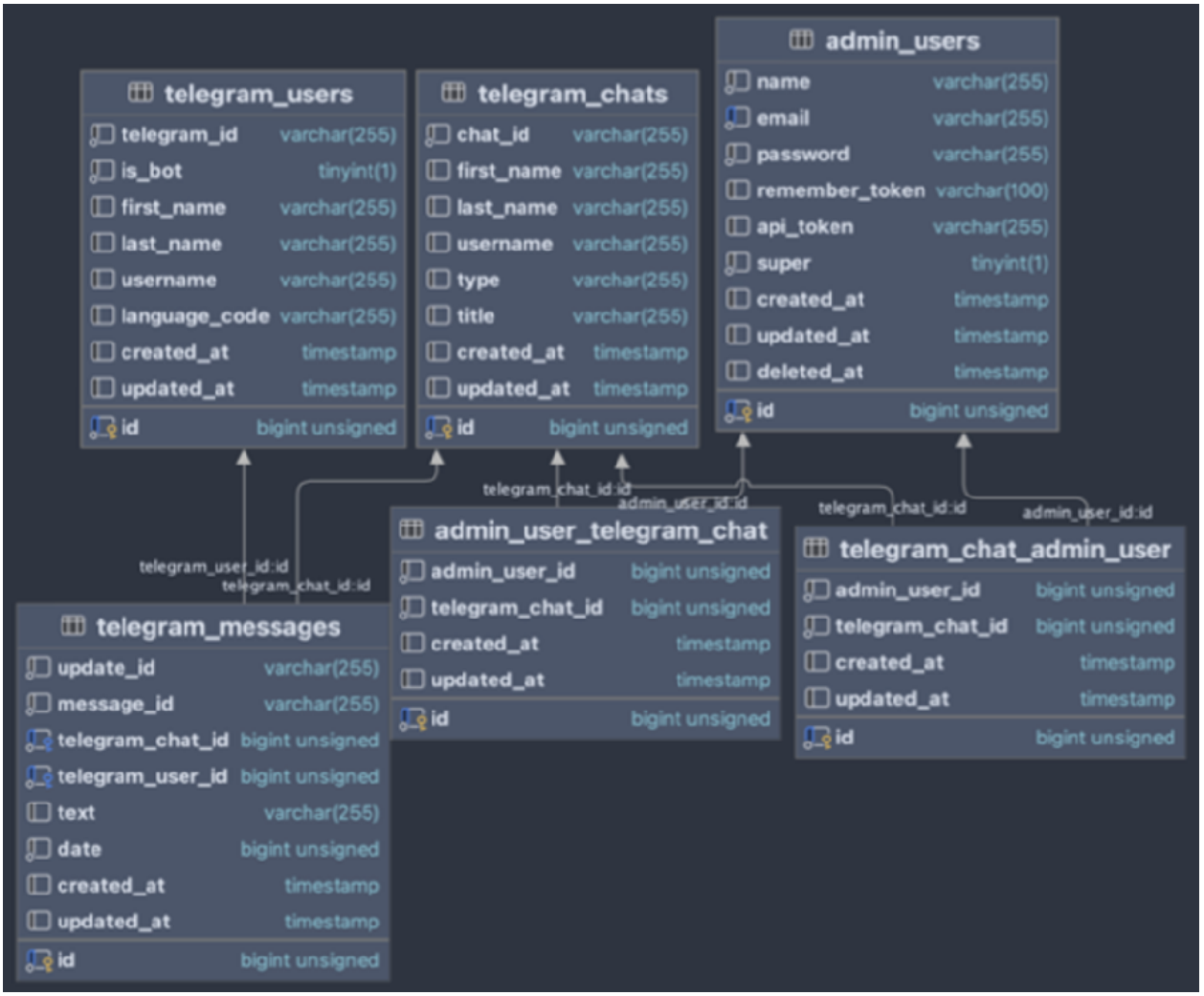

Рисунок 4.9 – Схема зв’язків реалізованих таблиць для Telegram бота.

Джерело:р автором

Після вибору користувачів із бази даних, система створює нові рядки в таблиці telegram_messages. Це забезпечує можливість відновлення процесу, якщо відбудеться перезапуск сервісу під час розсилки повідомлень. Далі, використовуючи ці новостворені записи з таблиці, розсилаються повідомлення обраним користувачам через сервіс взаємодії з Телеграм-бот API.

Під час інтеграції чат-бота Telegram із базою даних онлайн-магазину та створення додаткових таблиць слід враховувати кілька важливих міркувань, щоб переконатися, що система ефективна, безпечна та масштабована. Необхідно розробити схему бази даних для підтримки цілісності та узгодженості даних,

використовувати первинні ключі, зовнішні ключі та унікальні обмеження для забезпечення зв’язків і унікальності, де це необхідно. Також необхідно впроваджувати перевірки або обмеження, щоб забезпечити достовірність даних.

Також немало важливим є використання принципів нормалізації бази даних, щоб уникнути надлишкових даних, які можуть призвести до аномалій і невідповідностей, однак треба пам’ятати про надмірну нормалізацію, яка може призвести до складних запитів і вплинути на продуктивність.

Таблиці необхідно розробляти з урахуванням масштабованості, та передбачити майбутнє зростання обсягу даних і кількості користувачів. Також треба оптимізовувати структури таблиць для ефективності запитів, враховуючи індексацію важливих стовпців, до яких часто звертаються.

4.3 Створення базового функціоналу

На етапі додавання нового функціоналу до чат-бота важливо зрозуміти, як працює Telegram Bot API. API дозволяє боту виконувати різні дії, наприклад надсилати повідомлення, фотографії та обробляти різні типи введених користувачами даних. Взаємодіяти з цим API можна за допомогою Telegram Bot SDK [23].

Обробка вхідних оновлень: кожна взаємодія з чат-ботом є оновленням. Оновленням може бути повідомлення, команда, запит зворотного виклику з вбудованої клавіатури тощо. У методі обробки в TelegramBotController ми перевіряємо ці оновлення та вирішуємо, що з ними робити.

Нижче наведено шаблон коду для обробки вхідних оновлень:

```
  public function handle()
{
$telegram=new
Telegram\Bot\Api(config('services.telegram-bot-api.token'));
        $update = $telegram->commandsHandler(true);
        if ($update->isType('message')) {
            $this->handleMsg($update->getMsg());
```

```
        }
        // Обробка інших оновлень…
    }
    private function handleMsg($msg)
    {
        // Логіка обробки текстових повідомлень
    }
```

Відповідь на повідомлення є найпоширенішою функцією чат-боту. Наприклад, якщо необхідно відповісти на просте текстове повідомлення, то виконується такий код:

```
    private function handleMessage($msg)
    {
        $chatId = $msg->getChat()->getId();
        $text = $msg->getText();
        switch ($text) {
            case '/start':
                $response = 'Привіт!';
                break;
            case '/help':
                $response = 'Ось список команд...';
                break;
            // Додавання нових команд або відповідей за потреби
            default:
                $response = 'Вибачте, я не зрозумів вашого запиту.';
        }

        $telegram->sendMessage([
            'chat_id' => $chatId,
            'text' => $response
        ]);
    }
```

Якщо бот базується на командах (наприклад, /start, /help), то можна налаштувати ці команди в BotFather і обробляти їх у своїй програмі Laravel. Використовуйте умовні вирази або структуру змінного регістру, щоб визначити відповідь бота на кожну команду.

BotFather – це спеціальний бот у Telegram, який дозволяє користувачам створювати та керувати власними ботами Telegram. По суті, це бот, який створює інших ботів, наданий Telegram як офіційний спосіб створення та налаштування ботів на своїй платформі. Таким чином можна створити нового бота, надіславши команду /newbot до BotFather. Він проведе нас через процес, запитає ім’я та ім’я користувача

вашого бота, а потім згенерує унікальний маркер для бота. Цей маркер використовується для доступу до API Telegram Bot і керування ботом. BotFather надає різні команди для налаштування ботів. Наприклад, можна встановити описи ботів, зображення профілю та команди ботів. Це також дозволяє вмикати/вимикати певні функції, наприклад налаштування конфіденційності. Якщо якимось чином втрачено маркер API бота, то завжди можна отримати його знову, надіславши команду /token до BotFather. Якщо бот більше не потрібен, його можна видалити або вимкнути за допомогою BotFather [27].

Також можна отримати список усіх ботів, надіславши команду /mybots.

Щодо інтерактивних функції, для більшої інтерактивності можна використовувати клавіатури:

- Клавіатури відповідей: спеціальна клавіатура з попередньо визначеними відповідями.
- Вбудовані клавіатури: кнопки, що відображаються в повідомленні з даними зворотного виклику.

Обробка запитів зворотного виклику з вбудованої клавіатури вимагає додаткової логіки для обробки зворотних викликів і надсилання відповідних відповідей.

Окрім того, завжди необхідно включати обробку помилок у логіку свого бота для вирішення неочікуваних ситуацій. Функції логування Laravel можуть бути корисними для реєстрації помилок або важливої інформації.

У Laravel функції логування можуть бути вирішальними для моніторингу та налагодження чат-ботів Telegram. Laravel надає потужну систему логування, яка є гнучкою та налагоджувальною. Він заснований на бібліотеці Monolog, яка підтримує різноманітні обробники журналів. Можна використовувати можливості логування Laravel, щоб записувати важливу інформацію про операції, помилки та взаємодії свого бота Telegram [23, 28].

Надамо кілька ключових моментів щодо використання функцій логування Laravel у контексті чат-бота Telegram:

1. Налаштування каналів логування.

У Laravel проекті поведінка логування визначається у файлі config/logging.php. Laravel підтримує різні канали, такі як окремі файли, щоденні файли, Slack, Syslog тощо. Для бота Telegram можна вибрати щоденний файл, щоб журнали були впорядковані за датою, наприклад:

```
'daily' => [
    'driver' => 'daily',
    'path' => storage_path('logs/laravel.log'),
    'level' => 'debug',
    'days' => 7,
],
```

2. Запис повідомлень журналу.

