МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
Факультет електроніки та інформаційних технологій

Кафедра комп‘ютерних наук

Кваліфікаційна робота магістра

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ ПРОЄКТУВАННЯ ПЛАТФОРМИ ЕЛЕКТРОННОЇ КОМЕРЦІЇ НА БАЗІ ТЕЛЕГРАМ

Здобувач освіти гр. ІН.м-12ан/2у — Михайло БІРІНЦЕВ

Науковий керівник,
ст. викл., к.т.н. — Борис КУЗІКОВ

В. о. завідувача кафедри
доцент, к.т.н. — Ігор ШЕЛЕХОВ

Суми 2022

Сумський державний університет
(назва вузу)

Факультет *ЕлІТ* Кафедра *Комп'ютерних наук*

Спеціальність *«122 - Комп'ютерні науки»*

Затверджую:
В.О зав.кафедрою Шелехов І.В.

"_____" _____ 20___р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ МАГІСТРА

Бірінцеву Михайлу Олександровичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Інформаційна технологія проєктування платформи електронної комерції на базі Телеграм*

затверджую наказом по інституту від "____" _____ 20 ___ р. № _____

2. Термін здачі студентом закінченого роботи_____

3. Вхідні данні до роботи _____
_____
_____
_____

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Ознайомитися з теоретичними основами інформаційних технологій електронної комерції; 2) Провести аналіз аналогічних технологій е-комерції; 3) Постановка завдання щодо розробки експериментальної технології е-комерції; 4) Розробка програмного продукту – інформаційної технології платформи електронної комерції на базі Телеграм; 5) Аналіз результатів.*

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) _____
_____

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання _____

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів дипломного проекту (роботи)	Термін виконання проекту (роботи)	Примітка
1.	Ознайомлення з теоретичними основами електронної комерції		
2.	Постановка задачі та формування завдань дослі-дження.		
3.	Огляд аналогічних рішень.		
4.	Розробка додатку з використанням Telegram WebApp Bot API та Spring		
5.	Оформлення пояснювальної записки до кваліфікаційної магістерської роботи		

Студент – дипломник _____
(підпис)

Керівник проекту _____
(підпис)

РЕФЕРАТ

Записка: 81 стор., 29 рис., 4 таблиці, 2 додатки, 23 літературних джерел.

Об’єкт дослідження — Процес побудови комплексу програмного та інформаційного забезпечення для організації систем електронної комерції на базі месенджерів.

Мета роботи — розробка WEB платформи на з використанням Spring Boot та Telegram WebApp Bots API для систем електронної комерції.

Результати — проведений попередній та детальний аналіз сфери додатків вбудованих у месенджери та досліджений досвід готових аналогів. Після вивчення особливостей схожих рішень та аналізу їх переваг і недоліків, була розроблена WEB-орієнтована платформа електронної комерції на базі месенджеру Telegram, що дозволило скористатися популярністю цього месенджеру. Дана розробка має спростити оформлення , відстежування і виконання замовлень клієнтами, а також допомогти у проведенні інвентаризаційних заходів продавцеві. Платформа була розроблена з використанням мови програмування Java (Spring Boot), а також JavaScript.

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ ПРОЄКТУВАННЯ ПЛАТФОРМИ ЕЛЕКТРОННОЇ КОМЕРЦІЇ НА БАЗІ ТЕЛЕГРАМ, INFORMATION TECHNOLOGY FOR DESIGNING AN E-COMMERCE TELEGRAM-BASED PLATFORM, TELEGRAM, SPRING BOOT, E-COMMERCE

ЗМІСТ

ВСТУП

У сучасному світі, де людині майже завжди не вистачає часу, а іноді і фізичних сил, щоб відвідувати магазини та на довгі та нудні подорожі по них. Найпростіший варіант – зайти в веб-сервіс і вибрати потрібний товар. Сучасні інтернет-магазини, наприклад Lamoda[21], дозволяють вибрати певну кількість розмірів для доставки, щоб клієнт мав право приміряти речі, і вибрати потрібне. У сучасному світі існує багато інтернет-магазинів, наприклад: Lamoda, Aliexpress, Amazon, Ebay. Вони поділяються на: господарські магазини, магазини одягу, спортивні магазини, вузькоспеціалізовані магазини і т.д. Для залучення клієнтів інтернет-магазини організовують акцій і проводять рекламні кампанії. І при успішній рекламній кампанії вони залишаються в плюсі. Наприклад такі компанії як Amazon чи Ebay вклавши великі кошти в рекламу мають тепер величезний прибуток.

Розробка веб-сайтів є досить поширеною діяльністю. Найбільшим досягненням у технології розробки веб-сайтів є розробка веб-сервісів[3]. Інструменти управління контентом, наприклад Joomla, WordPress, Thunder та інші, широко використовуються в Інтернет-просторі при розробці веб-сайтів будь-якої складності.

Основною рисою систем управління торгівлею товарами, це можливість розробляти на практиці сайти будь-якого рівня складності з використанням Telegram ботів[10].

Виходячи із актуальності, темою роботи обрана: інформаційна технологія платформи електронної комерції на базі Телеграм.

Метою роботи є створення платформи електронної комерції з продажу електроніки на базі Telegram.

Завданнями випускного проекту, відповідно до мети, є:

1) Аналіз та опис предметної області;

2) Постановка завдання на створення платформи електронної комерції;

3) Створення методу вирішення проблеми створення веб-сервісу;

4) Підбір програмних засобів для реалізації створення платформи електронної комерції;

5) Проектування та опис архітектури веб-сервісу;

6) Програмна реалізація інтернет-магазину.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ЕЛЕКТРОННОЇ КОМЕРЦІЇ

1.1. Зміст та класифікація типів систем електронної комерції

Опис предметного середовища

Платформи електронної комерції – це веб платформа, яка рекламує товари чи послуги, приймає та обробляє заявки-замовлення на покупку певних товарів та послуг, надає користувачеві можливість вибору виду та варіанту оплати, виду покупки та оформлення замовлення рахунок на оплату[8].

Запровадження веб-платформи має такі цілі:

– збільшення ринку та збільшення кількості клієнтів;

– зниження витрат на оплату праці менеджерів з продажу;

– зниження витрат на постачання основних видів діяльності (збут);

– зменшення кількості помилок у звітності та збільшення швидкості її складання.

Існуючі найбільш популярні платформи електронної комерції, такі як Amazon чи Aliexpress, мають типові особливості, а саме:

– можливість реєстрації користувачів;

– можливість створення замовлення з будь-якої кількості товарів і послуг;

– наявність багаторівневих категорій товарів і послуг з простим і зручним пошуком за вказаними критеріями;

– особистий кабінет зареєстрованих користувачів;

– кошик, куди будуть додані товари та послуги, обрані клієнтами;

– реєстрація та управління замовленнями, товарами та послугами адміністрацією веб-сервісу;

– особистий кабінет користувача з відображенням замовлень;
– можливість писати коментарі та відгуки про інтернет-магазин і товар;

Створення сайтів

З моменту створення всесвітньої мережі пройшло багато років. З часом багато що змінилося: зріс технічний потенціал Мережі, зросла інтернет-аудиторія, завдання та цілі, заради яких створювалася Всесвітня павутина. Інтернет перестав існувати лише в наукових і військових цілях. Сьогодні Інтернет – це великий склад, де кожен може знайти потрібний текстовий документ, музику та виконавця, відео чи наукову літературу, а також на деякий час придбати все необхідне – від продуктів харчування до програмного забезпечення та комп'ютерних компонентів. Компанії твердо і чітко вважають за необхідне розробити мінімальне інтернет-представництво організації, іншими словами, все більше і більше комерційних сайтів, основною метою яких є розвиток і продаж товарів і послуг через Інтернет.

Хоча вже можна сказати, що через деякий час конкурентоспроможність у роздрібній торгівлі через Інтернет буде такою ж, як і у звичайній роздрібній торгівлі офлайн. Цей процес можна наглядати на прикладі популярного в Україні інтернет магазину Розетка. С кожним днем, все більше і більше купують роздрібний товар в цьому магазині.

Існують різні види веб-сервісів:

– сервіс, що реалізує певний вид товарів і послуг;
– сервіс, що реалізує продукцію вузько спрямованих категорій;
– електронні гіпермаркети;
– великі роздрібні мережі, що складаються з веб-сервісів.

Для всіх веб-сервісів характерний певний набір компонентів, таких як:

– Загальний каталог з категоріями, які показують усі наявні товари. Зовнішній вигляд каталогів може бути різним – деревоподібні, спливаючі або вкладені списки категорій.

– процедура реєстрації користувача, яка створить для кожного нового клієнта його особистий «кошик», куди можна «додати» вибраний товар чи обрану послугу, а потім замовити. У міру переміщення покупця по каталогу система також буде відслідковувати побажання покупця, на основі чого згодом можна буде будувати не тільки колекції магазину, а й архітектуру каталогу супутніх товарів. Наприклад, якщо людина вказала книгу у своєму замовленні, система може непомітно рекомендувати переглянути інші книги з тієї ж серії чи автора, або запропонувати прочитати рецензії тих, хто її вже читав. Система може «повідомити», що особа, яка замовила подібний товар, також зацікавилася супутніми товарами або послугами.

– Система оплати товарів і послуг: клієнту пропонується використовувати різні способи оплати товарів і послуг - безготівковий розрахунок, кредитні картки, оплата електронними грошима, оплата готівкою після доставки (кур'єром або при отриманні товару).

– Система доставки товарів і послуг: також є широкий спектр опцій: відправка EMAIL (програмне забезпечення, ліцензійний ключ продукту), доставка кур'єрськими компаніями або іншою кур'єрською службою або звичайною поштою.

Однак, ігноруючи загальні особливості, інтернет-магазини все одно відрізняються один від одного. Власник будь-якого інтернет-магазину прагне зробити свій сайт максимально комфортним для гостя, вдосконалюючи способи переходу з одного каталогу в інший. Як і в простому магазині, в інтернет-магазині є можливість влаштовувати знижки та розпродажі. Основна відмінність звичайного

магазину від інтернет-магазину – це можливість витрачати менше грошей, а також можливість зробити покупку, не відволікаючись на важливі справи і головне, не виходячи з дому чи роботи. Завдяки цьому виходить, що покупка в інтернет-магазині стає кращою.

1.2. Характеристика можливостей «мобільної» комерції

Сайти та соціальні мережі дозволяють отримувати необхідну інформацію у будь-якому місці та у будь-який час. Якщо у вас є мобільний телефон, планшет або ноутбук з доступом в Інтернет, ви можете переглянути новини, які ви пропустили, по телевізору або прямому ефірі. Інтернет – це ресурс, який зберігає різноманітну інформацію. Зазвичай, в телепередачах висвітлюється лише один із можливих поглядів на ситуацію. Проте будь-яку інформацію можна знайти в Інтернеті. Поповнення джерел інформації переважно здійснюється постійними користувачами соціальних мереж, які активно діляться новинами свого села, міста та області. В інтернеті існує багато різних поглядів на різні ситуації.

На сайтах ті самі новини можуть бути представлені в різній формі. Тобто до будь-якої актуальної інформації можна прикріпити фото та відео, які ніколи не будуть показані на телеканалах. Таким чином, аудиторія стикається з різними точками зору, і кожна людина може вибрати ту, що їй ближче. Однак ця різноманітність думок часто призводить до дезінформації.

Далі популярності набули чати. Тоді їх почали використовувати для організації групового спілкування між декількома користувачами (зазвичай між двома та більше користувачами).

Наступним етапом було створення чат-ботів[20].

Бот - це свого роду помічник, призначений для автоматичної взаємодії з отриманими повідомленнями. Боти можуть бути запрограмовані щоразу по-різному реагувати на повідомлення, що містять певні ключові слова, і навіть використовувати машинне навчання, щоб адаптувати відповіді до ситуації.

Зараз чат-боти поділяються на два типи[20]:

– саморозвиваючі (що мають штучну нейронну мережу, що дозволяє їм самонавчати під час діалогу, управляти контекстом розмови), які використовують логіку при побудові діалогу або обробку природної мови та машинне навчання для формування відповідей на повідомлення або обидва.

– обмежені (що відповідає лише на задану кількість фраз з точною відповідністю запиту), при якому весь діалог є попередньо сформованим шаблоном, а «сценарій» — це дерево рішень, в якому відповідь на запитання відкриває нову, попередньо запрограмоване.

Боти використовують середовище чату, таке як текст SMS, вікна чату на веб-сайті та служби обміну повідомленнями в соціальних мережах на різних платформах, таких як Facebook (рис. 1.1), Twitter або Vkontakte, щоб отримувати відповіді на повідомлення. Чат-боти популярні і є інструментом для ефективного діалогу, який зазвичай замінює інші засоби зв'язку, такі як електронна пошта або телефонні дзвінки. Їх також можна використовувати як для надсилання інформації, так і для її збирання. Сьогодні месенджери мають велику популярність, це пов'язано зі зміною сфери мобільного інтернету: високі швидкості, низька ціна та широке використання смартфонів, сценарій. Діалоги в них зазвичай лінійні та структуровані.