Щоб записувати повідомлення журналу в код бота Telegram, можна використовувати фасад журналу Laravel. Також можна реєструвати різні рівні інформації (налагодження, інформація, повідомлення, попередження, помилка, критичний, попередження, надзвичайна ситуація). Наприклад:

```
use Illuminate\Support\Facades\Log;
// Інформаційні логи
Log::info('Новий користувач взаємодіє з ботом', ['user_id' =>
$userId]);
// Логи помилок
Log::error('Помилка обробки запиту.', ['exception' =>
$exception->getMessage()]);
```

3. Вхід у різні середовища.

Якщо необхідно змінити свою поведінку логування залежно від середовища (локального, проміжного, робочого), Laravel дозволяє легко налаштовувати параметри логування для різних середовищ за допомогою системи налаштування середовища.

4. Реєстрація специфічних подій Telegram-бота.

Можна реєструвати певні події в робочому процесі свого бота, наприклад:

- Отримано команди користувача та надіслано відповіді.
- Винятки або помилки під час обробки оновлень.
- Системні події, такі як налаштування веб-хуку або зміни конфігурації.
- Користувальницькі події програми, що стосуються логіки вашого бота.

5. Перегляд і моніторинг логів.

Журнали за замовчуванням зберігаються в каталозі storage/logs. Можна переглядати їх безпосередньо, використовувати такі команди, як tail у системах на

базі Unix, або налаштувати перегляд журналів у своїй програмі. Для більш розширеного моніторингу можна інтегрувати зовнішні служби, такі як Bugsnag або Sentry.

6. Видалення старих логів.

Щоб журнали не займали надто багато місця, систему журналів Laravel можна налаштувати так, щоб файли журналів зберігалися певну кількість днів.

Отже функції логування Laravel можуть значно допомогти підтримувати працездатність і продуктивність вашого чат-бота Telegram. Належне ведення журналу допомагає у вирішенні проблем, розумінні взаємодії користувача та відстеженні активності бота. Найкраще включити комплексне ведення журналів у розробку вашого бота, щоб забезпечити безперебійну роботу та полегшити обслуговування.

Рисунок 4.10 – Приклад роботи команд start та help.

Джерело: розроблено автором

На рисунку 4.10 продемонстровано роботу команд start та help. Командою start можна розпочати діалог. При виклику команди help, бот надає список доступних команд, а також опис до них.

4.4 Налаштування роботи штучного інтелекту

На даному кроці необхідно інтегрувати ChatGPT в систему віртуального асистента. Потім розробляється сценарій чату, що включає навчання чат-бота реагувати згідно з інформацією про наш інтернет-магазин. Цей процес включає збір текстових даних, які можуть бути отримані шляхом транскрипції розмов. Завершивши навчання, наступає етап розгортання моделі за допомогою SDK платформи. На заключному етапі важливо протестувати помічника, задаючи йому різні запитання та перевіряючи відповіді.

Для підключення штучного інтелекту до віртуального асистента спочатку потрібно створити файли конфігурації. Для цього спочатку потрібно створити файл - assistant.php, який буде повертати ID асистента:

```
<?php
return [
    'default' => env('DEFAULT_AI_ASSISTANT_ID')
];
```

Також створюємо клас сервісу асистента AssistantService.php (рис. 4.11), в якому будуть усі необхідні методи для відстеження повідомлень.

Додаємо AssistantRepository.php та AssistantThreadRepository.php. В них будуть міститися логіка для обробки основних функцій віртуального асистента. AssistantThreadRepository.php використовується для керування потоками розмов та сеансами з користувачами. Він містить обробку, зберігання, отримання та оновлення станів розмови, що важливо для підтримки контексту в поточній взаємодії з

віртуальним асистентом. Це гарантує, що чат-бот може надавати послідовні та відповідні контексту відповіді протягом розмови.

Обидва репозиторії допомагають організувати кодову базу, зберігаючи бізнес-логіку окремо від логіки додатків і роблячи код більш придатним для обслуговування та масштабованим.

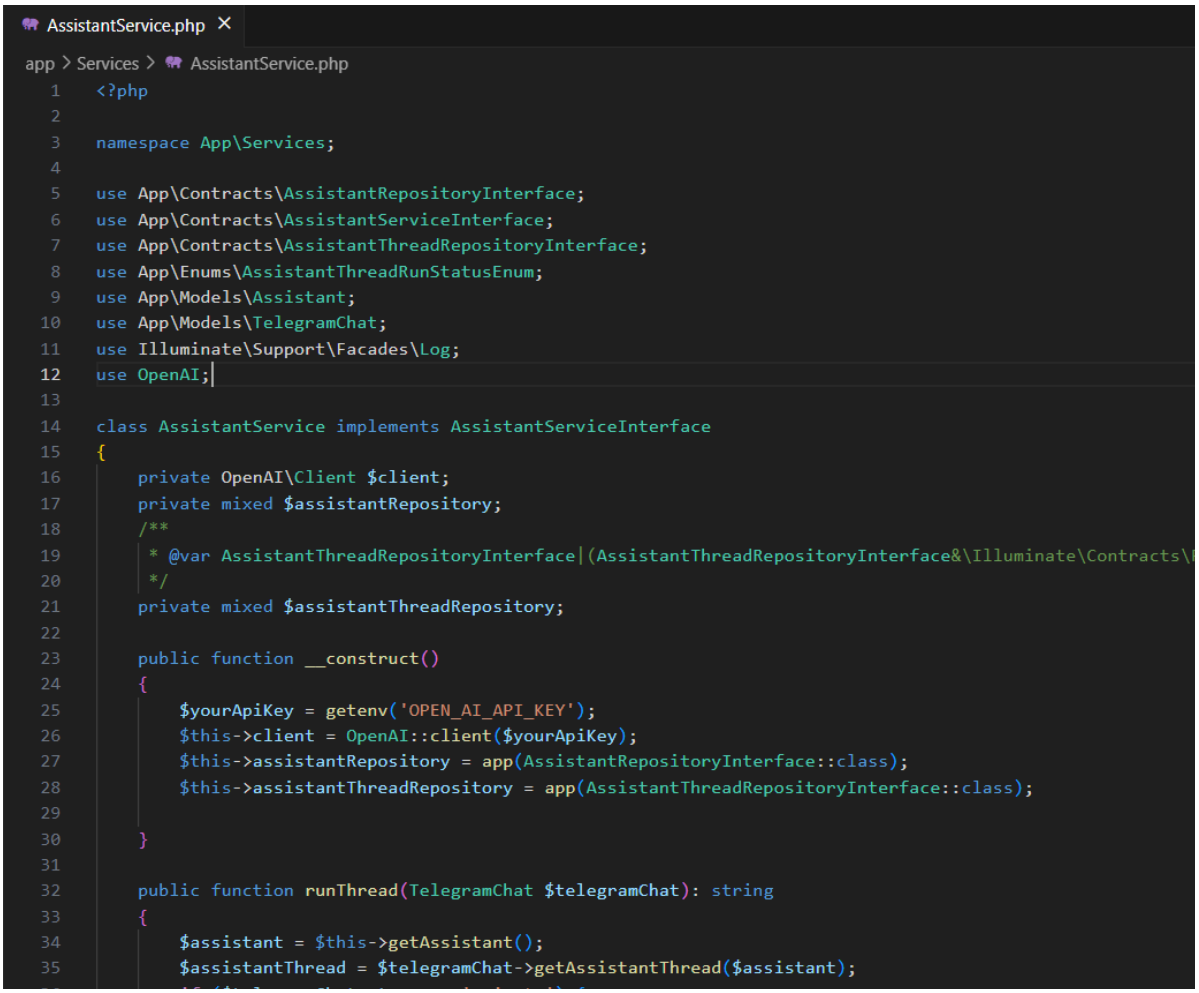

Рисунок 4.11 – Створення класу сервісу асистента AssistantService.php.

Джерело: розроблено автором

Створюємо файли моделі Assistant.php та AssistantThread.php.

Assistant.php та AssistantThread.php – це моделі, які описують сутність віртуального помічника (рис. 4.12).

Обидві моделі відіграють вирішальну роль у структуруванні та управлінні даними в додатку віртуального помічника, забезпечуючи надійну основу для зберігання та отримання важливої інформації.

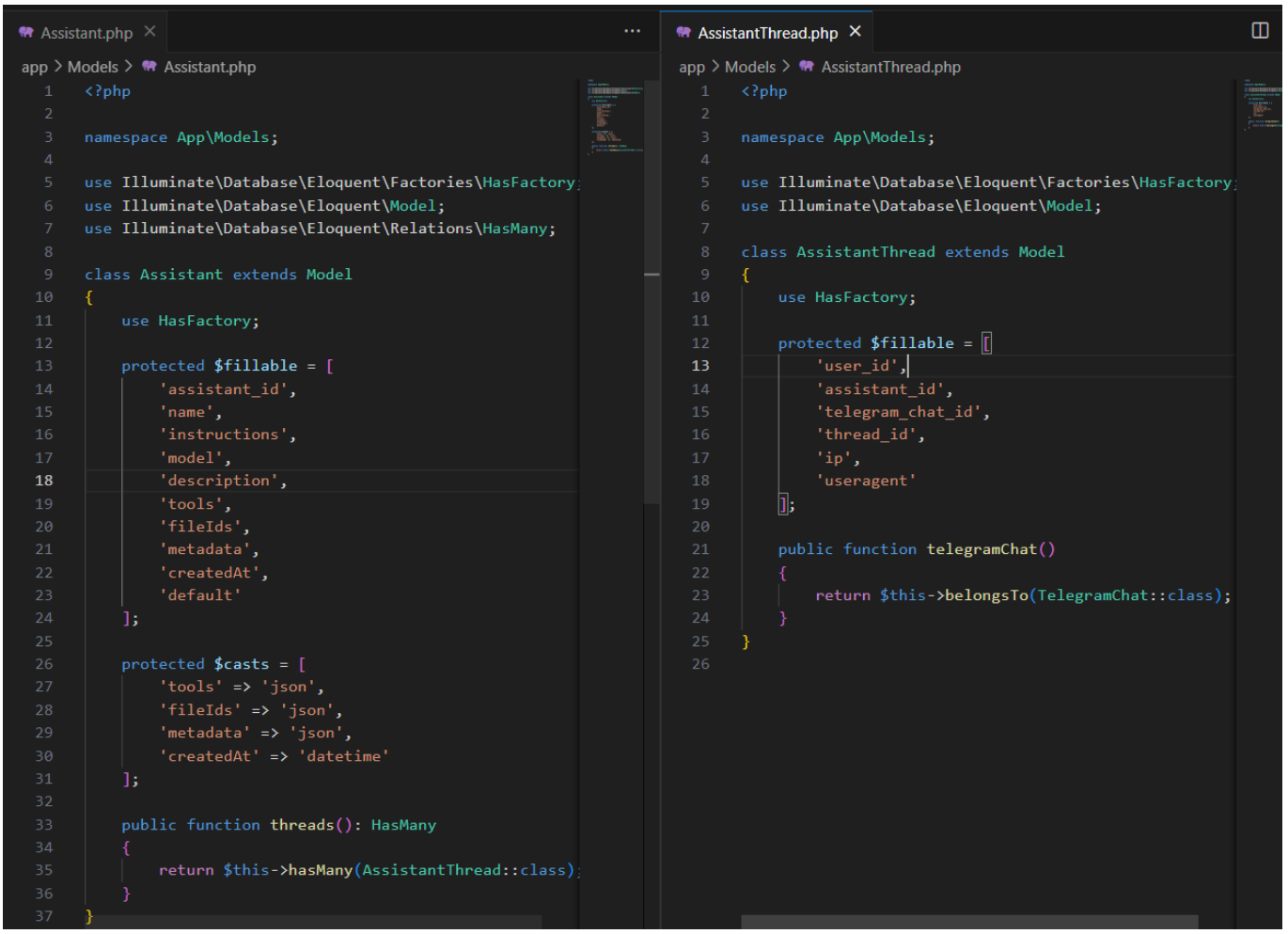

Рисунок 4.12 – Створення файлів моделі асистента Assistant.php та AssistantThread.php.

Джерело: розроблено автором

Файл HandleRequestToAssistant.php буде відповідати за обробку запитів, надісланих до віртуального асистента. Ці запити будуть ставати в чергу для асинхронного керування запитами користувачів, тим самим покращуючи продуктивність програми та взаємодію з користувачем. Робота може передбачати інтерпретацію введених користувачем даних, взаємодію з API ChatGPT для

отримання відповідей, а потім форматування та надсилання цих відповідей назад користувачеві. Використовуючи завдання в черзі, програма може обробляти складні або трудомісткі завдання у фоновому режимі.

Створення інтерфейсів для репозиторию та сервісу віртуального асистента:

- AssistantRepositoryInterface.php: цей файл визначає інтерфейс для допоміжного сховища. Він описує методи, які повинен мати будь-який клас, що реалізує цей інтерфейс. Ці методи можуть включати функції для керування та доступу до даних, пов’язаних з операціями віртуального асистента.
- AssistantServiceInterface.php: цей файл, визначає інтерфейс для служби помічника. У ньому будуть визначені основні функції бізнес-логіки, які повинен виконувати віртуальний помічник, абстрагуючись від того, як ці функції реалізовані.
- AssistantThreadRepositoryInterface.php: цей файл визначатиме інтерфейс для AssistantThreadRepository.php. Він описує методи керування потоками розмов або сеансами, гарантуючи, що будь-який клас реалізації надає функціональні можливості для зберігання та отримання контексту розмови та даних.

Використання інтерфейсів у такий спосіб сприяє чистій і модульній архітектурі, роблячи систему більш зручною для обслуговування, перевіркою та масштабованістю.

Для процесу навчання ChatGPT необхідні будуть дані із бази даних інтернет-магазину. Також вони будуть необхідні для того, щоб штучний інтелект міг продемонструвати, наскільки правильно він зрозумів надану інформацію. Для цього імпортуємо частину таблиць бази даних з формату sql у формат json. Далі надаємо ці json файли нашій моделі ChatGPT, пояснюючи, що саме міститься у цих файлах. В нашому випадку було сформовано наступні файли та передано для нашої моделі штучного інтелекту:

- products.json
- related_products.json
- sales.json
- sizes.json

- translatables.json
- brands.json
- categories.json
- colors.json
- payment_methods.json
- print_categories.json
- print_images.json
- product_sale.json

На рисунку 4.13 відображено інтерфейс налаштування моделі штучного інтелекту ChatGPT для віртуального асистента.

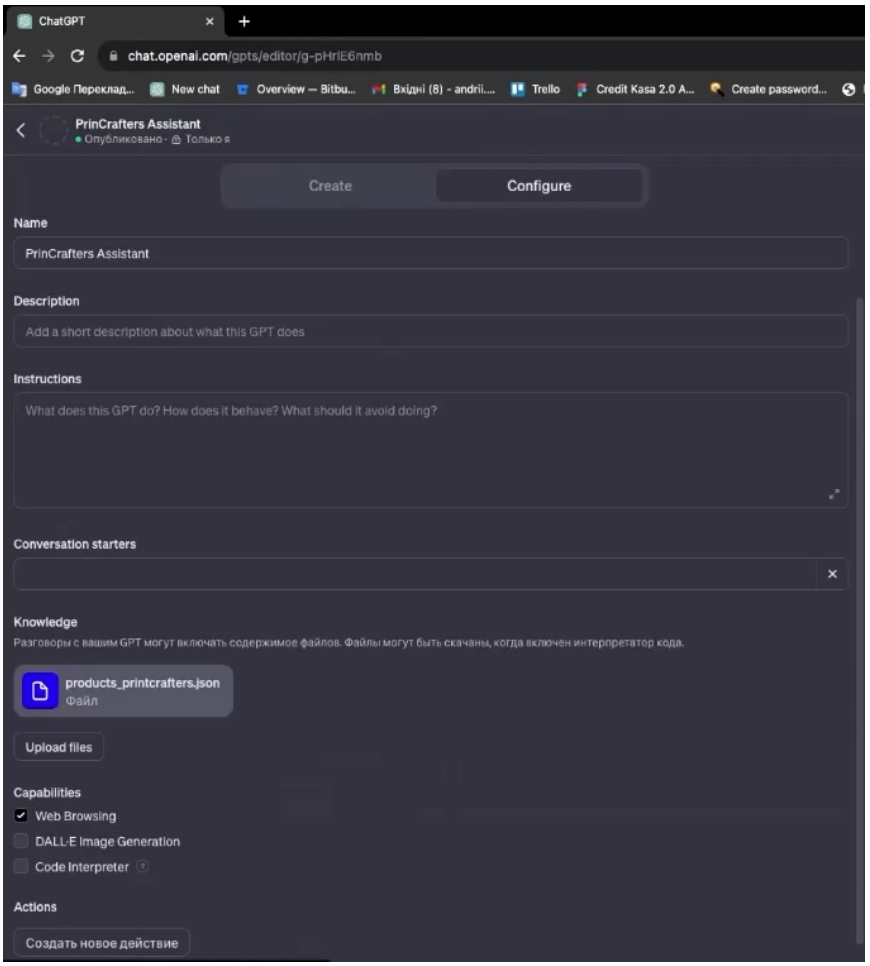

Рисунок 4.13 – Сторінка налаштування моделі штучного інтелекту ChatGPT для віртуального асистента.

Джерело: розроблено автором

Використана модель штучного інтелекту розробляється на основі великого обсягу текстів та вихідного коду. Процес навчання може зайняти кілька днів, в залежності від розміру даних і складності моделей. Після завершення навчання, модель можна адаптувати для конкретних цілей, забезпечуючи їй відповідну інформацію. Наприклад, для відповідей на питання за певною темою, модель потрібно навчити на відповідному наборі даних. Процес точного налаштування займає від декількох годин до декількох днів, особливо коли використовується більший обсяг даних. На рисунку 4.14 продемонстровано процес навчання моделі штучного інтелекту ChatGPT для віртуального асистента.

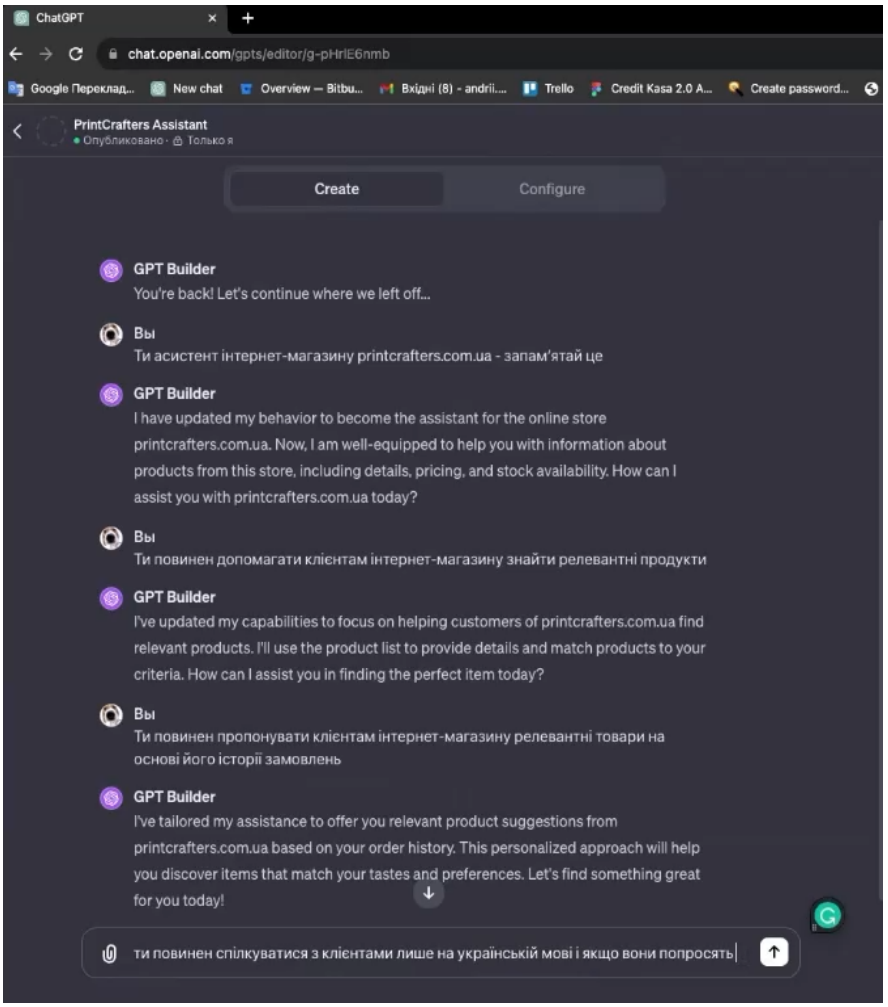

Рисунок 4.14 – Процес навчання моделі штучного інтелекту ChatGPT для віртуального асистента.