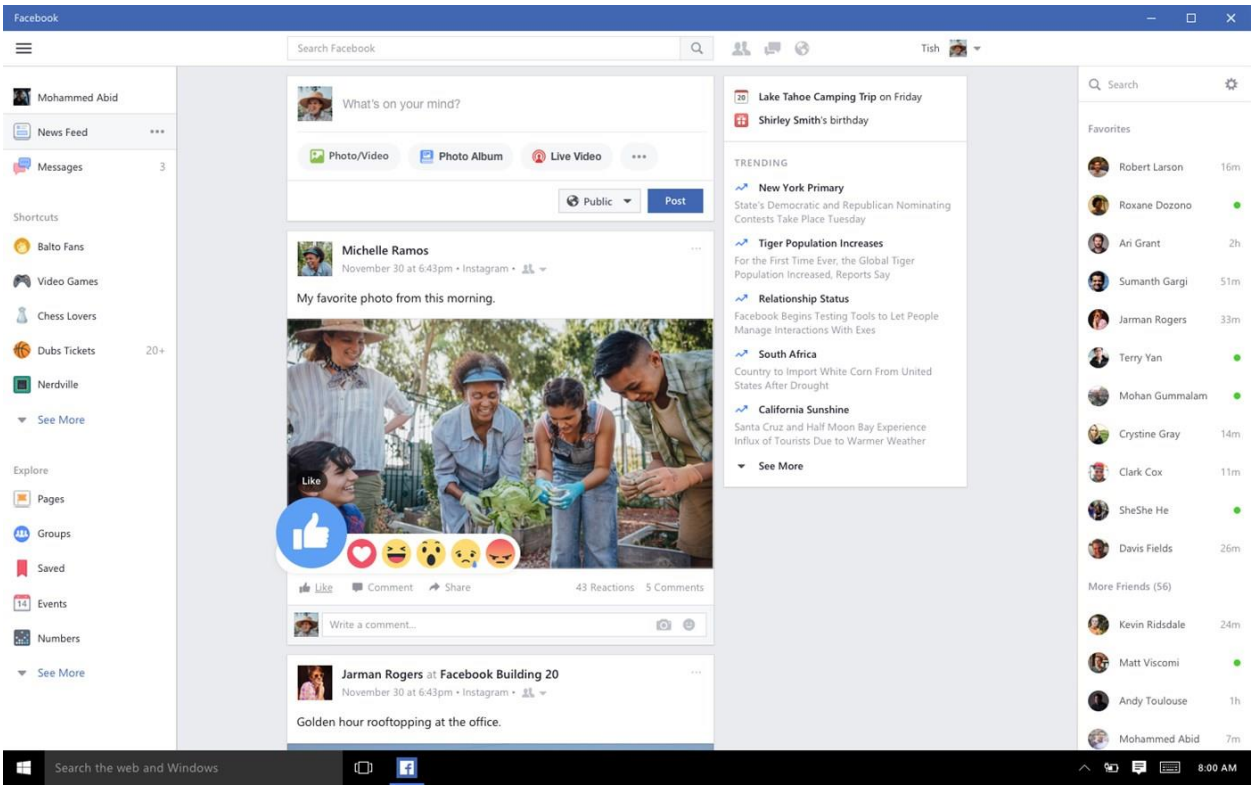

Рисунок 1.1 – Інтерфейс найбільшої соціальної мережі Facebook

Першим з таких чатів можна вважати Talkomatic. Він був розроблений у 1973 р. Д. Брауном та Д. Р. Уоллі в системі PLATO (Programmed Logic for Automated Teaching Operations) [2], яка на той час вважалась першою системою для електронного навчання в світі та була призначена для автоматизації навчального процесу та обміну даними Університету Іллінойсу. Інтерфейс Talkomatic виглядав наступним чином (рис. 1.2).

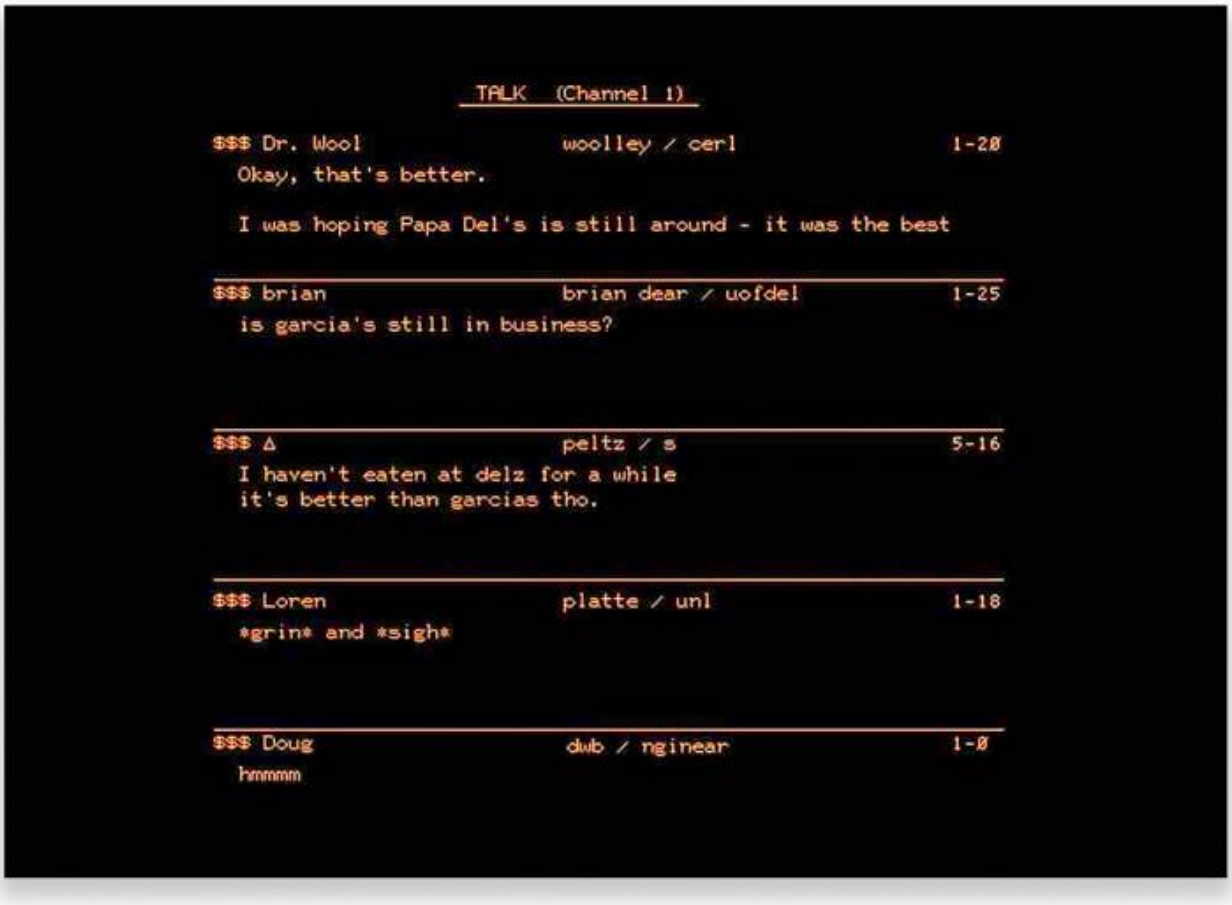

Рисунок 1.2 – Інтерфейс Talkomatic

У даній системі можна було отримати доступ до шести різних каналів, кожен з яких мав змогу вмістити до п'яти користувачів одночасно [3]. Звісно, з того часу чати еволюціонували, й тепер практично кожний сервіс, що надає послуги обміну повідомленнями та даними онлайн, немає жорстких обмежень щодо кількості користувачів та каналів, у яких бере участь користувач, як це було на початку 1970-х років. Також сучасні чати все частіше використовують новітні технології, такі як нейронні мережі, штучний інтелект, або інші.

У будь-якому випадку, усі ці нововведення були направлені на покращення взаємодії безпосередньо з користувачем. Паралельно з розробкою та освоєнням

користувачами перших чатів для швидкого обміну текстовими повідомленнями та іншими даними, юзери починали переходити на взаємодію з іншими користувачами мережі Інтернет за допомогою перших так званих месенджерів. Примітивні застосунки для передачі даних побачили світ майже 50 років тому.

Перші системи, що базувались на використанні графічного інтерфейсу користувача для відображення отриманого повідомлення, набули популярності у 1990-х роках. До перших таких месенджерів належали ICQ (рис. 1.3) та AOL Instant Messenger. У той же час інші компанії розробили власне програмне забезпечення для обміну миттєвими повідомленнями, а саме Ubique, MSN та Yahoo!.

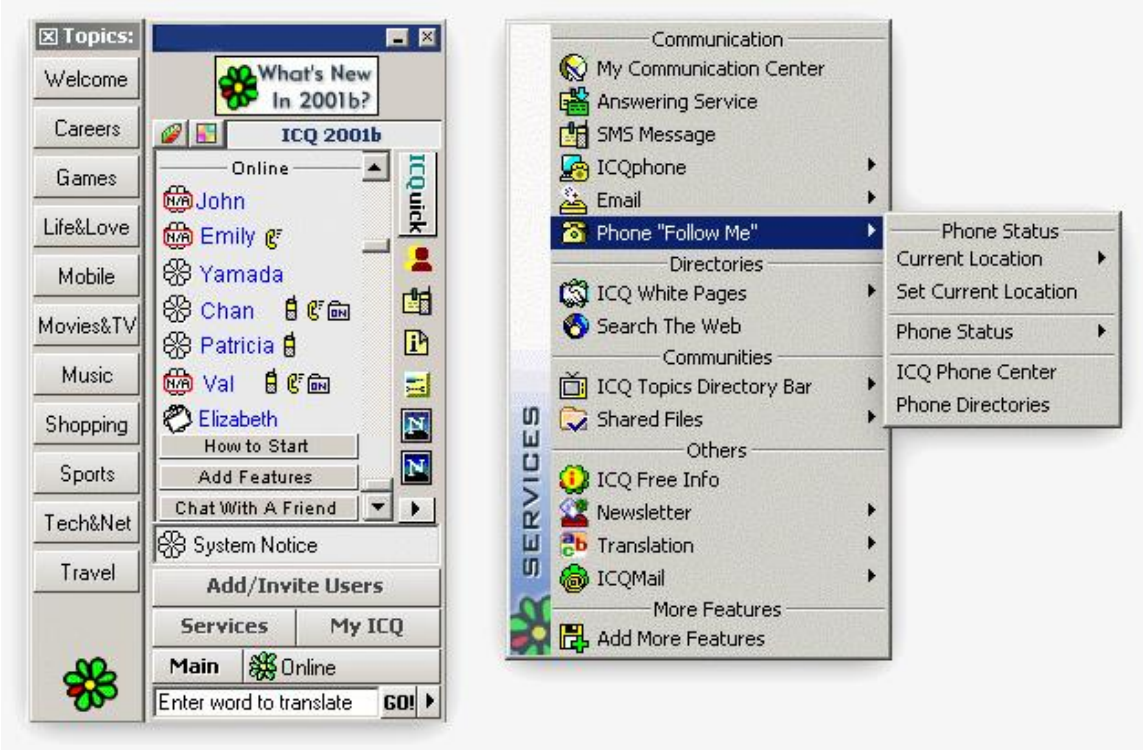

Рисунок 1.3 – Інтерфейс відомого месенджера ICQ

Кожен з таких застосунків, що на даний час здаються примітивними, мав свій власний протокол та клієнт, тому для того, щоб використовувати одразу декілька

месенджерів, користувачам необхідно було запускати відразу декілька додатків для коректної роботи. На початку 2000-х років було запущено у розробку ПЗ з відкритим програмним кодом та відкритий протокол Jabber, який успішно пройшов стандартизацію під звичною назвою XMPP (або Extensible Messaging and Presence Protocol). Сервери такого типу можуть слугувати в якості шлюзів для інших протоколів, що не потребує запуску відразу декількох клієнтів для роботи.

AOL Instant Messenger є попередником сучасних соціальних мереж та модерних засобів обміну текстовими повідомленнями та даними. У персональному профілі користувачі могли заповнити деякі відомості про себе. Профілі у мережі були доступні для пошуку, щоб інші користувачі могли переглядати інформацію. Це був один з найбільш інноваційних інструментів того часу [4]. Інтерфейс месенджеру зображено на рис. 1.4.

Рисунок 1.4 – Інтерфейс месенджера AOL Instant Messenger

1.3. Аналіз досвіду використання платіжних чат-ботів для е-комерції

Аналізуючи сучасний стан використання чат-ботів в месенджерах, можна прийти до висновку, що чат-боти є універсальними засобами, здатними до

вирішення різноманітних завдань – від спілкування до розваг, від надання медичної консультації до замовлення товарів та послуг за допомогою спеціалізованих прикладних рішень, від розпізнавання емоцій до вирішення складних консалтингових завдань в службах підтримки клієнтоорієнтованих інформаційних систем, а також забезпечення розпізнавання тексту, якщо це необхідно [13].

У будь-якому випадку, в незалежності від платформи, чат-бот — це прикладна програма, яка, отримуючи інформацію від користувача, формує коректні, логічно обґрунтовані відповіді. На сьогоднішній момент існує величезна різноманітність чат-ботів [13]. Деякі з можливих варіантів представлені нижче:

– ігрові чат-боти (квести або рольові ігри);

– рекламні чат-боти;

– новинні чат-боти;

– чат-боти для доставок, магазинів та сервісів (замовлення таксі, їжі, бронювання квитків, тощо);

– чат-боти для консультації та підтримки клієнтів;

– навчальні чат-боти.

Останнім часом все більші можливості відкривають інформаційні технології, пов'язані з чат-ботами. Вони настільки міцно увійшли в життя людей, що застосовуються в усіх сферах діяльності людини, і з кожним днем їх роль все більше зростає. В першу чергу, це пояснюється тим, що основну частину свого часу люди проводять за смартфоном в email-клієнтах та месенджерах [13].

В епоху інформаційних технологій – це доволі нормальне явище, а тим більше у мережі Інтернет, оскільки суспільство давно перейшло на новий рівень ділового та повсякденного спілкування. По-перше, чат-боти – це «платформи» для вирішення певних рутинних завдань, пошуку інформації, об'єднання даних, що допомагають у взаємодії з користувачами. По-друге, чат-бот – це програма, яка

підтримує діалог з користувачем, вибираючи відповіді з бази даних: ви формуєте певний запит і відразу отримуєте миттєву відповідь згідно зі сформованого запиту.

Чат-бот як віртуальний співрозмовник має базу знань, яка представляє собою набори можливих питань користувача та відповідних їм відповідей. Найбільш поширеними варіантами для отримання потрібної відповіді є ключові слова, збіг фрази, збіг контексту або інші вказані параметри. У сучасному суспільстві для збору та відображення інформації можна використовувати чат-ботів. Звичайно, це робиться за допомогою спілкування з людьми. Наприклад, в рамках якого-небудь заходу чат-боти можуть повідомляти всім учасникам новини та надавати довідкову інформацію [14].

Розглянемо декілька конструкторів, що дозволяють створювати ботів під різні функції.

InfoBot – з цим конструктором можна створити бота для відправки повідомлень, вміє відображати клавіатуру, можна переглядати діалог з користувачем та робити розсилки.

Під час тестування даної платформи створювати команди довелося з використанням інтуїції, оскільки відсутні підказки та плейсхолдери. Не весь функціонал даної платформи зручний для користувача. Також слід зазначити, що зовнішній вигляд та можливості конструктора доволі примітивні (рис 1.5). Доступ до платформи з мобільних пристроїв також не реалізовано.