Джерело: розроблено автором

Після очищення та впорядкування даних їх можна використовувати для початку навчання моделі віртуального помічника. Час, необхідний для цієї процедури, залежить від кількох факторів, включаючи кількість даних, які потрібно обробити, і складність моделі.

Таблиця 4.1 Таблиця поведінки віртуального асистента.

Етапи	**Приклад запиту користувача**	**Поведінка асистента**
Вітання	“Добрий день”	Привітатися, коротко надати інформацію про себе. Запитати, чим я можу бути корисним.
Первинний запит інформації про наявний товар	“Чи є у вас футболки?”	Надати чітку та зрозумілу відповіть, поставивши після цього уточнююче запитання.
Уточнюючий запит про товар	“Мене цікавить жовта дитяча”	Надати відповідь про наявність товару за уточнюючою інформацією та поставити після цього уточнююче запитання.
Кінцевий вибір конкретного товару	“Мені подобається цей товар, буду купувати”	Надати короткий опис про товар та посилання на нього. Подякувати клієнту.
Формування замовлення	-	Після створення замовлення клієнтом, у чат має прийти повідомлення про створене замовлення
Обробка помилок: 1. Недостатньо інформації; 2. Затримка з відповідями.	-	1. Задати уточнюючі питання; 2. Якщо запит довго обробляється, вибачитись та попросити клієнта трохи почекати.

Джерело: розроблено автором

Після навчання та налаштування моделі штучного інтелекту виртуального асистента, необхідно провести її тестування. Для цього було використано стратегію тестування на основі моделі поведінки системи (Model-based testing) та техніку Exploratory тестування [13]. Модель поведінки відображена в таблиці 4.1 як базовий список перевірок, на основі якого формуються подальші тестові сценарії.

На рисунку 4.15 продемонстровано діалог користувача з віртуальним асистентом.

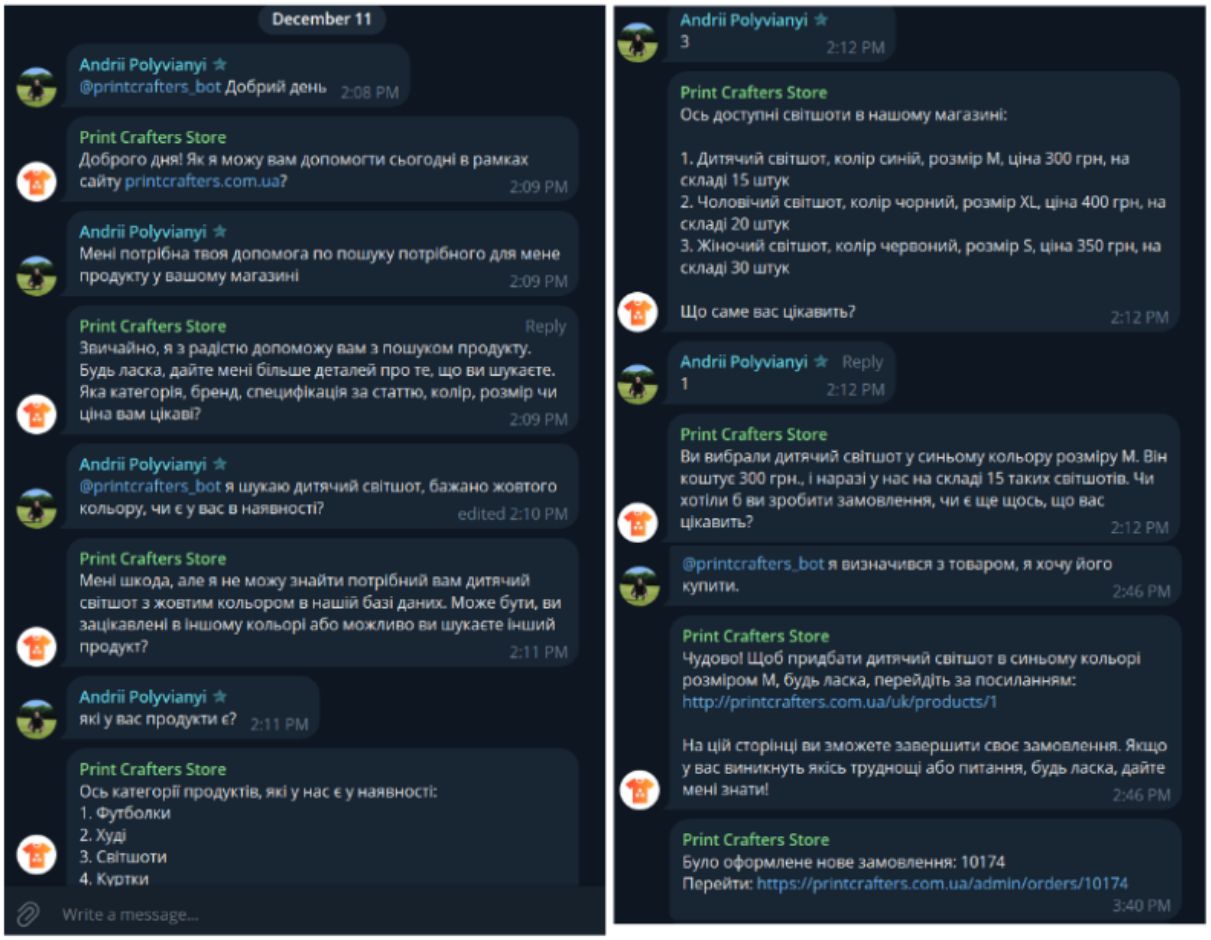

Рисунок 4.15 – Демонстрація діалогу з віртуальним асистентом.

Джерело: розроблено автором

ВИСНОВКИ

У контексті виконання кваліфікаційної роботи магістра була визначена мета – розроблення віртуального асистента для інформаційної системи підтримки обслуговування клієнтів.

Було проведено аналіз та вибір платформи чат-бота та обрано технології для розробки віртуального асистента. В якості платформи чат-бота було обрано месенджер Telegram, в якості інструментів для проектування системи управління чат-ботом бути обрані мова PHP та фреймворк Laravel, які дозволяють створити інтеграцію з месенджер-платформою Telegram.

Було розроблено віртуальний асистент для інформаційної системи підтримки клієнтів інтернет-магазину одягу та аксесуарів з принтами для реалізації такого функціоналу:

- здатність розуміти запити користувача, включаючи здатність розпізнавати наміри, сутності та контекст;
- надання персоналізованих рекомендації щодо продукту на основі уподобань користувача;
- підтримка клієнтів, відповідаючи на поширені запитання;
- масштабованість системи для можливості обслуговувати все більшу кількість користувачів і взаємодій без пропорційного зниження продуктивності;
- оперативні відповіді на запити користувачів із мінімальною затримкою та цілодобове обслуговування.

Було впроваджено штучний інтелект у віртуальний асистент для покращення можливостей генерування відповідей, а також забезпечення більш інтерактивного та зручного досвіду для клієнтів. Для цього було обрано інструмент ChatGPT від OpenAI, який є однією з найсучасніших систем, пропонуючи рішення на основі використання штучного інтелекту. Система має легкий та гнучкий в налаштуванні інтерфейс ChatGPT API.

Проведене навчання моделі штучного інтелекту для покращення взаємодії з клієнтом та створення поведінки максимально наближеної до людської з акцентом на професійне обслуговування клієнтів інтернет-магазину одягу та аксесуарів з принтами. Для цього були використані можливості налаштування за допомогою інструкцій та надання моделі штучного інтелекту необхідних даних про товари.

Для тестування роботи віртуального асистента було використано таблицю з тестовими запитами, яка дозволяє перевірити систему на кожному етапі взаємодії клієнта з віртуальним асистентом, починаючи з вітання і закінчуючи підтвердженням про сформоване замовлення.

Розроблена система дозволяє значно покращити взаємодію з клієнтами, надаючи персоналізовану допомогу, спрощуючи взаємодію та пропонуючи цінну інформацію.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. L. Cui, Sh. Huang, F. Wei, Ch. Tan, Ch. Duan, and M. Zhou “Super-Agent: A Customer Service Chatbot for E-commerce Websites”, Microsoft Research Asia {lecu, shaohanh, fuwei, v-chutan, v-chadu, mingzhou} @microsoft.com, 2017
2. Stefanus Ardhito Prasetya, Alva Erwin, Maulahikmah Galinium, “Implementing a Chatbot for E-Commerce Site Using Artificial Intelligence Markup Language (AIML)” - Faculty of Engineering and Information Technology, Swiss German University, 2018.
3. Anushree Deshmukh, Smit Shah, Heena Puthran, N. Shah, “Virtual Shopping Assistant for Online Fashion Store” - International Journal for Research in Applied Science & Engineering Technology (IJRASET), 31 May 2022, Computer Science.
4. Srividya, V., Tripathy, B.K., Akhtar, N., Katiyar, A. (2021). “AgentG: An Engaging Bot to Chat for E-Commerce Lovers.” In: Tripathy, A., Sarkar, M., Sahoo, J., Li, KC., Chinara, S. (eds) Advances in Distributed Computing and Machine Learning. Lecture Notes in Networks and Systems, vol 127. Springer, Singapore. https://doi.org/10.1007/978-981-15-4218-3_27
5. Sarthak V. Doshi, “Artificial Intelligence Chabot in Android System using Open Source”, Program-International Journal of Advanced Research in Computer and Communication Engineering ISO 3297:2007 Certified Vol. 6, Issue 4, April 2017.
6. Aniket Dole, “Intelligent Chat Bot for Banking System”, International Journal for Research in Applied Science & Engineering Technology (IJRASET) Volume 4 Issue IV, April 2016.
7. Nuruzzaman M, Hussain OK (2018) “A survey on chatbot implementation in customer service industry through deep neural networks.” In: 2018 IEEE 15th international conference on e-business engineering (ICEBE), IEEE.
8. Nikita Hatwar, “AI BASED CHATBOT”, International Journal of Emerging Trends in Engineering and Basic Sciences (IJEEBS) Volume 3, Issue 2 (MarchApril 2016)

9. Hubtype, 2018. “Rule-Based vs AI Chatbots.” Available from: https://www.hubtype.com/blog/rulebased-vs-ai-chatbots/. [Accessed: 22 May 2019].
10. Hristidis V (2018), “Chatbot technologies and challenges.” In: 2018 first international conference on artificial intelligence for industries (AI4I), IEEE.
11. Iren Korkishko “UI/UX design guide with terms, explanations, tips and trends”, 2019, – Режим доступу до ресурсу: https://medium.com/swlh/ui-ux-design-guide-with-terms-explanations-tips-and-trends-754b9356d914.
12. Setiaji B, Wibowo FW (2016), “Chatbot using a knowledge in database: human-to-machine conversation modeling.” In: 2016 7th international conference on intelligent systems, modelling and simulation (ISMS), IEEE.
13. International Software Testing Qualifications Board, “Certified Tester Foundation Level (CTFL) Syllabus. Version 2018 v3.1.1”, 2018, https://astqb.org/assets/documents/CTFL-2018-Syllabus.pdf
14. “Top 12 Database Design Principles in 2023” [Електронний ресурс] – Режим доступу до ресурсу: https://vertabelo.com/blog/database-design-principles/
15. Olayele Adelakun, Sergio Galvan-Cruz & Fen Wang. Impacts of IDEF0-Based Models on the Usefulness, Learning, and Value Metrics of Scrum and XP Project Management Guides [Електронний ресурс] – Режим доступу до ресурсу: https://www.tandfonline.com/doi/abs/10.1080/10429247.2021.1958631
16. MERN in client-server architecture [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/mern-stack/ (Дата звернення 11.11.2023)
17. “What is Use Case Diagram?” [Електронний ресурс], Режим доступу до ресурсу: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/
18. Cooper G., "Examining Science Education in ChatGPT: An Exploratory Study of Generative Artificial Intelligence," Journal of Science Education and Technology32, 22 March 2023, https://doi.org/10.1007/s10956-023-10039-y

19. H. E. . Nugroho and A. Nugroho. (2021, Mei 1). Analisis Dan Perancangan E-Commerce Pada Toko Sepatu Dope13store Menggunakan Framework Laravel. Diakses pada 1 November 2021, dari https://jurnal.amikom.ac.id/index.php/infos/article/view/565.
20. Ponsen Sindu Prawito dan Rahadi.(2020, Desember). “Perancangan Sistem Informasi Toko Online Berbasis Web dengan Menggunakan Laravel dan Api Rajaongkir.” Diakses pada 1 November 2021, dari https://www.jurnal.syntaxliterate.co.id/index.php/syntax-literate/article/view/1849.
21. Okstania, C., Novita, M., & Latifah, K. (2019). "Sistem Informasi Media Transparansi Program Pemberdayaan Kesejahteraan Keluarga (Pkk) Dan Event Reminder Dengan Api Telegram Berbasis Web Pada Rw 3 Kelurahan Karang Tempel Semarang." Science And Engineering National Seminar 4 (SENS 4), 4(1).
22. Olayele Adelakun, Sergio Galvan-Cruz & Fen Wang. Impacts of IDEF0-Based Models on the Usefulness, Learning, and Value Metrics of Scrum and XP Project Management Guides [Електронний ресурс] – Режим доступу до ресурсу: https://www.tandfonline.com/doi/abs/10.1080/10429247.2021.1958631
23. Pratik Salve, “College Enquiry Chat Bot”, International Journal on Recent and Innovation Trends in Computing and Communication ISSN: 2321-8169 Volume:5 Issue: 3.
24. Stefanus Ardhito Prasetya, Alva Erwin, Maulahikmah Galinium, “Implementing a Chatbot for E-Commerce Site Using Artificial Intelligence Markup Language (AIML)” - Faculty of Engineering and Information Technology, Swiss German University, 2018.
25. Iren Korkishko “UI/UX design guide with terms, explanations, tips and trends”, 2019, – Режим доступу до ресурсу: https://medium.com/swlh/ui-ux-design-guide-with-terms-explanations-tips-and-trends-754b9356d914.
26. Bayu Setiaji, “Chatbot Using A Knowledge in Database”, International Conference on Intelligent System, Modling and Simulation, 2016.

27. Carlos Binker, Hugo Tantignone, Eliseo Zurdo, Guillermo Buranits, "Acceso remoto a dispositivos IoT mediante técnicas de mensajería empleando bots y freertos" ,Computer Science, 2020.
28. Nofiati Nofiati, April Firman Daru, “Sistem Informasi Perpustakaan Berbasis Web Dengan Framework Laravel”, Information Science and Library, 2021.
29. Yi Yang, Wen-tau Yih, and Christopher Meek. 2015. “Wikiqa: A challenge dataset for opendomain question answering.” In Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing. Association for Computational Linguistics.
30. Annabell Ho, Jeff Hancock, and Adam S Miner. 2018. Psychological, Relational, and Emotional Effects of Self-Disclosure After Conversations With a Chatbot. J Commun 68, 4 (August 2018), 712–733. DOI:https://doi.org/10.1093/joc/jqy026
31. Cui et al., 2017. “The system overview of SuperAgent Chatbot.” Available from https://www.aclweb.org/anthology/P17-4017
32. MERN in client-server architecture [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/mern-stack/ (Дата звернення 11.11.2023)
33. Alia Sa’ad Eldin AbuSahyon, Omar Alshorman, Basim Al-Absi, “Investigating The Impact of AI- Driven Chatbots On the Acquisition of English as A Foreign Language Among Saudi Undergraduate Students”, International Journal of Membrane Science and Technology: Vol. 10 No. 2 (2023).
34. C. Anderson, “The Long Tail: Why the Future of Business is Selling More for Less”, 2006.
35. K. Long et al., “‘Could you define that in bot terms’?,” in Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems, May 2017, doi: 10.1145/3025453.3025830.
36. P. Langley, C. Thompson, R. Elio, and A. Haddadi, “An adaptive conversational interface for destination advice,” in Lecture Notes in Computer Science, Springer Berlin Heidelberg, 1999, doi: 10.1007/3-540-48414-0_24.

37. A. D.Angeli, G. I. Johnson, and L. Coventry, “The unfriendly user: exploring social reactions to chatterbots,” in Proceedings of The International Conference on AffectiveHuman Factors Design, 2001
38. O. Gambino, A. Augello, A. Caronia, G. Pilato, R. Pirrone, and S. Gaglio, “Virtual conversation with a real talking head,” in 2008 Conference on Human System Interactions, May 2008, doi: 10.1109/HSI.2008.4581446.
39. Ortegón-Cortázar, L., Royo-Vela, M., “Attraction factors of shopping centers.” European Journal of Management and Business Economics, 26(2), 2017.
40. Mejri, A., Dhruv, B. (2014), CSR: “Consumer responses to the social quality of private labels. Journal of Retailing and Consumer Services.”, 21(3), 2014.

ДОДАТОК А

Деталізація мети проекту методом SMART

Продуктами дипломного проекту є інформаційна система підтримки клієнтів інтернет-магазину одягу та віртуальний асистент Telegram bot.
Результати деталізації методом SMART розміщені у табл. А.1.

Таблиця А.1 – Формалізація мети за технологією SMART

Specific	Розробка віртуального асистента Telegram bot для онлайн-магазину printcrafters.com.ua.
Measurable	Запуск віртуального асистента Telegram bot для спрощення пошуку товарів, пропонування релевантного товару і заохочення клієнта.
Achievable	Є узгоджене технічне завдання, також знання необхідних мов програмування та інструментів розробки: PHP, Laravel, Visual Studio Code, MySql.
Revelant	Покращення задоволеності клієнтів та підвищення продажів за рахунок підвищення якості взаємодії з клієнтом та використання штучного інтелекту.
Time-bound	Завершення усіх запланованих робіт до 20 листопада 2023 року, а також запуск в експлуатацію до кінця 2023 року.

Джерело: Розроблено автором

Таблиця А.2 – Виконавці проекту.