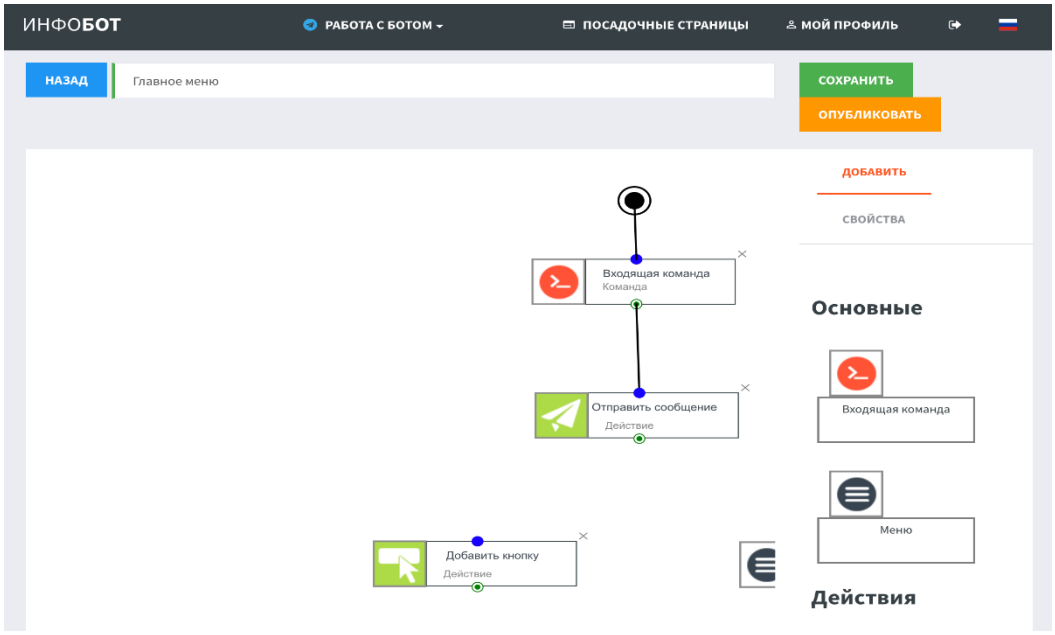

Рисунок 1.5 – Інтерфейс платформи InfoBot

Ebot one – дана платформа для створення та редагування Telegram-ботів дозволяє без навичок програмування сформувати структуру бота та реалізувати її. Якщо певні навички програмування все ж є – логіку роботи можна зробити набагато складніше, а бота цікавішим й кориснішим для використання [16].

Слід також зазначити, що доступ до платформи з мобільних пристроїв не реалізовано. До того ж, платформа (рис. 1.6) має доволі розвинений функціонал. Останнє оновлення сервісу було ще у 2018 році.

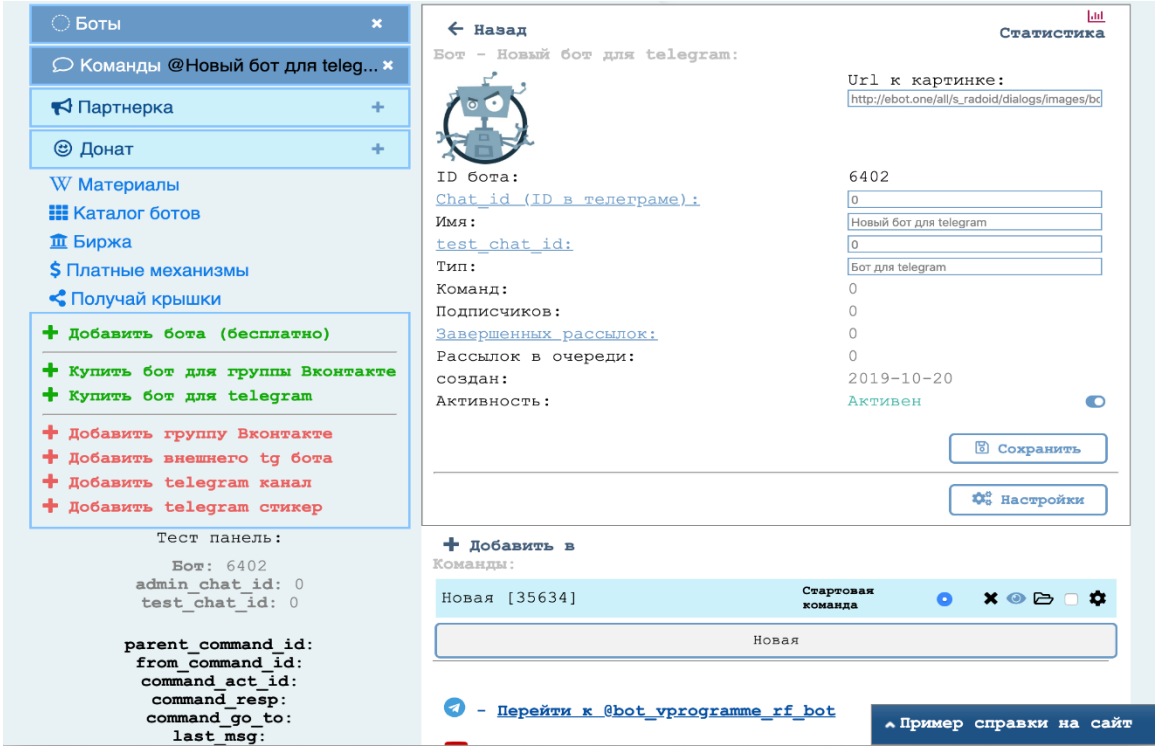

Рисунок 1.6 – Інтерфейс платформи Ebot one

Botkits – простий сервіс (рис. 1.7) для створення простих ботів, які можуть надсилати різного роду інформацію (текст, зображення або документи) з можливістю прикріплювання клавіатури[17].

Щодо наявного функціоналу, це доволі простий сервіс з обмеженими функціональними можливостями для створення примітивних ботів. Більшість користувачів також відзначає, що дана платформа досить довго обробляє запити та завантажує дані, часто з'являються помилки з повідомленням, наприклад: «Fatal error: Call to a member function findFile() on a non-object in /home/ebot/ebot-one/vendor/composer/ClassLoader.php on line 427», що сервер на даний час не відповідає, тому в повній мірі протестувати цей сервіс не вдалось.

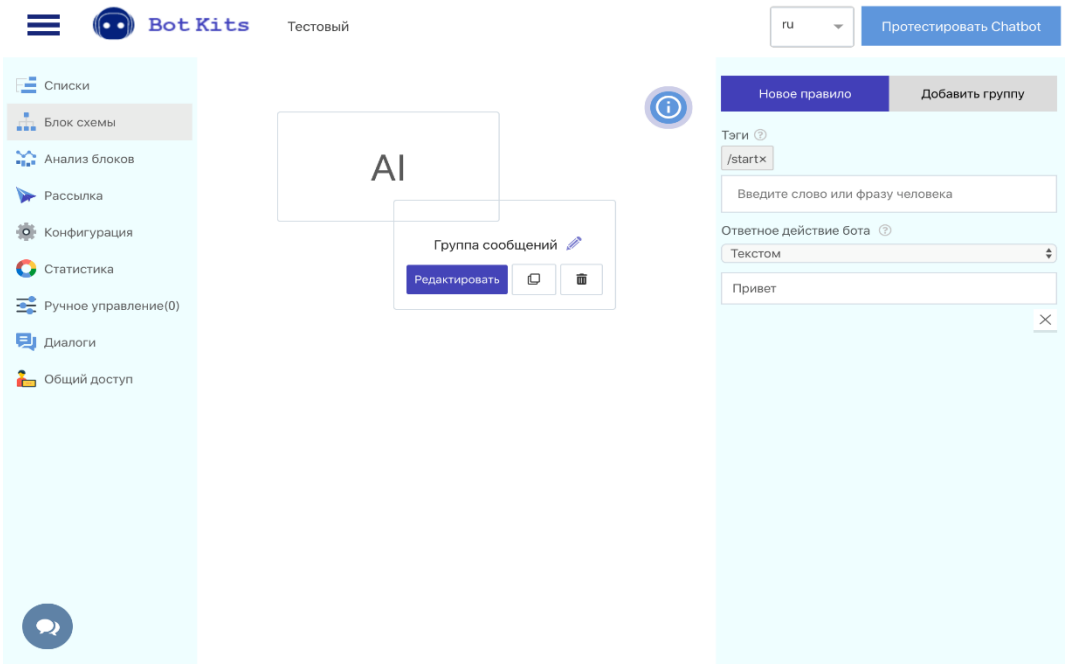

Рисунок 1.7 – Інтерфейс платформи Botkits

Manybot – особистий кабінет сервісу представлений у вигляді бота в Telegram. Конструктор дозволяє створювати меню, підменю, форму зворотного зв'язку, підключати RSS-стрічки та робити розсилки по всім користувачам, що є підписниками.

Позитивним моментом є створена мобільна версія, що також представлена у вигляді сервісу, що працює через месенджер Telegram [18]. Дана платформа підходить для невеликих проектів, але створювати меню більше чотирьох рівнів через інтерфейс бота стає складно. Крім цього, бот довго відповідає або інколи не відповідає зовсім на деякі запити користувача. Слід також додати, що сервіс не оновлюється з 2015 року. Інтерфейс бота наведено на рис. 1.8.

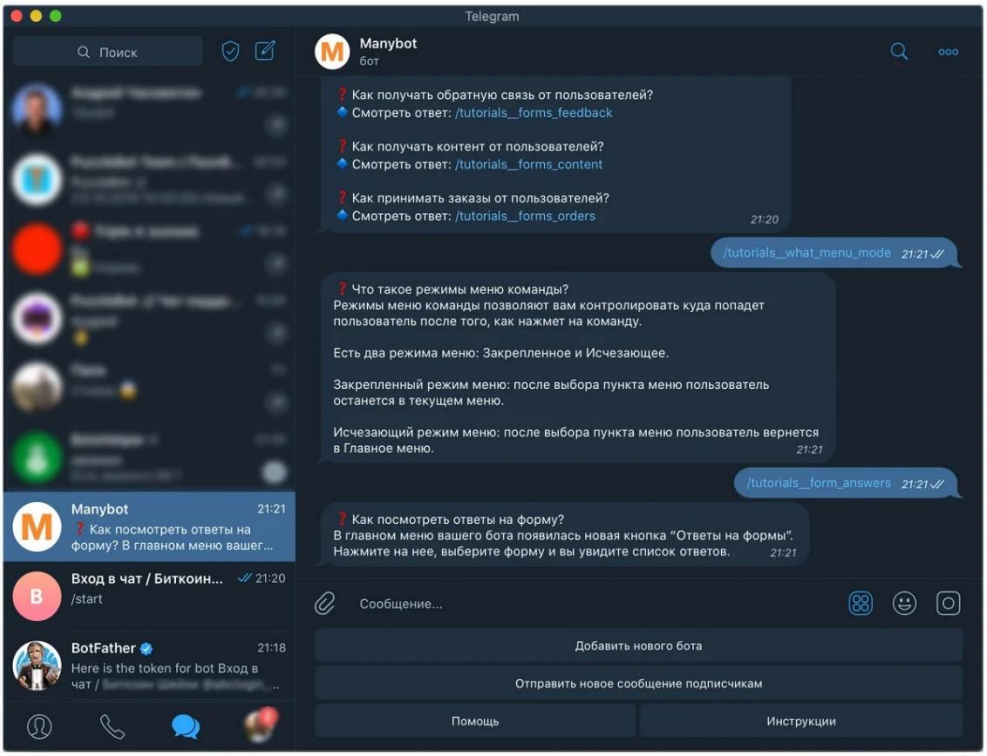

Рисунок 1.8 – Інтерфейс платформи Manybot

WeChat — це Китайськи аналог чат-боту. Крім функцій соціальних мереж, ви можете замовляти їжу, викликати таксі, бронювати готелі та читати новини [19]. Коли мова йде про фінанси — отримання виписок з банківського рахунку, оплата комунальних послуг, надсилання грошей — одним словом, увесь Інтернет в одному місці.

WeChat використовується в Китаї, Гонконгу, Тайвані, Малайзії, Таїланді, Південній Африці та США. Функція повністю доступна лише в Китаї, де 889 мільйонів користувачів щомісяця використовують додаток. Серед його шанувальників є високопоставлені члени Комуністичної партії Китаю.

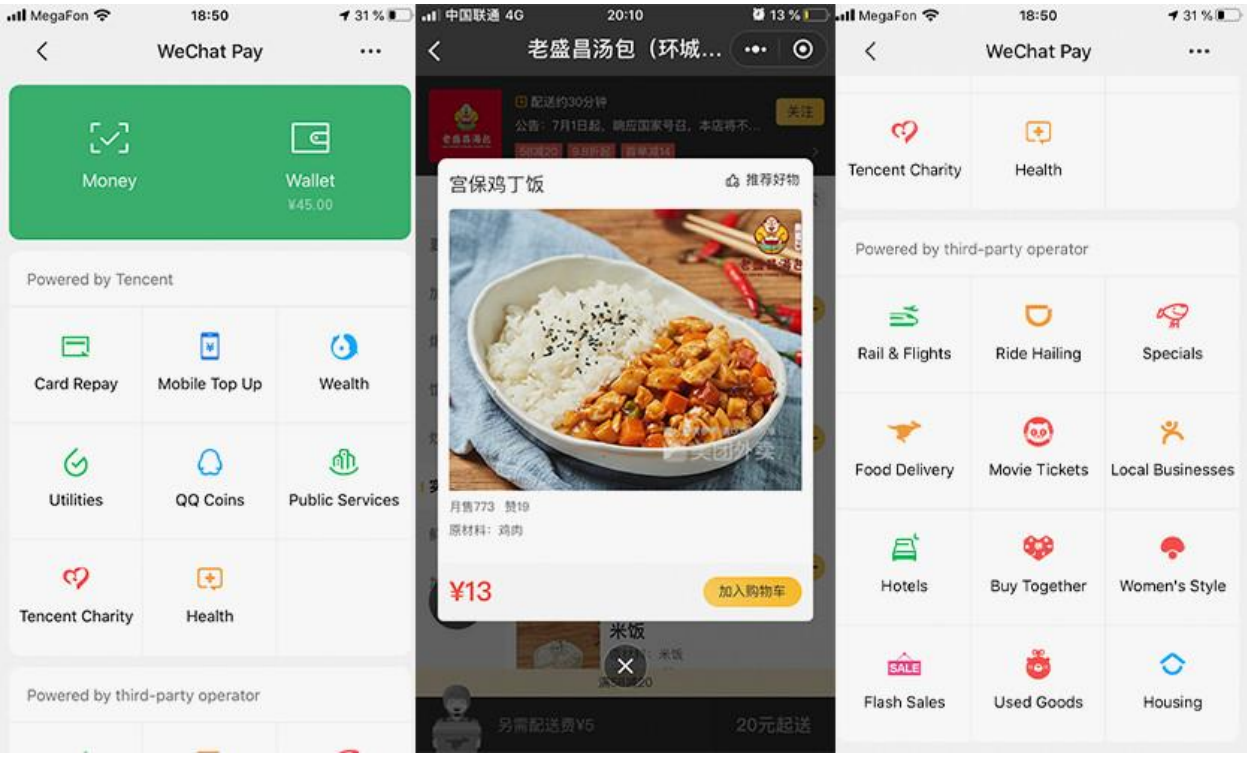

Рисунок 1.9 – Інтерфейс платформи WeChat

Залучення аудиторії та глибина платформи вражають: 90% користувачів користуються додатком щодня, а половина з них витрачає на нього більше години — як і на всі додатки Facebook разом узяті. 61% користувачів відкривають програми більше 10 разів на день, а 36% більше 30 разів.