Роль	ПІБ	Проектна роль
Developer 1	Поливяний А.	Frontend та backend
Developer 2	Бараннік Д.	Інтеграція та функціонал Telegram bot асистента
DevOps	Поливяний А.	Автоматизація процесу розгортання проекту та неперервна інтеграція
BA	Поливяний А.	Встановлення бізнес-вимог, переклад бізнес-вимог у технічні специфікації
Project Manager	Поливяний А.	Слідкування за дотриманням термінів виконання, розподіл ресурсів та завдань серед учасників. Збір та обробка інформації
Керівник проекту	Неня А.В.	Формування завдання на розробку проекту
Web Artist	Бараннік Д.	Дизайн інтерфейсу і користувацького досвіду
Support Team	Бараннік Д.	Надання технічної підтримки клієнтам та усунення несправностей
QA	Бараннік Д.	Контроль та забезпечення якості продукту

Джерело: Розроблено автором

Структура розподілу робіт (WBS)

Структура декомпозиції робіт (WBS) у проектному менеджменті є орієнтованою на доконане виконання проекту декомпозицією проекту на менші частки. Структура декомпозиції робіт є ключовою часткою робіт по проекту, яка організовує командну роботу по проекту у керовані частини.

WBS є ієрархічною декомпозицією проекту у фази, кінцеві результати та пакети робіт. Вона є ієрархічною структурою, що показує подальший розподіл необхідних для виконання мети зусиль; наприклад, програма, проект чи договір. У проекті чи договорі, розробка WBS відбувається, починаючи з кінцевих цілей та успішного розподілу її у керовані частини, що можуть бути оцінені за критеріями розміру, тривалості та відповідальностей (наприклад, системи, підсистеми, компоненти, задачі, підзадачі та пакети робіт) та включають усі необхідні для досягнення мети проекту кроки.

Система декомпозиції робіт надає загальний каркас для природнього розвитку загального планування та контролю договору і є базисом для розподілу роботи на частини, що можуть бути визначеними, та з яких може бути зроблене Технічне Завдання і установлені звіти по технічним даним, графікам, вартостям, робочим годинам.

Структура декомпозиції робіт дозволяє зібрати докупи підлеглі витрати по задачах, матеріалах тощо на вищий рівень "батьківських" задач, матеріалів тощо. Для кожного елементу структури декомпозиції робіт генерується опис задачі, що має бути виконаною. Ця техніка (іноді називається структурою декомпозиції системи використовується для визначення і налагодження сумарних рамок проекту.

WBS організовується навколо ключових продуктів проекту (чи запланованих результатів), а не необхідних робіт для випуску продукту (заплановані дії). Так як заплановані результати є бажаним завершенням проекту, вони формують відносно стабільний набір категорій, у яких ціни запланованих для їх досягнення необхідних дій можуть бути зібрані докупи. Добре розроблена WBS робить легко досяжним

призначення кожної діяльності проекту до виключно однієї термінальної події у WBS. Додатково до її функцій у обліку витрат WBS також допомагає співвіднести вимоги одного рівня системних специфікацій до іншого, наприклад, відповідність матриці вимог перехресних посилань до функціональних вимог на вищий чи нижчий рівні документації.

Розробка WBS зазвичай має відбуватися на початку проекту і перед детальним плануванням проекту і задач. WBS є попереднім етапом, основою для розробки мережевих і календарних планів, які потребують повного переліку всіх робіт за проектом, які можна отримати, маючи пакети робіт. WBS наочно демонструє весь обсяг робіт і місце окремих виконавців.

Основні етапи розробки WBS:

- визначення ступеня деталізації проектних робіт (так, щоб вони піддавалися оцінці);
- визначення кількості рівнів (як правило, три-чотири, для сучасних компаній чотири – оптимально);
- розробка структури кожного рівня (формуються горизонтальні рівні);
- підготовка опису елементів WBS (коротка назва кожної складової WBS);
- формування системи кодування (кодуються всі блоки);
- проведення зворотних обчислень (витрати знизу догори за принципом: відділ локалізації – субпідрядник).

WBS може застосовуватися для об'єднання робіт, які необхідно виконати, організаційних структур і відповідальності за роботу з підсистемами планування, оцінки, розподілу витрат і ресурсів, аналізу, контролю та обліку в єдину взаємопов'язану інтегровану систему управління проектом. Будуємо таблицю WBS.

На рисунку А.1 представлено WBS-структуру робіт проекту

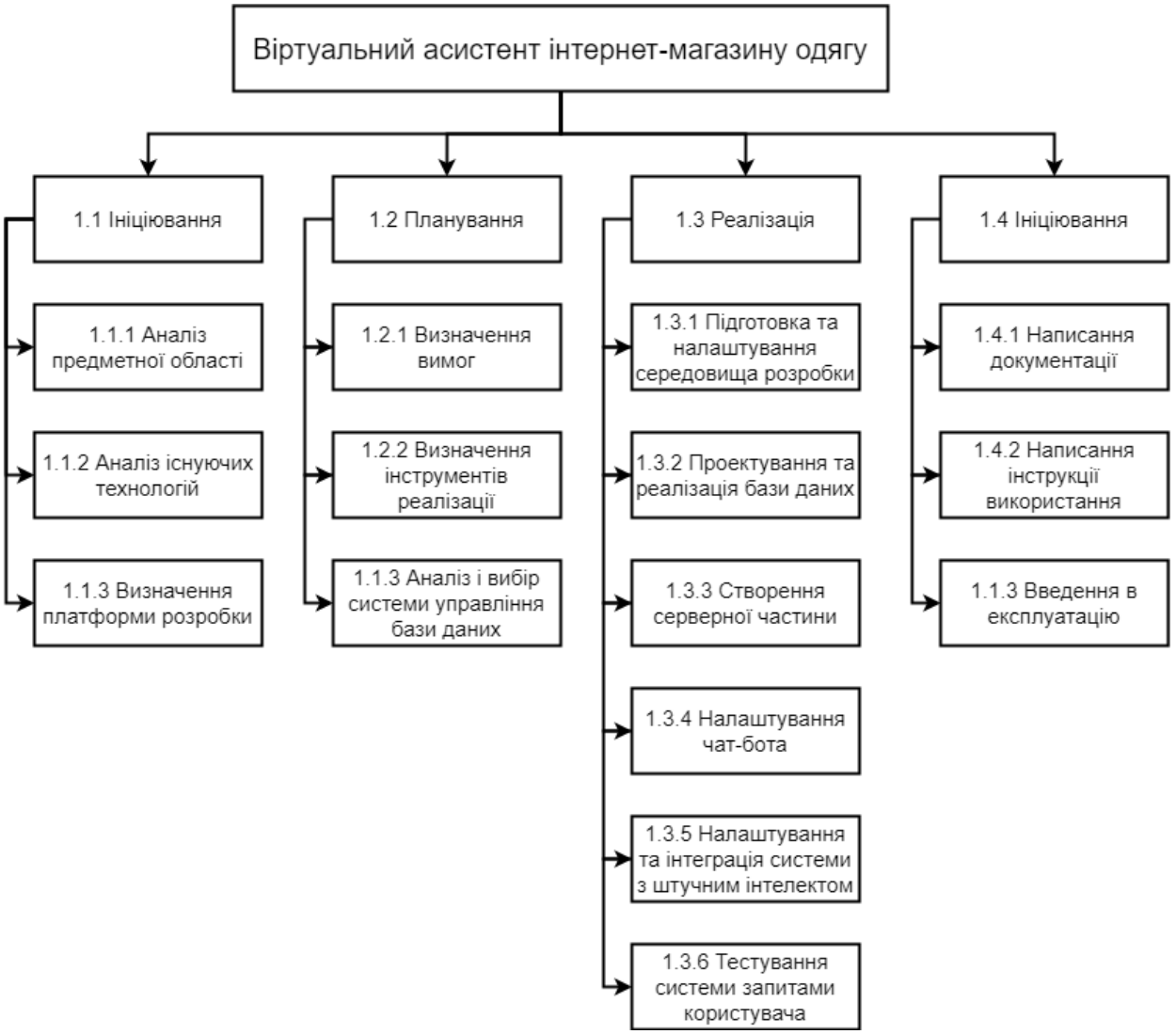

Рисунок А.1 – WBS-структура робіт проекту.

Джерело: Розроблено автором

Організаційна структура робіт (OBS)

Ця структура стосується тільки внутрішньої організаційної структури проекту і не зачіпає відносин проектних груп або учасників з батьківськими організаціями. Будується OBS аналогічно робочої структури, а саме:

- на першому рівні відображається організаційна структура як єдиний елемент;
- на другому і нижчих рівнях ділиться нижча частина структури на основні організаційні елементи.

Цей процес повторюється до найбільш низького рівня - базових робочих груп (змішаних цільових або функціональних), а при реалізації малих проектів - до окремих виконавців.

Обсяг робіт для цих найнижчих організаційних рівнів являє собою найбільш низькі елементи WBS, кожен з яких можна планувати і контролювати як окремі одиниці. Саме таке правило діє для створення OBS. Кількість рівнів залежить від розміру проекту.

Об'єднання робочої та організаційної структури дає можливість інтегрувати, планувати і контролювати роботу та порівнювати її виконання по підрозділах і організації взагалі. Кожен менеджер в цій ієрархії має свій набір планів і звітів за своїми сферами відповідальності. Як вже підкреслювалося, розподіл WBS здійснюється до робочого пакету, який виконується окремою групою. OBS, в свою чергу, розбивається до рівня груп, які виконують найнижчий рівень робіт в WBS. Таким чином, роботи найнижчого рівня WBS притаманні як WBS, так і OBS, тобто це - фундаментальні блоки обох структур. Будуємо структуру OBS (рис. А.2).

На рисунку А.2 представлено OBS-структуру робіт проекту.