1.4. Функціональні вимоги

У сучасному світі значного розвитку набувають технології швидкісного обміну інформацією та повідомленнями між користувачами за допомогою мережі Інтернет. Ця тенденція актуальна не тільки для особистого користування, а й активно використовується у процесі підтримки роботи по всьому світі.

Доцільність та важливість розроблюваного програмного продукту полягає в наступних аспектах:

1. Зручний, інтуїтивно зрозумілий інтерфейс. легкий у використанні та вивченні графічний інтерфейс є одним із надважливих пунктів для проекту, адже він дозволить використовувати Telegram-бот всім користувачам, незалежно від рівня їх компетенцій.

2. Незалежність від платформи. Telegram-бот може бути запущений на будь-якій платформі з мінімальними характеристиками, незалежно від використовуваної операційної системи та встановленого програмного забезпечення.

3. Необмежена кількість користувачів. Платформа Telegram дозволяє створювати сутності, до яких може приєднатися до 200 тис. незалежних користувачів, що в реальних умовах, є фактично безлімітною кількістю унікальних користувачів [18].

В розроблюваній системі EcommerceTelegram заплановано перелічені нижче функціональні можливості, які будуть делегуватися на два рівні доступу користувачів – BUYER та SELLER. BUYER – це тип користувача, який має право:

- переглядати лист усіх товарів магазину;
- фільтрувати товари за текстом назви чи опису товару;
- додавати до корзини товари;
- оформити замовлення.

SELLER адміністратори «створюються» вручну через базу даних (у першій версії боту).

За замовчуванням SELLER має ті ж самі права, що і BUYER а також додатковий функціонал:

- додавання, редагування та видалення товарів

– повідомлення підписників (можливість створення повідомлень для підписників чат-бота);

– додавання нових команд користувача (розширення функціоналу бота за допомогою додавання нових команд користувача);

– перегляд користувачів бота.

У створених товарів є власні атрибути, такі як:

– name;

– description

– price

– picture

Для успішного використання розроблюваного рішення також необхідна наявність пристроїв, які будуть відповідати мінімальним вимогам для створюваної платформи для розробки телеграм-ботів для електронної комерції. З переліком мінімальних вимог для використання месенджера Telegram, як основи, на базі якої й розроблюється програмний продукт, можна ознайомитись при встановленні застосунку Telegram у офіційних джерелах.

1.5. Постановка задачі дослідження

Практичне значення даного випускного проекту – розробка платформи електронної комерції, який після встановлення та розміщення, а також наповнення товарами та послугами почне повноцінно виконувати свої цілі, функції та завдання.

Ці функції включають:

1) можливість створювати, переглядати та редагувати категорії товарів і послуг;

2) можливість редагування, обслуговування замовлення;

3) право вибору видів і способів оплати товарів і послуг;

4) право вибору видів і способів доставки;

5) наявність панелі керування адміністратора веб-сервісу.

Основними способами виконання завдання є вивчення та аналіз подібних проектів за темою, вивчення та розвиток інтернет-магазину з його подальшою програмною реалізацією.

РОЗДІЛ 2. ХАРАКТЕРИСТИКА ТА АНАЛІЗ ЕКСПЕРИМЕНТАЛЬНОЇ ТЕХНОЛОГІЇ Е-КОМЕРЦІЇ

2.1. Зміст завдання та технологічні особливості створення чат-ботів

Метою дипломного проекту є розробка зручного та простого веб-сервісу (телеграм-бота) з продажу товарів для збільшення прибутку компанії, розширення клієнтської бази та меж бізнесу.

Основна мета програми – наблизити покупця до інтернет-магазину, до товару. Ми стали частіше користуватись месенджерами відколи вони з’явилися на мобільних пристроях. Це помітили автори/розробники Telegram/Viber/… та стали додавати нові функції до них (як-от здійснення оплати за товари та послуги, перегляд новин та ін.). Така система, з високим рівнем впровадження, повинна полегшити роботу працівників магазину.

Система повинна передбачати поділ прав доступу і бути реалізована для двох категорій користувачів: BUYER та SELLER.

Частина системи, призначена для BUYER, у свою чергу повинна забезпечувати наступні можливості:

- здійснювати пошук товарів за вибраними фільтрами;
- переглянути інформацію про товар;
- переглянути список коментарів до товару;
- зручний, інтуїтивно зрозумілий інтерфейс з підказками;
- додати товари в кошик.
- аутентифікація користувача в системі;
- перегляд списку товарів у кошику та видалення товарів із кошика, зміна кількості товарів у кошику;
- оформлення замовлення на покупку товару в кошику;

– перегляд історії замовлень, зроблених користувачем;

Система для SELLER магазину повинна забезпечувати наступні можливості:

– можливість додавати, видаляти, змінювати інформацію про категорії;
– можливість додавати, видаляти, змінювати інформацію про товари;
– отримання текстової та графічної інформації про історію покупок;
– всі можливості, надані покупцям товарів.

З метою полегшення підтримки та обслуговування програми, а також її подальшого розширення, вона повинна бути розроблена та створена на основі трирівневої архітектури (клієнт, сервер додатків і база даних) для побудови програмних систем та мати клієнтську та серверну частину. Серверна частина повинна відповідати за доступ до даних і містити бізнес-логіку програми. Клієнтська частина, у свою чергу, повинна реалізувати користувальницький інтерфейс програми.

Задля спрощення і підвищення розробки MVP було обрано використовувати СКБД як рівень доступу до даних, який забезпечує централізоване структуроване зберігання всіх системних даних, гарантуючи їх цілісність і узгодженість, а також надає безліч низькорівневих сервісів для: читання даних зі сховища, збереження даних, зміни їх структура і т.д. На цьому рівні має бути створена база даних, яка буде зберігати всі дані системи. Реалізація команд отримання даних, контроль цілісності та узгодженості даних повинні здійснюватися за допомогою відповідних збережених процедур, тригерів та інших об'єктів, наданих сервером.

Рівень бізнес-логіки буде розгорнутий на сервері додатків і представлятиме ядро системи. Більша частина бізнес-логіки системи повинна бути зосереджена на цьому рівні:

– алгоритми авторизації користувачів системи, перевірки прав доступу;

– правила обробки даних, такі як: перевірка правильності заповнення даних користувачем, перевірка та організація взаємозв'язків даних;

– клас для підключення до бази даних і виконання транзакцій;

– класи та алгоритми роботи з таблицями БД та запуску виконання відповідних збережених процедур і функцій на сервері.

На рівень презентації необхідно перенести найпростішу бізнес-логіку: інтерфейс для відображення товарів, інтерфейс для відображення кошика користувача, інтерфейс для відображення профілю користувача, різноманітні операції з товарами.

2.2. Вибір платформи мови програмування для створення платформи електронної комерції у Телеграм

Технологічний стек, який використовується при розробці платформи електронної комерції складається з наступного переліку:

Java

Вся логіка системного процесу написана на мові високого рівня Java, яка є об'єктно-орієнтованою мовою програмування. Вибір був зроблений тому, що платформа Java має велику кількість технологій і фреймворків, створених для вирішення широкого кола прикладних задач. Ще однією перевагою Java є віртуальна машина, яка забезпечує кросплатформенну сумісність. Основним підходом Java вважається «напиши один раз, використовуй будь-де».

На додаток до крос-платформної сумісності, перевага мови полягає в її високій надійності, оскільки вона була розроблена як строго типізована об’єктно-орієнтована мова високого рівня.

Рисунок. 2.1 – Логотип мови програмування Java

Java Development Kit (JDK) — це комплект для розробки Java. Використовуючи JDK і стандартний блокнот, ви можете писати та запускати/компілювати код Java.

Java Runtime Environment (JRE) — це система виконання Java. Механізм розповсюдження програмного забезпечення складається з автономної віртуальної машини Java, стандартної бібліотеки класів Java (Java Class Library) і засобів налаштування.

Spring Boot

Spring Boot — це надбудова над фреймворком Spring та автоконфігурації на найбільш поширені use-case у Spring. Основна перевага його використання полягає в тому, що він сам керує версіями необхідних бібліотек, тому немає проблем у разі невідповідності версій. Також містить сервер додатків Tomcat.

Рисунок 2.2 – логотип фреймворку Spring

Thymeleaf

Thymleaf — це сучасний механізм шаблонів Java на стороні сервера для веб- та офлайн-середовищ, здатний обробляти HTML.

Основна мета Thymeleaf — створити елегантний і зручний спосіб створення шаблонів. Щоб досягти цього, Thymeleaf спирається на концепцію природних шаблонів, впроваджуючи свою логіку у файли шаблонів таким чином, щоб шаблон не впливав на рендеринг дизайну прототипу. Це покращує комунікацію в команді та зменшує розрив між командами дизайнерів і програмістів.

Рисунок 2.3 – логотип Thymeleaf

Thymeleaf також розроблено з урахуванням веб-стандартів, особливо HTML5, що дозволяє створювати повністю сумісні шаблони

Hibernate

Hibernate — це структура для зіставлення таблиць у базі даних зі звичайними об’єктами Java. Це спрощує процес розробки, оскільки маніпулювання операціями CRUD стає легшим.

Рисунок 2.4 – логотип Hibernate

Hibernate є одним із найпопулярніших фреймворків Java ORM.

Переваги Hibernate:

– Hibernate усуває багато повторюваного коду, приховує багато коду, необхідного для управління ресурсами від розробників, і дозволяє зосередитися на бізнес-логіці;

– Hibernate підтримує анотації XML і JPA, що дозволяє зробити реалізацію коду незалежною;

– Hibernate надає власну потужну мову запитів (HQL), подібну до SQL. Варто зазначити, що HQL повністю об'єктно-орієнтований і розуміє такі принципи, як успадкування, поліморфізм і асоціація (з'єднання);

– Hibernate є широко використовуваним проектом з відкритим кодом. Завдяки цьому доступні тисячі публічних статей, прикладів і документації щодо використання фреймворку;

– Hibernate легко інтегрується з іншими фреймворками Java EE, наприклад, Spring Framework підтримує вбудовану інтеграцію з Hibernate;

– Hibernate підтримує відкладену ініціалізацію за допомогою проксі-об'єктів і виконує запити до бази даних лише за необхідності;

– Hibernate підтримує різні рівні кешування, таким чином покращуючи продуктивність;

– Важливо, що Hibernate може використовувати чистий SQL, а це означає, що він підтримує оптимізовані запити та можливість працювати з будь-яким стороннім постачальником бази даних.

JavaScript

Розглянемо основні бібліотеки та фреймворки JavaScript, які використовуються в сучасному WEB-програмуванні. У роботі [10] порівнюється їх

продуктивність (згідно з рисунком 2.5) у браузері Google Chrome 48. У цьому випадку ми тестуємо створення 1000 рядків відразу після завантаження сторінки («створити 1000 рядків»), оновлюючи 1000 рядків у таблиці після 5 ітерацій «розігрів» движка JavaScript («оновити 1000 рядків (гаряче)») часткове оновлення рядків у таблиці після 5 ітерацій «розігріву» движка JavaScript (додавання крапки в кінець кожного 10-го рядка, «часткове оновлення»), виділення рядка після 5 ітерацій «прогрівання» движка JavaScript (візуальне виділення рядка, «вибрати рядок»), видалення рядка після 5 ітерацій «розігріву» движка JavaScript («видалення рядка»). Як видно з гістограми на рис. 2.5, бібліотека VueJS працює найкраще, за винятком оновлення рядків у таблиці. WEB-інтерфейс передбачає візуальне оновлення даних в режимі реального часу, а створення візуальної «підкладки» для відображення тестових запитань створюється лише один раз, тому потрібно вибрати бібліотеку, продуктивну при оновленні даних і йде добре з реалізацією маршрутизації клієнтів.

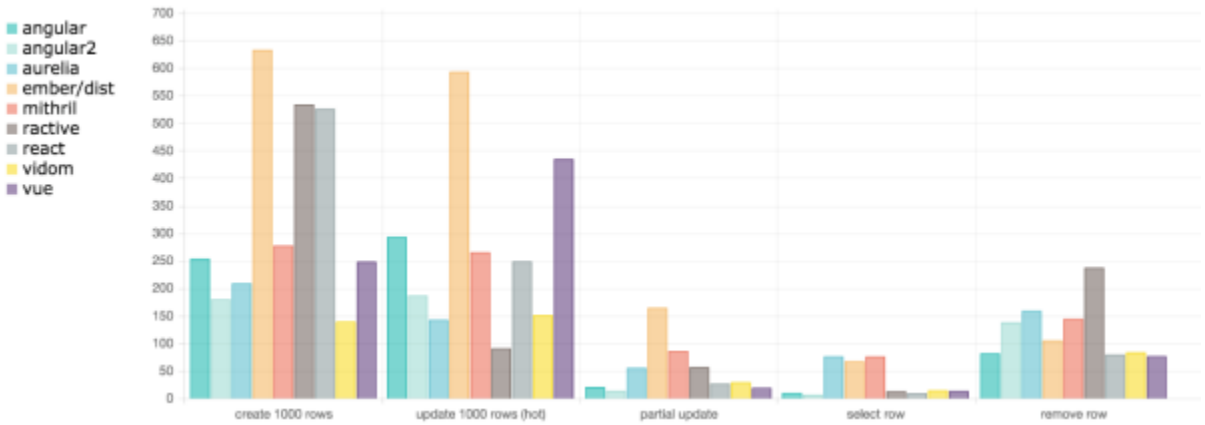

Рис. 2.5 -Гістограма продуктивності фреймворку JS (мс)

Оскільки більшість браузерів, які використовуються сьогодні, не підтримують сучасні стандарти EcmaScript [7], браузери повинні використовувати старий стандарт EcmaScript 5, який підтримується всіма сучасними браузерами, включаючи Internet Explorer 11 [8], що дозволяє значно розширити можливості за допомогою розробленого WEB-інтерфейсу. Однак, відмовляючись від новітніх стандартів мови програмування, розробник також відмовляється від можливості використання нового синтаксичного цукру та деяких оптимізаційних рішень нових стандартів. Для цього необхідно перевести JavaScript новішого стандарту [7] на JavaScript стандарту EcmaScript 5, з цим завданням справляється Babel (компілятор JavaScript, його документація є на сайті [9]).

В результаті виникає досить багато залежностей, які потрібно імпортувати в більшість файлів, і в міру розвитку програми код зростає і з'являється багато файлів. Щоб зібрати файли JavaScript і файли стилів в один файл JavaScript і файл стилів відповідно, вам знадобиться система для побудови коду і пакетів у WEB-інтерфейсі, webpack, документація якого знаходиться на сайті [10].

MySQL

База даних — це структурований набір даних. Ці дані можуть бути чим завгодно: від простого списку майбутніх покупок до списку експонатів у художній галереї чи великої кількості інформації в мережі компанії. Для запису, отримання та обробки даних, що зберігаються в комп'ютерних базах даних, потрібна система керування базами даних або програмне забезпечення MySQL. Оскільки комп'ютери добре обробляють великі обсяги даних, керування базами даних відіграє центральну роль в обчисленнях. Це керування можна реалізувати різними способами — або у вигляді окремої утиліти, або у вигляді коду, включеного в інші програми.

Рис. 2.6 – логотип MySQL

Таблиці пов’язані між собою за допомогою зв’язків, що дає змогу об’єднувати дані з кількох таблиць під час виконання запиту. SQL, який є частиною системи MySQL, можна описати як структуровану мову запитів і найпоширенішу стандартну мову для доступу до баз даних. Це програмне забезпечення з відкритим кодом. Будь-хто може використовувати та змінювати його. Таке програмне забезпечення доступне в Інтернеті та використовується безкоштовно. При цьому кожен користувач може вивчити вихідний код і внести зміни відповідно до своїх потреб.

MySQL — це клієнт-серверна система, яка включає багатопоточний сервер SQL, який забезпечує підтримку різноманітних машин баз даних, кілька різних клієнтських програм і бібліотек, інструменти адміністрування та розширений інтерфейс програмування (API).

Docker

Docker — це набір продуктів платформи як послуги (PaaS), які використовують віртуалізацію на рівні операційної системи для доставки програмного забезпечення в пакетах, які називаються контейнерами. Контейнери ізольовані один від одного, мають власне програмне забезпечення, бібліотеки та

конфігураційні файли; вони можуть спілкуватися один з одним через чітко визначені канали, які називаються портами.

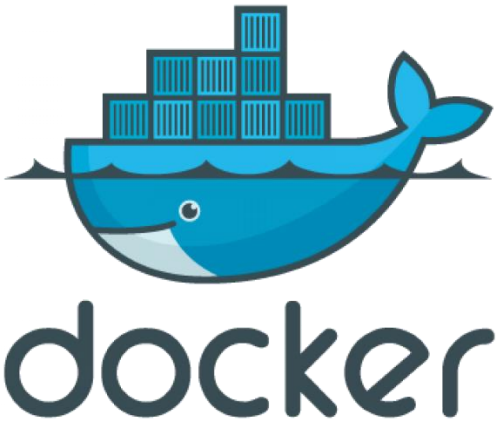

Рисунок 2.7 - логотип Docker

Оскільки всі контейнери використовують служби з одного ядра операційної системи, вони використовують набагато менше ресурсів, ніж віртуальні машини [7].

Docker може запакувати програму та її залежності у віртуальний контейнер, який можна запускати на будь-якому комп'ютері Linux, Windows або macOS. Це дозволяє програмам запускатися в різних місцях, наприклад локально, у відкритій хмарі та/або в приватній хмарі. Під час роботи в Linux Docker використовує функції ізоляції ресурсів ядра Linux (такі як контрольні групи та простори імен ядра) і об'єднані файлові системи (такі як OverlayFS), щоб дозволити контейнерам працювати в одному екземплярі Linux, уникаючи накладних витрат на роботу та підтримку віртуальних машин. У macOS Docker використовує віртуальні машини Linux для запуску контейнерів [17].

Gitlab + Gitlab CI

GitLab — це сайт і система керування сховищами програмного коду Git із додатковими функціями: власною вікі-сторіною та системою відстеження помилок [8].

GitLab – надає користувачам послуги, подібні до GitHub, із додатковими перевагами, такими як приватні сховища для безкоштовних передплатників. Ще однією істотною перевагою є можливість розгортання системи на сторонніх серверах.

Програмне забезпечення доступне в системі керування пакетами Omnibus.

Понад 100 000 організацій використовують систему, включаючи NASA, CERN і Alibaba.

CI/CD (Continuous Integration and Delivery) — це техніка для автоматизації тестування та доставки нових модулів уже розробленого проекту.

Безперервна інтеграція (CI) і безперервна доставка (CD) втілюють культуру, набір принципів і набір практик, які дозволяють командам розробників додатків вносити зміни в код частіше і надійніше.

Безперервна інтеграція - це кодування та набір практик, які заохочують команди розробників до внесення невеликих змін і частого розміщення коду в сховищах контролю версій. Оскільки для більшості сучасних програм потрібна розробка коду на різних платформах і інструментах, командам потрібен механізм для інтеграції та перевірки внесених змін.

Технічною метою безперервної інтеграції є створення узгодженого та автоматизованого способу створення, упаковки та тестування програм. Завдяки узгодженості процесу інтеграції команди частіше змінюють код, що забезпечує кращу співпрацю та якість програмного забезпечення.

Безперервна доставка починається там, де безперервна інтеграція припинилася. CD автоматично доставляє програми до вибраних інфраструктурних середовищ. Більшість команд використовують кілька середовищ на додаток до виробництва, таких як розробка та тестування, а CD забезпечує автоматичний спосіб надсилати їм зміни коду.

Інструменти CI/CD допомагають зберігати специфічні параметри середовища, які повинні бути упаковані з кожною поставкою. Потім автоматизація CI/CD здійснює будь-які необхідні виклики служб до веб-серверів, баз даних та інших служб, які, можливо, потрібно буде перезапустити або виконати інші процедури під час розгортання програми.

Безперервна інтеграція та безперервне постачання вимагають постійного тестування, оскільки метою є надання високоякісних програм і коду користувачам.

Безперервне тестування часто реалізується як набір автоматизованих тестів регресії, продуктивності та інших тестів, що виконуються в рамках завдання CI/CD.

CI/CD також є однією з найкращих практик для впровадження команд devops.

Це також найкраща практика гнучкої методології, оскільки вона дозволяє групі розробників програмного забезпечення зосередитися на дотриманні бізнес-вимог, якості коду та безпеці, оскільки етапи розгортання автоматизовані.

2.3 Telegram Bot API

Bot API представляє собою HTTP-інтерфейс для роботи з ботами в Telegram. Кожен бот – це спеціальний аккаунт, створений для оброблення та відправлення повідомлень. Існує два протилежних за логікою способу отримання оновлень від бота:

– long pulling – додаток автоматично опитує сервера Telegram на наявність будь-яких оновлень для бота, де час затримки між запитами складає за замовчуванням 100 мс;
– webhook – сервера Telegram самі сповіщають додаток на сервері, як тільки з'являться будь-які оновлення.

Вхідні оновлення будуть зберігатися на сервері до тих пір, поки їх не оброблять, але не довше 24 годин. Незалежно від способу отримання оновлень, у відповідь відправляється об'єкт Update, серіалізований в JSON [26].

Всі запити до Telegram Bot API повинні здійснюватися через HTTPS в наступному вигляді: https://api.telegram.org/bot<token>/НАЗВА_МЕТОДУ.

Принцип роботи взаємодії чат-бота і користувача зображений на рис 2.8.

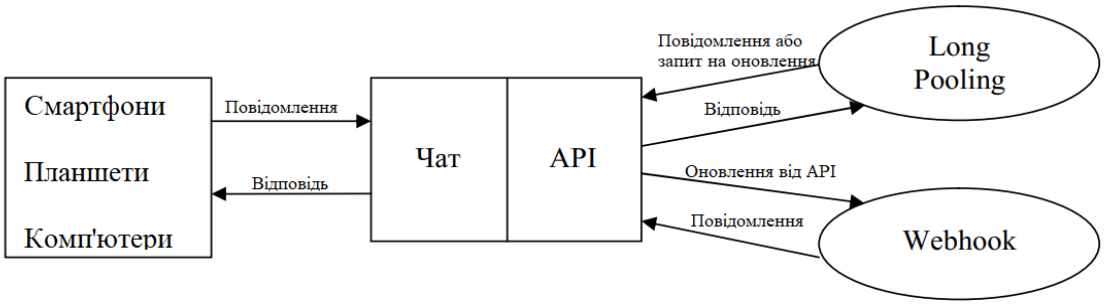

Рисунок 2.8 – Принцип роботи Telegram BOT API

Для того, щоб отримати token, необхідно написати спеціальному боту @BotFather.

Приклади доступних для API методів описані нижче:

– getUpdates – цей метод використовується для отримання оновлень за технологією long polling;

– setWebhook – метод прив'язує до боту url домену, де міститься запущений бот;

– sendMessage – метод відправляє текстове повідомлення до клієнту Telegram;

– sendLocation – метод відправляє повідомлення з координатами до клієнту Telegram;

– getFile – метод повертає дані з файлу по його імені та інше.

Web Apps for Bots

Боти Telegram можуть повністю замінити будь-який сайт. Вони підтримують безперебійну авторизацію, інтегровані платежі через 15 платіжних постачальників (з Google Pay і Apple Pay із коробки), доставку індивідуальних push-повідомлень користувачам і багато іншого.

З Web Apps боти отримують абсолютно новий вимір. Розробники ботів можуть створювати нескінченно гнучкі інтерфейси за допомогою JavaScript, найпоширенішої мови програмування у світі.

2.4 Опис архітектури програмного забезпечення

Проведемо проектування програмного забезпечення. Основним та найважливішим етапом проектування програмного забезпечення є вибір архітектури.

Архітектура програмного забезпечення - це процес структурування програмного забезпечення на слабко пов'язані та незалежні частини, що утворюють зв'язки та описують процеси передачі даних між ними.

Архітектура програмного забезпечення побудована таким чином, щоб максимально відповідати вимогам проекту.

Відповідно до сформованих функціональних вимог було прийнято рішення розглянути frontend, backend та MVC.

Архітектура клієнт-сервер є домінуючою концепцією створення розподілених мережних додатків, що встановлює правила взаємодії та обміну даними між ними.

Передбачені такі компоненти:

1. Набір серверів, які надають інформацію на запит.
2. Набір клієнтів, які використовують сервери та отриману від них інформацію.
3. Мережа, що забезпечує обмін інформацією між клієнтами та серверами.

Система управління чат-ботом має три сторони:

1. API ботів Telegram.
2. Сервер із базовою логікою чат-бота.
3. Клієнт.

Серверна частина - це сервери Telegram по відношенню до сервера логіки чат-бота, сервер логіки чат-бота по відношенню до веб-інтерфейсу клієнта, а клієнти - це сервери логіки чат-бота по відношенню до серверів Telegram та веб-інтерфейс клієнта по відношенню до сервери логіки чат бота (рис. 2.9).

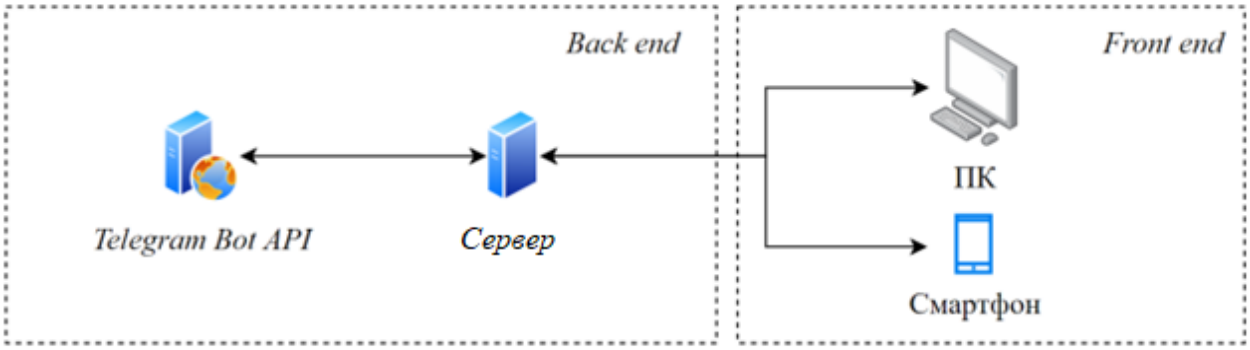

Рисунок 2.9 - Клієнт-серверний погляд на архітектуру системи

Клієнтський веб-інтерфейс відповідає за рівень представлення даних (відображення даних та введення команд).

Монолітна архітектура. При проектуванні великих систем чи систем із великою кількістю незакріплених компонентів дуже важливо вибрати базову архітектуру. За роки розробки програмного забезпечення сформувалися два основних підходи.