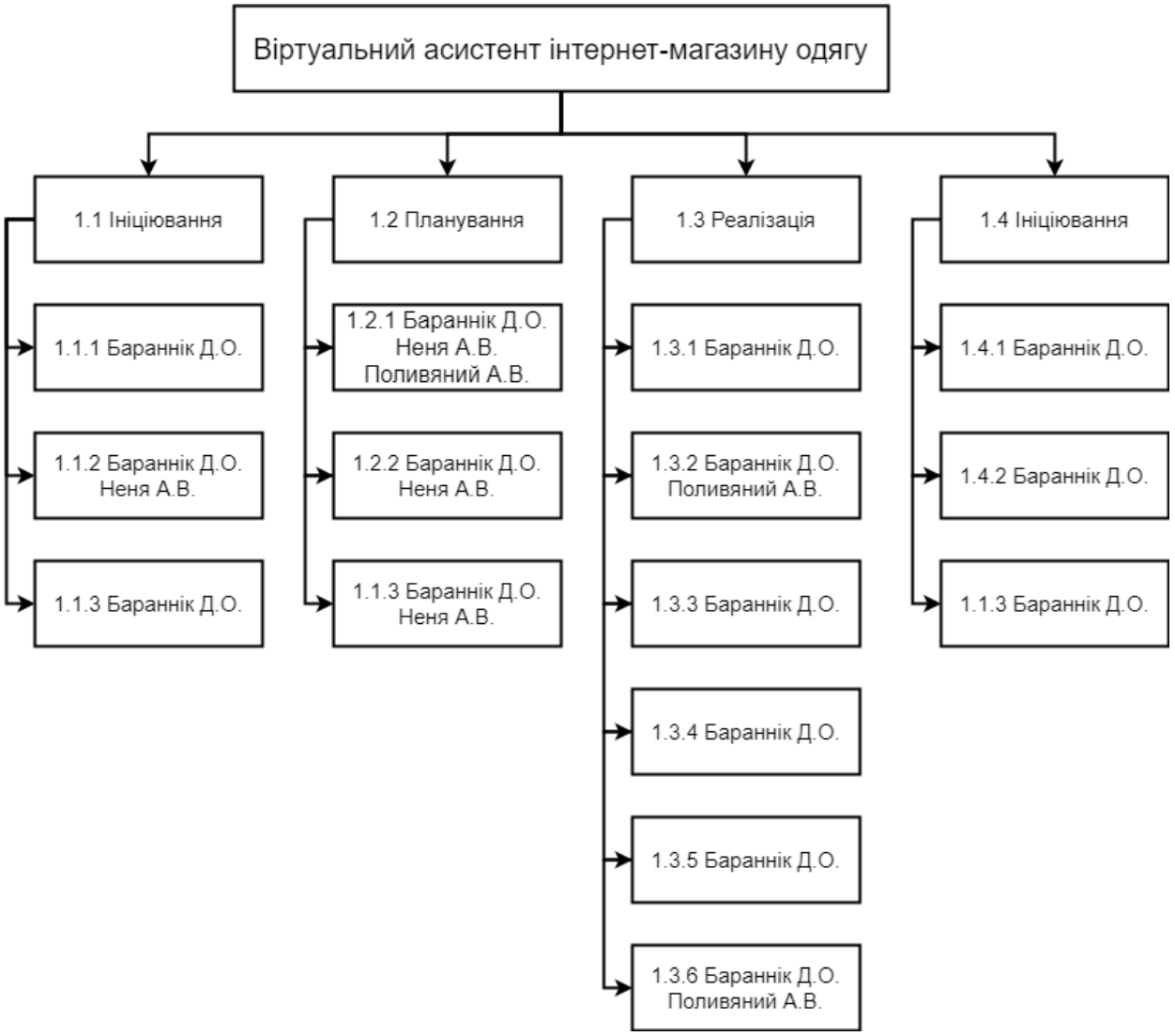

Рисунок А.2 – OBS-структура робіт проекту.

Джерело: Розроблено автором

Діаграма Ганта

Для того, щоб мати реальне уявлення про тривалість виконання робіт з урахуванням обмеженості у використанні ресурсів, на підставі часткових мережевих

моделей, а також, проекту в цілому з урахуванням вихідних і святкових днів, будують календарний графік робіт.

Він є реальним розподілом робіт по пакету по календарними датами, тобто своєрідним розкладом виконання робіт. Діаграма Ганта є досить зручним для користування. Діаграма Ганта представляє собою відрізки, які розміщені на горизонтальній шкалі часу. Кожен відрізок відповідає окремому завданню або підзадачі. Завдання і підзадачі, як складова плану, розміщуються по вертикалі. Початок, кінець і довжина відрізків на шкалі часу відповідають початку, кінця і тривалості завдання.

Календарний план проекту показаний на рисунку А3.

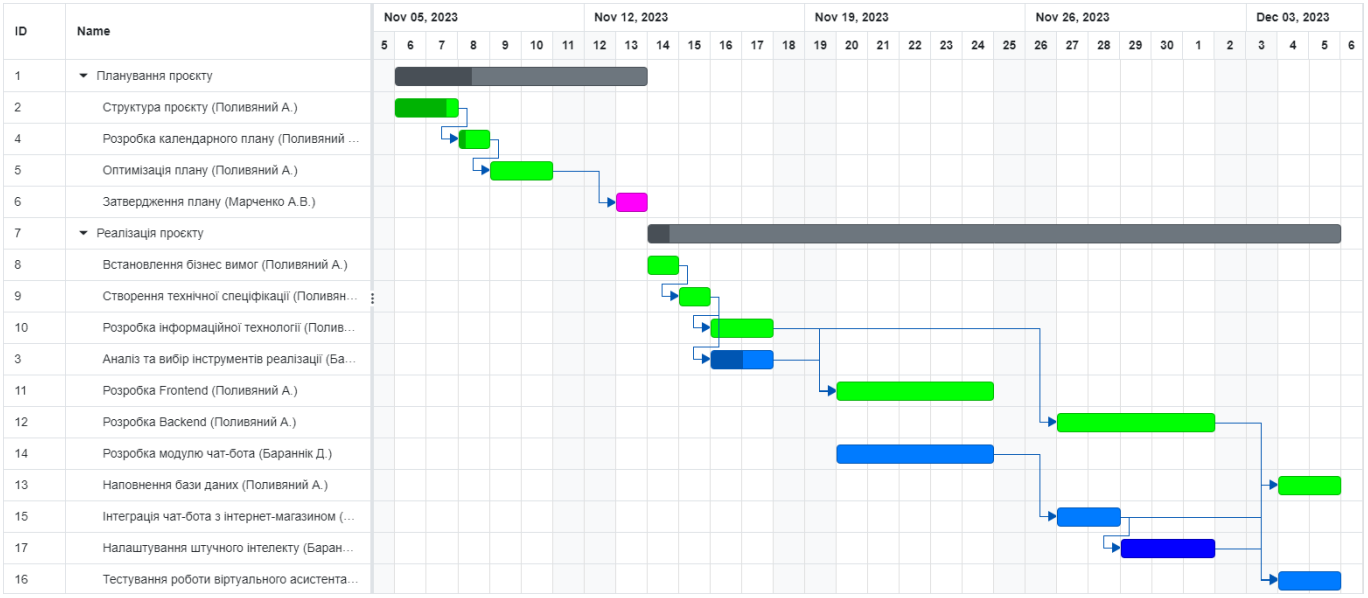

Рисунок А.3 – Діаграма Ганта.

Джерело: Розроблено автором

Управління ризиками

До основних ризиків розробки Web-додатку для інформаційної системи підтримки діяльності сервісного центру «Сігма-сервіс» є:

- неглибоке вивчення предметної області;
- збільшення навантаження під час реалізації проекту;
- зміна цілей у ході реалізації проекту;
- людський фактор;
- відмова обладнання;
- відсутність кваліфікованого програміста;
- зміна строків виконання роботи;
- незнаходження спільної мови між керівником і виконавцем;
- зростання вимог до проекту.

Таблиця А.3. Ймовірність виникнення і величина ризику

№	Ризики	Виникнення	Втрати
1	Кібербезпека та злам системи	2	4
2	Відмова в обслуговуванні (DoS) атаки	3	4
3	Технічні несправності та відмови в роботі	3	6
4	Проблеми з конфіденційністю даних	2	3
5	Неадекватна обробка та зберігання даних	4	5
6	Застарілість та неактуальність інформації	1	2

Джерело: Розроблено автором

Таблиця А.4 – Матриця впливу.

Вірогідність виникнення	Матриця впливу				
5		5	3	5	
4					
3			2		
2	6			1	
1					
Ступінь впливу	1	2	3	4	5

Джерело: Розроблено автором

ДОДАТОК Б

Лістинг програмного коду

config\assistant.php

```
<?php
return [
    'default' => env('DEFAULT_AI_ASSISTANT_ID')
];
```