Монолітна архітектура — це програмна архітектура, коли всі компоненти програми перебувають у межах єдиного процесу ОС і обмінюються даними через внутрішні інтерфейси.

Використання серверної архітектури має такі переваги:

1. Спрощений процес розробки. Сучасні інтегровані середовища розробки (IDE) насамперед підтримують розробку серверних програм.
2. Спрощений процес розгортання.
3. Спрощений процес масштабування. Програму можна масштабувати, запускаючи кілька копій на балансувальнику навантаження.

Однак зі збільшенням складності та розміру програмного забезпечення мікросерверна архітектура набуває недоліків:

1. Складність у підтримці, розумінні та доопрацюваннях при великій кодовій базі програми.
2. Уповільнення швидкості IDE, процесу завантаження програм.
3. Запобігання частим оновленням програми, оновлення одного компонента заважає роботі інших через необхідність перезапуску всієї програми.
4. Помилка в одному компоненті призводить до відмови всієї системи.

5. Масштабування програми можливе лише в одній площині XY (рис.2.10).

6. Складність зміни технологій та їх конкретних версій.

Мікросереверна архітектура - це програмна архітектура, в якій всі її компоненти логічно розділені для виконання слабо пов'язаних функцій і виконання різних процесів ОС. Мікросервери однієї програми можуть працювати на різних фізичних машинах, а обмін між ними ґрунтується на стандартизованих протоколах RPI або обміну повідомленнями.

Використання мікросерверної архітектури має такі переваги:

1. Створення невеликих слабопов'язаних компонентів спрощує їх розуміння, розробку, тестування, підтримку, оновлення та розгортання.

2. Прискорити розробку за рахунок збільшення швидкості та скорочення часу завантаження серверів.

3. Підвищення надійності системи. Відмова або помилка в одному компоненті ізольовані від втручання у роботу інших.

4. Можливість заміни або оновлення технологій та фреймворків без критичного впливу на терміни або роботу проекту.

Недоліки мікросерверної архітектури:

1. Необхідно створити механізми обміну даними між серверами.

2. Ускладнює синхронізацію даних.

3. Ускладняються процедура розгортання мікросерверної програми.

Відповідно до завдання дипломного проекту, вимог та аналізу недоліків мікросерверної архітектури робиться висновок про те, що використання мікросерверної архітектури дасть наступні переваги:

1. Забезпечити незалежне розроблення кожного компонента системи.

2. Приведе до можливості більш простого масштабування за так званими осями X, Y та Z у порівнянні з серверною архітектурою (рис. 2.10) [13].

3. Забезпечити використання сучасних технологій та фреймворків, з можливістю легкої заміни за потреби.

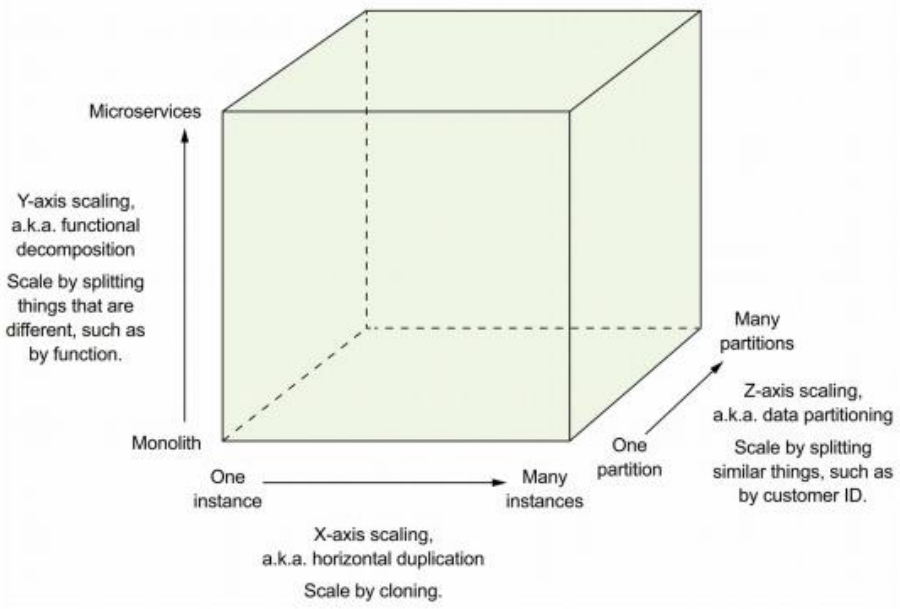

Рисунок 2.10. Масштабування серверних програм

REST API

REST — це стиль архітектури програмного забезпечення, що використовується для створення розподілених веб-сервісів, що масштабуються, що використовують HTTP-запити.

REST (Representational State Transfer) — стиль взаємодії компонентів розподіленої програми у мережі. REST має узгоджений набір обмежень, які враховуються під час розробки розподіленої системи. У певних випадках (інтернет-магазини, пошукові системи) це призводить до підвищення продуктивності та зміни архітектури, а саме її спрощення.

Компоненти в REST своєю взаємодією нагадують взаємодію клієнта та сервера в Інтернеті.

Виклик віддаленої процедури в Інтернеті може бути простим запитом HTTP (зазвичай «GET» або «POST»; такий запит називається «REST-запитом»), а всі дані, необхідні для передачі, передаються як параметри запиту. Для сервісів, створених з урахуванням REST, використовується так званий термін RESTful.

На відміну від веб-служб на основі SOAP, немає єдиного офіційного стандарту для терміну веб-API RESTful. Тому що REST – це архітектурний стиль, а SOAP – це протокол. REST сам не є стандартом, і, незважаючи на це, у більшості реалізацій RESTful використовуються добре відомі стандарти, такі як HTTP, URL, JSON і XML.

Розроблена система насамперед є програмним інструментом та її можна використовувати для будь-якої архітектури, а також вона заснована на алгоритмі та інструментах його обробки, однакових для будь-якої Web-плаформи.

Готове програмне забезпечення можна розмістити на будь-якій платформі, додавши додаткове програмне забезпечення – сервер, плагін або інтерфейс програми.

Клієнт-серверна архітектура підходить для організації на Web-платформі.

Для цього необхідно створити середовище у вигляді контейнера Docker з необхідною операційною системою, розмістити сервер, який керуватиме перетворювачем через свій програмний інтерфейс, та веб-сторінку для доступу та управління користувачами.

Звичайно, хостинг повинен підтримувати зовнішні статичні адреси. Також можна використати клієнт-сервер архітектуру, але це знизить загальну гнучкість системи. Загальна схема такої архітектури представлена нижче.

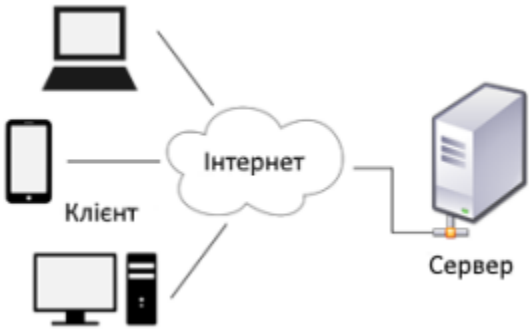

Рисунок 2.11 – Загальна схема клієнт-серверної архітектури

В веб-платформі, архітектура працюватиме як дочірній компонент основної програми, який міститиме графічний інтерфейс користувача та всю необхідну бізнес-логіку. Така архітектура називається міжпроцесною взаємодією.

Хорошим рішенням було б використовувати апаратне прискорення для архітектури, чи то потужний графічний процесор.

Виходячи з вимог кроссплатформенного перетворювача, що розробляється, і простоти реалізації готового програмного забезпечення, що демонструє його роботу. Готове програмне забезпечення матиме власний програмний інтерфейс та виконуватиме дочірній процес програми з графічним інтерфейсом користувача. Windows була обрана як цільова операційна система.

Message queue

Message queue (MQ) — це архітектура обміну повідомленнями між різними компонентами програмних систем в асинхронному режимі. Це дозволяє відправити повідомлення в одиному відправляючий компонент системи в один момент часу, а прийняти та обробити інший приймаючий компонент у зовсім іншому. Системи цього типу складаються з виробника (відправника) та споживача (одержувача), які

взаємодіють один з одним через провайдера (брокера), який інакше можна визначити як сервер MQ.

Черга - це структура даних з обмеженим доступом до елементів, яку можна описати як "перший прийшов, першим обслужений". Обмеження в тому, що елемент можна додати тільки до кінця черги, а вибрати елемент тільки з початку, при видаленні елемента з черги він знищується.

Особливості використання MQ-архітектури:

а) слабке зв'язування - створюються неявні інтерфейси, які обмінюються даними, дозволяють процесам бути незалежними один від одного та дозволяють підтримувати потрібний формат повідомлень;

б) надмірність - дозволяє уникнути нераціонального використання ресурсів системи чи мережі, своєю чергою, зберігання необробленої інформації;

в) масштабованість - за рахунок розподілу процесів обробки інформації дозволяє збільшити продуктивність MQ-сервера;

г) еластичність і здатність витримувати пікові навантаження - при високому навантаженні крім цього черги повідомлень можуть бути так званим буфером для накопичення інформації, дозволяючи змінювати швидкість обробки інформації і, таким чином, знижувати загальне навантаження на систему або мережу;

д) стабільність – дозволяє відокремити один процес від іншого та отримувати повідомлення, таким чином, у разі відмови процесора є можливість відновити робочий стан системи, відклавши обробку повідомлень;

е) гарантована доставка – повідомлення у будь-якому разі будуть оброблені та доставлені, незалежно від робочого статусу системи-відправника та системи-одержувача повідомлень. Це досягається за рахунок використання асинхронного зв'язку та можливості зберігати повідомлення на сервері MQ доти, доки воно не буде оброблено;

ж) гарантована та одноразова процедура доставки - значна частина серверів MQ дозволяє доставляти дані в тому порядку, в якому вони були надіслані, при цьому гарантуючи, що після прочитання повідомлення воно буде видалено з черги;

з) буферизація - це гарантія доставки повідомлень з максимальною ефективністю, яка досягається за рахунок незалежності швидкості обробки повідомлень від швидкості їх відправлення, це своє чергу пов'язано з використанням вищеописаної структури черги - своєрідного буфера між системою обробника і системою відправника.

2.5 Логічна та фізична модель бази даних

Для кращого розуміння створено модель бази даних, діаграму класів, діаграму варіантів використання. Ця діаграма (Додаток А) показує модель даних, яка описує концептуальні схеми за допомогою узагальненої блокової структури.

Діаграми класів є основою будь-якої проектної документації програмного забезпечення.

Діаграми класів дозволяють формалізувати логічну модель вашої програми на рівні структурних елементів системи (тобто класів). Віна надає статичне представлення структури програми – з яких класів вона складається, які зв’язки існують між цими класами та з чого складається кожен клас.

Модель кожного класу є шаблоном для майбутніх об’єктів, створених на основі цього класу. Кожна програма може містити велику кількість різних об'єктів. Як відомо, згідно з концепцією ООП, об’єкти мають стан і поведінку.

На рівні класу стан описується полями даних, а поведінка – методами.

На діаграмі (рисунок 2.12) показані всі основні класи, без яких система певною мірою не може функціонувати.

У лівій частині малюнка описано один із найважливіших класів - Користувач, який є класом, що описує всіх користувачів системи (адміністраторів, користувач). Також показано зв’язок між класом «Користувач» і класом доступу «Дозвіл» і зв’язок «багато-до-багатьох» із проміжним класом.

Ці відвідування, у свою чергу, пов’язуються з категоріями груп користувачів через відповідні посилання.

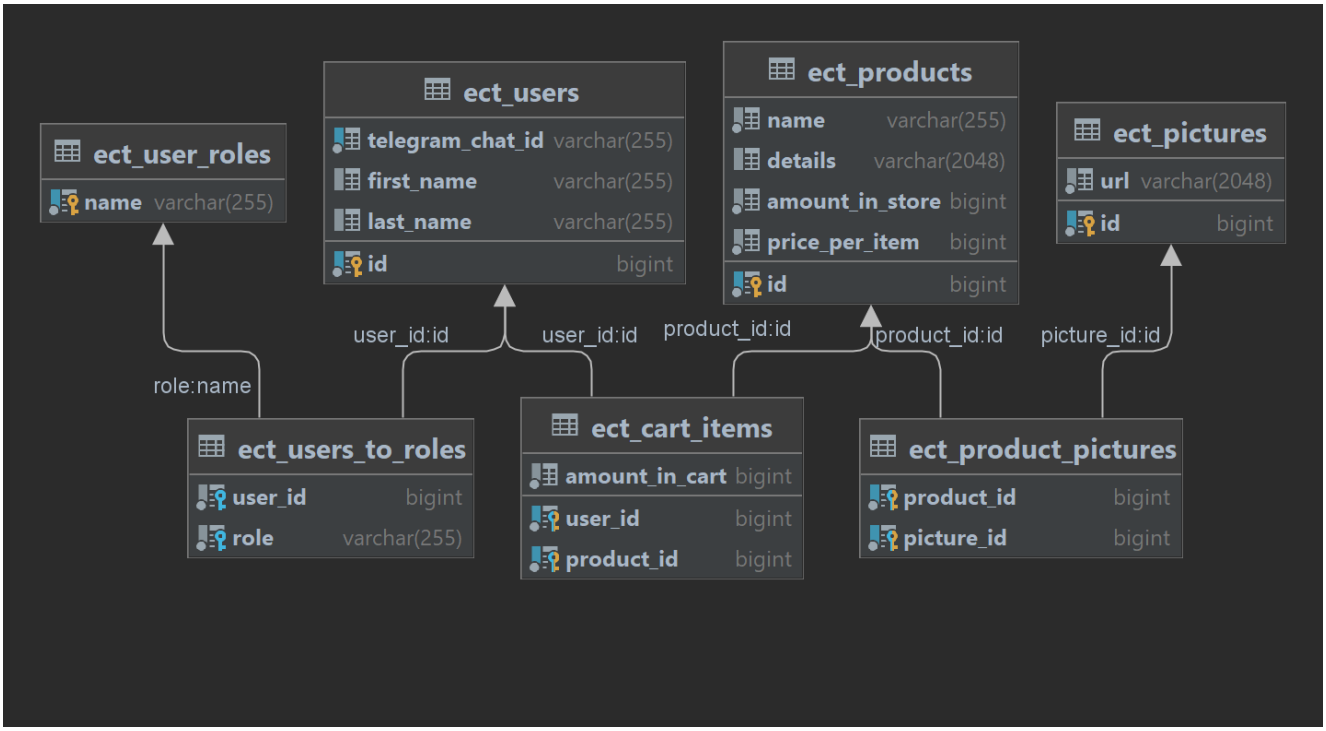

Рисунок 2.12 - Діаграма бази даних

Use case діаграма

Метою діаграми є охоплення динамічних аспектів системи.