app\Services\AssistantService.php

```
<?php

namespace App\Services;

use App\Contracts\AssistantRepositoryInterface;
use App\Contracts\AssistantServiceInterface;
use App\Contracts\AssistantThreadRepositoryInterface;
use App\Enums\AssistantThreadRunStatusEnum;
use App\Models\Assistant;
use App\Models\TelegramChat;
use Illuminate\Support\Facades\Log;
use OpenAI;

class AssistantService implements AssistantServiceInterface
{
    private OpenAI\Client $client;
    private mixed $assistantRepository;
    /**
     * @var
AssistantThreadRepositoryInterface|(AssistantThreadRepositoryInterface
&\Illuminate\Contracts\Foundation\Application)|\Illuminate\Contracts\F
oundation\Application|\Illuminate\Foundation\Application|mixed
     */
    private mixed $assistantThreadRepository;

    public function __construct()
    {
        $yourApiKey = getenv('OPEN_AI_API_KEY');
        $this->client = OpenAI::client($yourApiKey);
```

```
            $this->assistantRepository =
app(AssistantRepositoryInterface::class);
            $this->assistantThreadRepository =
app(AssistantThreadRepositoryInterface::class);

        }

        public function runThread(TelegramChat $telegramChat): string
        {
            $assistant = $this->getAssistant();
            $assistantThread =
$telegramChat->getAssistantThread($assistant);
            if ($telegramChat->type == 'private') {
                $clientName = !($telegramChat->firstName &&
$telegramChat->lastName)
                    ? $telegramChat->firstName . ' ' .
$telegramChat->lastName
                    : $telegramChat->username;
            } else {
                $clientName = $telegramChat->title;
            }
            $threadTitle = 'Assisting ' . $clientName;
            $message = $telegramChat->messages->last()->text;
            if (!$assistantThread) {
                $ai_run = $this->client->threads()->createAndRun(
                    [
                        'assistant_id' => $assistant->assistant_id,
                        'thread' => [
                            'metadata' => [
                                'title' => $threadTitle
                            ],
                            'messages' =>
                                [
                                    [
                                        'role' => 'user',
                                        'content' => $message,
                                    ],
                                ],
                        ],
                    ],
                );
                $ai_thread_id = $ai_run->threadId;
                $ai_run_id = $ai_run->id;

$this->assistantThreadRepository->createAssistantThread($assistant,
$ai_run->threadId, $telegramChat->id);
            } else {
                $ai_thread_id =
$this->client->threads()->retrieve($assistantThread->thread_id)->id;
                $responseMessage =
$this->client->threads()->messages()->create($ai_thread_id, [
                    'role' => 'user',
                    'content' => $message,
```

```
                ]);
                $ai_run_id = $this->createRun($ai_thread_id,
$assistant->assistant_id)->id;
            }
            $completedRun = $this->checkingRunStatus($ai_thread_id,
$ai_run_id);
            if ($completedRun->status ===
AssistantThreadRunStatusEnum::COMPLETED->value) {
                return
$this->getLastAssistantMessageText($assistantThread) ??
__('assistant.smth_went_wrong');
            } else if (isset($completedRun->lastError)) {
                Log::error('Open AI Error: ' .
$completedRun->lastError->message);
            }
            return __('assistant.smth_went_wrong');
        }

        private function checkingRunStatus($ai_thread_id, $ai_run_id)
        {
            do {
                $response = $this->getRunById($ai_thread_id,
$ai_run_id);
                sleep(1);
            } while ($response->status ===
AssistantThreadRunStatusEnum::IN_PROGRESS->value);

            return $response;
        }

        private function getRunById(string $threadId, string $runId):
OpenAI\Responses\Threads\Runs\ThreadRunResponse
        {
            return $this->client->threads()->runs()->retrieve(
                threadId: $threadId,
                runId: $runId,
            );
        }

        private function getRunsList(string $threadId):
OpenAI\Responses\Threads\Runs\ThreadRunListResponse
        {
            return $this->client->threads()->runs()->list(
                threadId: $threadId,
                parameters: [
                    'limit' => 10,
                ],
            );

        }

        private function createRun($threadId, $assistantId)
        {
```

```
            return $this->client->threads()->runs()->create(
                threadId: $threadId,
                parameters: [
                    'assistant_id' => $assistantId,
                ],
            );
        }

        private function getRunStepList($threadId, $runId)
        {
            return $this->client->threads()->runs()->steps()->list(
                threadId: $threadId,
                runId: $runId,
                parameters: [
                    'limit' => 10,
                ],
            );
        }

        private function getThreadMessages($assistantThread):
OpenAI\Responses\Threads\Messages\ThreadMessageListResponse
        {
            return
$this->client->threads()->messages()->list($assistantThread->thread_id
, [
                'limit' => 10,
            ]);
        }

        public function getAssistant(): Assistant
        {
            $assistant =
$this->assistantRepository->getDefaultAssistant();
            if (!$assistant) {
                $response = $this->client->assistants()->list([
                    'limit' => 10,
                ]);
                $assistant_data =
collect($response->data)->filter(function ($assistant) {
                    return $assistant->id ===
config('assistant.default');
                })->first();
                $assistant =
$this->assistantRepository->storeAssistant($assistant_data);
            }
            return $assistant;
        }

        private function
getLastAssistantMessageText($assistantThread)
        {
            $messages = $this->getThreadMessages($assistantThread);
```

```
            return
collect($messages->data)->first()?->content[0]->text->value;
        }
    }
```

app\Models\Assistant.php

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\HasMany;

class Assistant extends Model
{
    use HasFactory;

    protected $fillable = [
        'assistant_id',
        'name',
        'instructions',
        'model',
        'description',
        'tools',
        'fileIds',
        'metadata',
        'createdAt',
        'default'
    ];

    protected $casts = [
        'tools' => 'json',
        'fileIds' => 'json',
        'metadata' => 'json',
        'createdAt' => 'datetime'
    ];

    public function threads(): HasMany
    {
        return $this->hasMany(AssistantThread::class);
    }
}
```

app\Models\AssistantThread.php

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class AssistantThread extends Model
{
    use HasFactory;

    protected $fillable = [
        'user_id',
        'assistant_id',
        'telegram_chat_id',
        'thread_id',
        'ip',
        'useragent'
    ];

    public function telegramChat()
    {
        return $this->belongsTo(TelegramChat::class);
    }
}
```

app\Jobs\HandleRequestToAssistant.php

```
<?php

namespace App\Jobs;

use App\Contracts\AssistantServiceInterface;
use App\Contracts\TelegramServiceInterface;
use App\Models\TelegramChat;
use Illuminate\Bus\Queueable;
use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Foundation\Bus\Dispatchable;
use Illuminate\Queue\InteractsWithQueue;
use Illuminate\Queue\SerializesModels;

class HandleRequestToAssistant implements ShouldQueue
{
            use   Dispatchable,   InteractsWithQueue,   Queueable,
SerializesModels;

    private TelegramChat $telegramChat;
```

```
        /**
         * Create a new job instance.
         */
        public function __construct(TelegramChat $telegramChat)
        {
            $this->telegramChat = $telegramChat;
        }

        /**
         * Execute the job.
         */
        public function handle(): void
        {
                                                        $response      =
app(AssistantServiceInterface::class)->runThread($this->telegramChat);

app(TelegramServiceInterface::class)->sendMessage($response,
$this->telegramChat->chat_id);
        }
    }
```

app\Repositories\AssistantThreadRepository.php

```
    <?php

    namespace App\Repositories;

    use App\Contracts\AssistantThreadRepositoryInterface;
    use App\Models\Assistant;
    use App\Models\AssistantThread;

    class             AssistantThreadRepository             implements
AssistantThreadRepositoryInterface
    {

          public function createAssistantThread(Assistant $assistant,
$threadId, $telegramChatId, $userId = null)
        {
            $ip = geoip()->getClientIP();
            return AssistantThread::query()->create([
                'assistant_id' => $assistant->id,
                'thread_id' => $threadId,
                'user_id' => $userId,
                'telegram_chat_id' => $telegramChatId,
                'ip' => $ip,
                'useragent' => request()->header('User-Agent')
            ]);
        }
```

```
}
```

app\Repositories\AssistantRepository.php

```
<?php

namespace App\Repositories;

use App\Contracts\AssistantRepositoryInterface;
use App\Models\Assistant;
use Illuminate\Database\Eloquent\Builder;
use Illuminate\Database\Eloquent\Model;
use OpenAI\Responses\Assistants\AssistantResponse;

class AssistantRepository implements AssistantRepositoryInterface
{

    public function getDefaultAssistant()
    {
                return  Assistant::query()->where('default',  '=',
1)->get()->first();
    }

     public function storeAssistant(AssistantResponse $response):
Model|Builder
    {
        return Assistant::query()->createOrFirst([
            'assistant_id' => $response->id,
            'name' => $response->name,
            'instructions' => $response->instructions,
            'model' => $response->model,
            'description' => $response->description,
            'tools' => $response->tools,
            'fileIds' => $response->fileIds,
            'metadata' => $response->metadata,
            'createdAt' => $response->createdAt,
                    'default'  =>  config('assistant.default')  ===
$response->id
        ]);
    }
}
```

app\Contracts\AssistantServiceInterface.php

```
<?php
```

```
namespace App\Contracts;

use App\Models\Assistant;
use App\Models\TelegramChat;

interface AssistantServiceInterface
{
        public function runThread(TelegramChat $telegramChat): string;

    public function getAssistant(): Assistant;
}
```

app\Contracts\AssistantThreadRepositoryInterface.php

```
<?php

namespace App\Contracts;

use App\Models\Assistant;

interface AssistantThreadRepositoryInterface
{
      public function createAssistantThread(Assistant $assistant, $threadId, $telegramChatId, $userId = null);
}
```

app\Contracts\AssistantRepositoryInterface.php

```
<?php

namespace App\Contracts;

use OpenAI\Responses\Assistants\AssistantResponse;

interface AssistantRepositoryInterface
{
    public function getDefaultAssistant();

    public function storeAssistant(AssistantResponse $response);
}
```

database\migrations\2023_12_11_100003_create_assistants_table.php

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration {
    /**
     * Run the migrations.
     */
    public function up(): void
    {
         Schema::create('assistants', function (Blueprint $table)
{
            $table->id();
            $table->string('assistant_id');
            $table->string('name')->nullable();
            $table->text('instructions')->nullable();
            $table->string('model');
            $table->text('description')->nullable();
            $table->jsonb('tools')->nullable();
            $table->jsonb('fileIds')->nullable();
            $table->jsonb('metadata')->nullable();
            $table->string('createdAt')->nullable();
            $table->boolean('default')->default(false);
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('assistants');
    }
};
```

database\migrations\2023_12_11_100006_create_assistant_threads_table.php

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;
```

```
    return new class extends Migration {
        /**
         * Run the migrations.
         */
        public function up(): void
        {
              Schema::create('assistant_threads', function (Blueprint
$table) {
                $table->id();
                $table->unsignedBigInteger('user_id')->nullable();
                $table->unsignedBigInteger('assistant_id');

$table->unsignedBigInteger('telegram_chat_id')->nullable();
                $table->string('thread_id')->nullable();
                $table->string('ip')->nullable();
                $table->string('useragent')->nullable();
                $table->timestamps();

$table->foreign('user_id')->references('id')->on('users')->onDelete('s
et null');

$table->foreign('telegram_chat_id')->references('id')->on('telegram_ch
ats')->onDelete('set null');

$table->foreign('assistant_id')->references('id')->on('assistants')->o
nDelete('cascade');
            });
        }

        /**
         * Reverse the migrations.
         */
        public function down(): void
        {
            Schema::dropIfExists('assistant_threads');
        }
    };
```