Основними цілями створення діаграми варіантів використання (рис. 2.13) є:

– збирати вимоги;

– отримати зовнішній вигляд системи;

– вплив зовнішніх і внутрішніх факторів;

– візуалізація взаємодії між вимогами та темами.

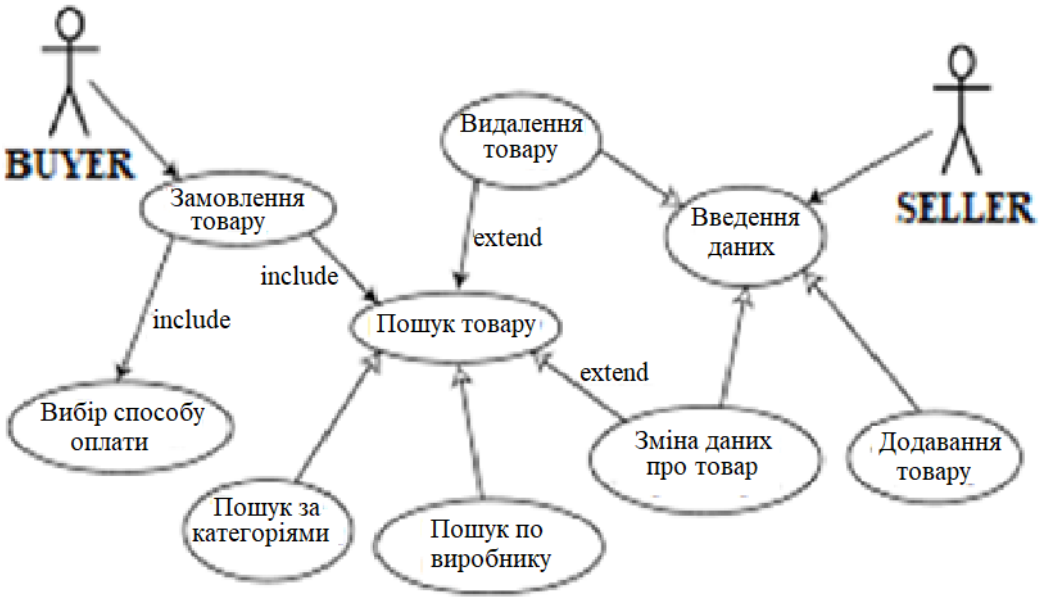

Рисунок 2.13 - Діаграма use case

РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ – ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ ПЛАТФОРМИ ЕЛЕКТРОННОЇ КОМЕРЦІЇ НА БАЗІ ТЕЛЕГРАМ

3.1 Розробка програмного продукту

Почнемо з того, що для кожної послуги необхідно було вибрати загальний шаблон побудови, тому, проаналізувавши існуючі концепції, був обраний шаблон, який розділяє послуги на відповідні рівні відповідальності. Суть цього шаблону полягає в тому, що він визначає межу між додатком і сервісним рівнем, формує набір доступних операцій і в кожній операції керує результатами додатка. Тобто шаблон визначає набір дозволених операцій для програми з точки зору клієнта. Він інкапсулює бізнес-логіку програми, керує транзакціями та керує відповідями під час виконання цих операцій.

Тому основні модулі, які варто виділити, будуть містити всі сервіси магазину, що розробляється, а саме:

- рівень web-api;
- рівень бізнес-логіки;
- обробка помилок;
- інтеграція з іншими ресурсами;
- конфігурація;
- стабільність;
- рівень взаємодії з репозиторієм;
- модуль util.

Рівень Web-API відповідає за відображення інформації користувачеві та інтерпретацію запитів клієнта. Клієнтом може бути як користувач системи, так і інший сервер. Сервери отримують запити за допомогою HTTP, а також мають підтримувати принципи REST для кращої зручності.

Підсумовуючи, можна зрозуміти, що цей модуль використовується як вхід до серверу, тобто клієнт надсилає запити на ресурси, рівень web-api отримує, аналізує вхідні параметри та передає бізнес-логіку відповідному сервісу, з якого він формує зворотну відповідь і повертає її клієнту

Бізнес-рівень — це частина корпоративної системи, яка визначає, як певні дані обробляються, обробляються чи обчислюються. У прикладі продукту, який ми розробляємо, логіка створення нових замовлень або перегляду та редагування існуючих замовлень є бізнес-логікою, механізмом, який вирішує бізнес-завдання програми.

Репозиторій — це компонент, який надає багато інструментів для взаємодії з базою даних. Як правило, репозиторій застосовується лише до одного типу сутності. У цих компонентах реалізовані всі методи пошуку, зберігання та видалення за певних умов.

Для обробки всіх можливих помилок у веб-додатку створюються відповідні модулі. Як правило, їхні завдання включають виявлення помилок, аналіз умов, за яких вони виникли, та генерування відповідних повідомлень. Оскільки наша програма базується на сервері, модуль обробки помилок повинен вибрати код для відповіді мережі на запит, коли виникає помилка, залежно від обставин, за яких сталася помилка.

Для інтеграції з іншими ресурсами мережі були створені окремі модулі, які також називаються комунікаційними сервісами. Основною відповідальністю таких служб є створення нових HTTP-запитів до інших ресурсів, а також обробка відповідей на запити та обробки помилок, пов’язаних з інтеграцією.

Часто на практиці додатки потрібно налаштовувати за певних умов. Наприклад, потрібно відключити модулі інтеграції з платіжними системами. Клас конфігурації був створений для вирішення цієї проблеми. Як правило, вони

читають параметри конфігурації та застосовують їх до логіки програми. Наприклад: налаштування запитів: очікування відповіді, обмеження розміру вхідних запитів, підтримка розширення ресурсів, перехоплювачі при отриманні запитів.

Таблиця 3.1 – обрання технологій та реалізацій

№ п/п	Ідея проекту	Технології її реалізації	Наявність технологій	Доступність технологій
1.	Мова програмування	Java	Наявні	Доступні
2.	Мова запитів до бази даних	SQL	Наявні	Доступні
3.	Система керування базою даних	MySQL	Наявні	Доступні
4.	ORM система	Hibernate	Наявні	Доступні
5.	Система допомоги обробки запитів	Spring Data	Наявні	Доступні
6.	Середа розгортання за допомогою	Tomcat Spring Boot	Наявні	Доступні
7.	Безпека системи	Spring Security	Наявні	Доступні
8.	Інтерфейс користувача	HTML/CSS	Наявні	Доступні
9.	Взаємодія між користувачем і системою	Spring MVC/ REST	Наявні	Доступні
10.	Обробка запитів користувача	Власна реалізація	Власна реалізація	Власна реалізація
11.	Збереження даних користувача	Власна реалізація	Власна реалізація	Власна реалізація

Інтерфейс між покупцем та базою даних категорій і товарів повинен бути інтуїтивно зрозумілим, дружнім і легкодоступним. Основне завдання – швидкість процесу роботи з базою даних категорій і товарів.

Для того, щоб можна було додавати новий продукт або послугу, підкатегорії або категорії, в базу даних будуть створені спеціальні форми з полями для введення даних, будь то текст і/або зображення.

Адміністратор інтернет-магазину наповнює телеграм-бот магазину контентом, тобто товарами та послугами, вводить такі дані, як опис товару, умови доставки, зображення товару, способи оплати та інша важлива інформація. Покупець, вивчаючи категорії товарів телеграм-боту, робить вибір на користь необхідного товару, створює замовлення, при цьому йому необхідно вказати свої дані, вибрати спосіб оплати та доставки, після цих дій адміністратор телеграм-боту буде надіслано електронний лист з повідомленням про формування замовлення, а покупцеві - лист з повідомленням про підтвердження замовлення. Потім покупець оплачує товари, при цьому будуть задіяні використовувані платіжні системи, і отримує товар обраним ним способом.

Процес взаємних дій між клієнтом і інтернет-магазином буде виглядати так:

1) відвідування телеграм-боту;

2) вибір необхідного товару та додавання його в кошик;

3) менеджер компанії здійснює зворотний дзвінок, за допомогою якого підтверджує замовлення, перевіряє контактні дані, вказує адресу доставки та спосіб оплати;

4) після отримання інформації від покупця вони будуть передані на склад, а потім у службу доставки;

5) товар буде доставлено у визначений замовником час та за вказаною адресою.

У системі керування контентом застосовується СУБД MySQL. База даних складається з основних таблиць:

1. Товари у кошику
2. Картинки
3. Картинки продуктів
4. Продукти
5. Роль користувачів
6. Користувачі
7. Взаємодія користувачів

Всі дані зберігаються на локальному сервері в окремих таблицях базі даних

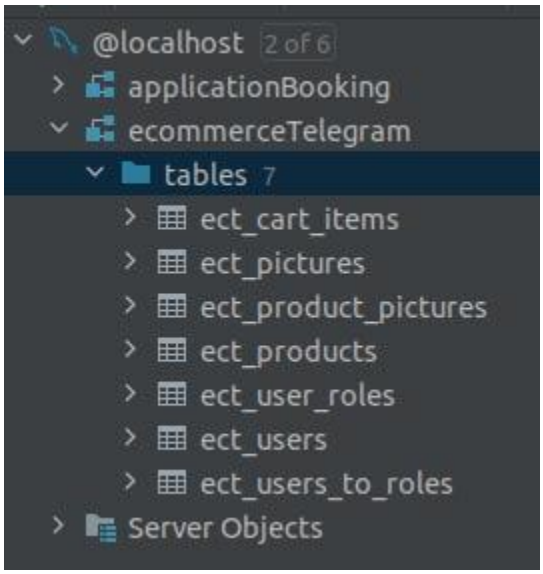

Рис. 3.1 – Структура БД(Загальна)

3.2 Опис функціоналу

Усі сервери побудовані з використанням середовища Spring, що значно спрощує процес впровадження програмного забезпечення, оскільки Spring бере на себе багато завдань. Наприклад, великі частини коду часто повторюються. Щоб уникнути повторення, Spring надає стереотипи для інкапсуляції анотацій для повторюваного коду. Усі додатки повинні запускатися та бути налаштованими на певному сервері. Фреймворк повністю виконує це завдання, і анотація @SpringBootApplication використовується в наступному фрагменті коду, що означає, що фреймворк запустить окремий сервер TomCat під час запуску програми та запустить програму на ньому.

```
@EnableScheduling
@SpringBootApplicat
ion
public class
    EcommerceTelegramBotApplication {
    public static void main(String[]
    args) {
        ApiContextInitializer.init();
        SpringApplication.run(ClientTelegramApplication.cl
        ass, args);
    }}
```

Цей фрагмент є точкою ініціалізації для серверу. Ви також можете помітити анотацію, яка дозволяє використовувати базу даних MySQL, і анотацію @EnableScheduling, яка вказує програмі мати класи, логіка яких виконується циклічно через певні проміжки часу. Кожен сервер містить файли властивостей application.yml — вони зберігають конфігураційну інформацію програми у вигляді ключових значень, наприклад, з якого порту запускається програма, з якою адресою працює база даних, токен-бот, який використовується для доступу до Telegram, URI інших серверів і багатьох інших важливих властивостей. Крім того,

у кожному проекті є файл build, який містить описи залежностей, які необхідні для створення програми за допомогою Maven.

Розглянемо реалізацію основних модулів як приклад магазину. Почнемо з пакету API, який відповідає за обробку всіх HTTP-запитів до серверу. Цей пакет містить класи, які є контролерами відповідно до їх функціональних можливостей, і такі контролери позначені анотацією @RestController, яка, у свою чергу, є ідентифікатором фреймворку IoC. Тобто, коли програма отримає запит від третьої сторони, фреймворк перевіряє, чи немає в цих контролерах методів, які відповідають параметрам запиту. Пакет API містить такі контролери: замовлення, транзакції, товари, інвентар, користувачі, статистика. Наприклад, розглянемо UserController.

```
@Slf4j
@RestControl
ler
@RequiredArgsConstructo
r
@RequestMapping("api/us
er") public class
UserController {
    private final UserService
    userService;
    @GetMapping("/{id}")
    public ResponseEntity<UserDto> getById(@PathVariable int
        id) {  UserDto byId =
        UserTransformer.transform(userService.getById(id));

        sout.debug("User {} found", byId);
        return new ResponseEntity<>(byId,
    HttpStatus.OK); }
@PostMapping
public ResponseEntity<UserDto> create(@RequestBody
    UserDto userDto) { userDto.setAdmin(false);
    userDto.setSupport(fa
    lse); UserDto save =
UserTransformer.transform(userService.saveNewUser(UserTransformer.
transform(user Dto)));
```

```
    sout.debug("User {} saved", save);
    return new ResponseEntity<>(save,
    HttpStatus.CREATED);} @PutMapping("/{id}")
    public ResponseEntity<UserDto> update(@PathVariable int id,
@RequestBody UserDto user) {UserDto update =
UserTransformer.transform(userService.update(id,
UserTransformer.transform(user)));
        sout.debug("User with id:{} updated to {}",
        id, user); return new ResponseEntity<>(update,
        HttpStatus.CREATED);
    }
```

У наведеному прикладі реалізовано функціонал пошуку користувачів за ідентифікатором та створення нових користувачів у магазині. Ви помітите, що анотація @RequestMapping використовується для вказівки, з якого URI цей контролер отримуватиме запити. Методи включають класи з префіксом DTO - Data Transfer Object. Це ознака об'єкта, що він потрапив у додаток ззовні. У кожному методі використовується метод для перетворення об’єкта в модель, об’єкт, з яким працює служба. Ви можете помітити анотацію @Slf4 - вона додає механізм журналювання до класу, приклад можна побачити в методі UserController. Зазвичай такі логи записуються у файл, і за допомогою цього файлу, якщо програма робить помилку, можна зрозуміти, де і чому сталася помилка. У пакеті API є пакет dto - він зберігає об'єкти, які надходять до програми в http-запитах. У http-запитах вони відображаються як текст, але коли вони входять у програму, Spring автоматично використовує вбудовані бібліотеки для серіалізації таких об’єктів із тексту в класи Java. Приклад такого об'єкта наведено нижче.

```
@ Data
@ Builderr
@
NoArgsConstructo
r @
AllArgsConstruct
```

```
or
public class UserDto {
    private Integer id;
    private Long chatId;
    private Integer
    telegaId;
    private String
    telegaFullName; private
    String telegaUserName;
    private String language;
    private String fullName;
    private String phone;
    private String
    deliveryAddress; private
    Boolean admin;
    private Boolean
    support; private
    Boolean modified;
    private Boolean
    created; private
    Boolean superAdmin;}
```

Для анотацій @Data, @Builder, @NoArgsConstructor, @AllArgsConstructor — підтримується бібліотека Lombok. Ці анотації генерують код під час компіляції JVM. @data генерує методи ToString, геттери та сетери для всіх полів класу, а також конструктори для створення об’єктів і методи для хеш-кодів і дорівнює.

@NoArgsConstructor генерує конструктор без аргументів, якому потрібен такий клас для серіалізації JSON у клас JAVA. @Builder реалізує конструктор у класах шаблонів, які можна використовувати як заміну конструкторів, де кожне поле створеного класу можна передати як виклик нового методу. Наступним важливим компонентом є клас Transformer, який містить методи для перетворення DTO в сутності. Приклад цього класу наведено нижче.

Щоб запустити всю систему серверів, повністю готову до взаємодії, необхідно зробити наступне: - зареєструвати бота в системі telegram;

- запуск бази даних MySQL для магазину;

– встановіть JDK 17 і запустіть файл jar серверу;

Щоб створити бота в системі Telegram, це потрібно зробити за допомогою офіційного бота Telegram BotFather. на рис. 3.2 наведено приклад створення бота - реєстрація його публічного імені та отримання токена.

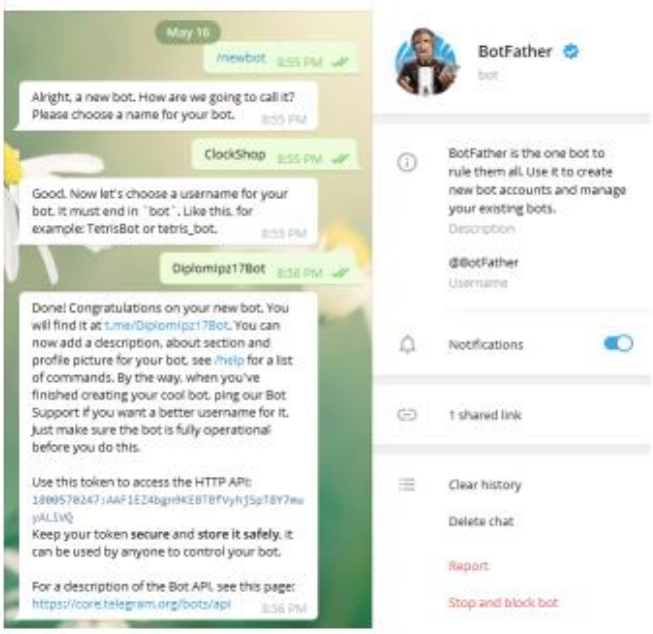

Рисунок 3.2 – Процес створення чат-бота

Згенерований маркер і логін повинні бути вказані в application.uml. Наступний важливий момент полягає в тому, що на сервері, де розгортається програма, повинна бути встановлена JDK 17.

Детальні інструкції щодо встановлення реляційної бази даних можна знайти на офіційному веб-сайті MySQL [21], а також ви можете знайти відповідні інструкції відповідно до ОС сервера. Базу даних можна зручно налаштувати, підключені до бази даних, а саме на порт і адресу. Ці дані потрібно передати в програму Java за допомогою параметрів.

3.3. Тестування програмного продукту для е-комерції

Вибираючи інструменти та інструменти для тестування програми, слід мати на увазі, що розроблений додаток має бути розташований на сервері, а також, слід враховувати, що кожен окремий сервіс виконує певний набір завдань.

Враховуючи вищезазначені аспекти серверу, було вирішено реалізувати детальну модульність та інтеграційне тестування логіки серверу системи.

Unit тестування – це перша фаза тестування системи, яка включає індивідуальне тестування кожного unit коду. Unit - це найменша частина програми, яку можна протестувати. На прикладі веб-додатку, що розробляється, такі модулі є методами класу. Як правило, такі тести створюються розробниками під час розробки бізнес-логіки, щоб переконатися, що код відповідає вимогам технічного завдання, архітектурі програми та має очікувану поведінку.

Наступним етапом тестування програми є інтеграційне тестування, яке відбувається послідовно після модульного тестування. Суть інтеграційного тестування полягає в модульному тестуванні окремого компонента програми, тоді як в інтеграційному тестуванні програмні модулі об’єднуються для тестування та взаємодії. Як правило, для цього типу тестування необхідно розгорнути прикладне середовище, у якому відповідний компонент буде працювати. Важливим моментом є, оскільки веб-програми взаємодіють одна з одною через запити, ці запити включені в деякі заглушки (моки) для тестування. Такі макети завжди повертають результати, описані в макеті. Це дозволяє тестувати кожен сервер окремо.

Для прикладу інтеграційного тесту взаємодія клієнта з магазином буде заблокована, і тест не виходитиме за межі самого серверу. Зазвичай цей підхід до попереднього тестування, як і підхід до інтеграції, відповідає принципу «чорного

ящика» — під час тестування не враховуються внутрішні механізми системного додатка.

Тести веб-додатку написані після впровадження основного функціоналу відповідно до підходу BDD (Behavior Driven Development). BDD поєднує основи та методи TDD, головним чином дозволяючи розробникам писати тести після реалізації логіки в системах, які важко розробити в деталях.

Для модульного та ітераційного тестування Java-додатків використовувалися такі бібліотеки: Junit, Mockito, Assertj, H2 Database, Spring Boot Starter Test. Junit є одним із найпопулярніших фреймворків модульного тестування в екосистемі Java, який використовується в 30% усіх проектів на GitHub станом на 2021 рік[21].

Поточна версія 5 включає великий набір модулів і інструментів тестування інтеграції. За допомогою цього фреймворку ви можете структурувати умови тестування та створювати навантажувальні тести програмного забезпечення та динамічні тести.

AssertJ — це бібліотека для Java, яка спрощує процес написання тестів. Він надає великий набір тверджень та інформативних повідомлень на основі тестів, покращуючи розуміння тестового коду.

Фреймворк Mockito надає багато варіантів для створення заглушок замість реальних класів або інтерфейсів під час написання тестів за допомогою Junit. База даних H2 генерує в пам’яті умовну базу даних, достатню для тестування. Зазвичай надає мало функціональних можливостей, але, з іншого боку, забезпечує високу продуктивність і усуває необхідність ініціалізації окремого сервера бази даних для тестування. Spring Boot Test Harness — це модуль Spring Framework, який надає можливість створювати окреме тестове середовище для кожного виконання тесту.

Почнемо з тестування окремих модулів програми. У тестовому сценарії ми перевіряємо правильність читання та обробки певної компонентної інформації.

Приклади таких тестових сценаріїв модульного тесту описано та наведено в таблиці 3.2.

Таблиця 3.2 – Тестові сценарії для модульних тестів

Модуль	Вихідні дані	Очікуваний результат
Перегляд товарів	Фільтри і сортування	Список товарів
Редагування контакту	ПІБ, телефон, адреса	Користувач створений
Додавання товарів в корзину	Динамічно сформований	Замовлення успішно створено, якщо товарів немає в наявності - відповідна помилка.
Відміна замовлення	Існуюче замовлення	Замовлення успішно відмінено, користувачу повертаються вартість замовлення
Зміна прав користувача	Ідентифікатор користувача	Користувачу надані права відповідно вхідним даним
Перегляд інформації про замовлення	Ідентифікатор товару	Інформація про товар успішно сформувалась
Обробка вхідного повідомлення користувача	Вхідне текстове повідомлення, сесія користувача	Визначення дії клієнта. Аналіз введеної інформації на відповідність. Обробка подій з введеним текстом.

Переходимо до інтеграційного тестування. Для прикладу розглянемо функціональність використання багатьох компонентів програми, таких як: перегляд і додавання товарів у кошик, створення замовлень. Для виконання таких операцій використовується модуль контролера client-tg, сервіс client tg, інтеграція з серверами та модулі для взаємодії з Telegram API. Було використано бібліотеки Mockito та JUnit для «спрощення» запитів до admin-tg. Подібні сценарії для цих інтеграційних тестів показано в таблиці 3.3 нижче.

Таблиця 3.3 – Тестові сценарії для інтеграційних тестів

Модуль	Вихідні дані	Очікуваний результат
Перегляд існуючих в магазині товарів	Сортування за ціною	Звернення до магазину. Список товарів, які відсортовані по вартості
Оновлення персональної інформації користувача	Ідентифікатор користувача. Ім’я, номер, адреса доставки товарів	Звернення до магазину. Валідація вхідних даних, оновлення існуючої інформації про користувача.
Створення нової оплати	Дані транзакції	Валідація даних, створення нового замовлення і формування посилання для оплати замовлення.

Аналіз результатів тестування системи.

Модульні та інтеграційні тести мають бути виконані правильно.

Використано інструменти створення проектів Maven та інструменти IDE для тестування. На рисунку 3.3 показана частина інтерфейсу Intellij IDEA, яка показує можливі тести на сервері.

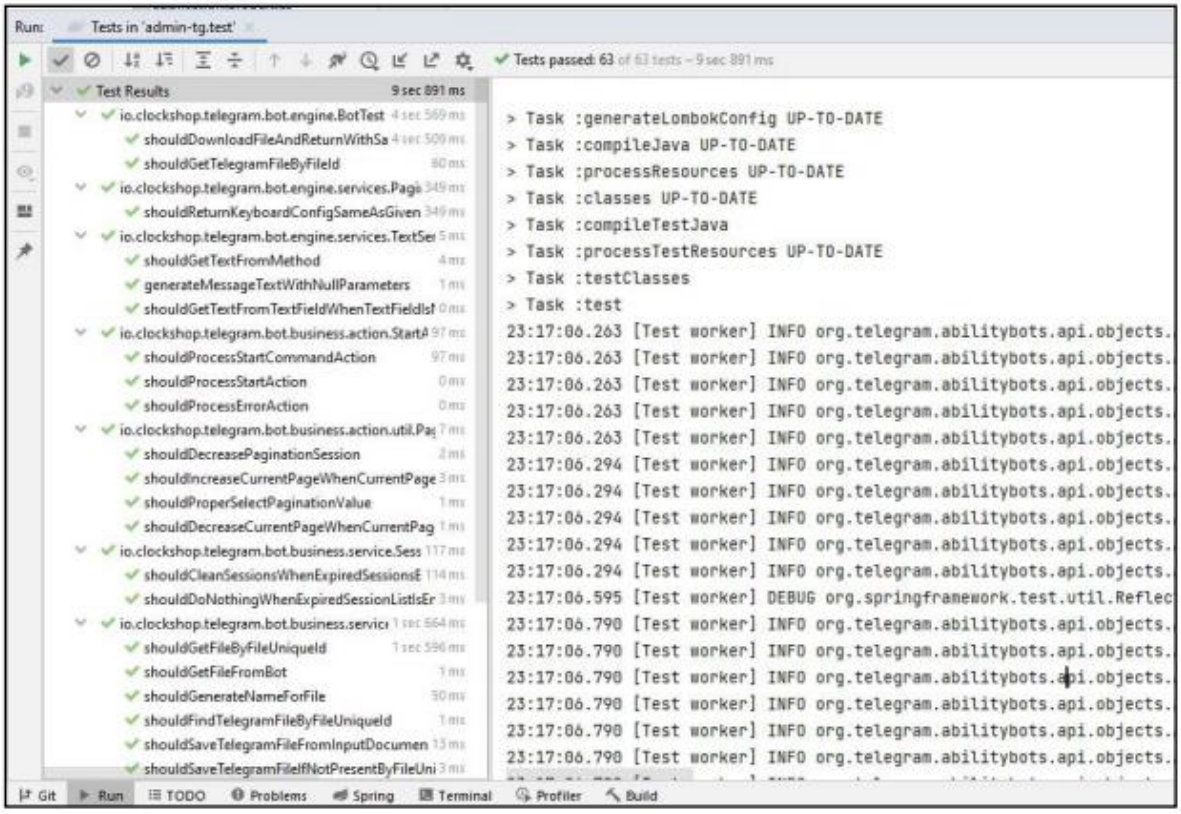

Рисунок 3.3 – Запропоновані тестування

На рисунку ми бачимо, що всі 63 модулі та інтеграційні тести серверу admin-tg які було запропоновано для реалізації. Також можна побачити журнали, що описують життєвий цикл кожного тесту в інтерфейсі. Такий журнал може контролювати виконання тестових операцій у разі невдалого виконання тесту.

Запропоновані сценарії для тестування наведено в таблиці 3.4:

Таблиця 3.4 – Запропоновані тестових сценаріїв

Опис	Вхідні значення	Вихідні значення	Результат
Перегляд товарів	Фільтри і сортування	Список товарів	Правильно

Редагування контакту	ПІБ, телефон, адреса	Користувач створений	Правильно
Добавлення товарів в корзину	Динамічно сформований товар	Замовлення успішно створено, якщо товарів немає в наявності - відповідна помилка.	Правильно
Відміна замовлення	Існуюче замовлення	Замовлення успішно відмінено, користувачу повертаються вартість замовлення	Правильно
Перегляд інформації про замовлення	Ідентифікатор товару	Інформація про товар успішно сформувалась	Правильно
Аналіз вхідного тексту користувача	Повідомлення, сесія користувача	Аналіз дії користувача. Перевірка інформації на відповідність. Аналіз дії з введеним текстом.	Правильно
Перегляд існуючих в магазині товарів	Сортування за ціною за зростанням	Звернення до магазину для отримання списку товарів.	Правильно
Обновлення персональної інформації користувача	Ідентифікатор користувача. Ім’я, номер, адреса доставки товарів	Оновлена інформація про користувача магазину	Правильно
Створення нової оплати	Ідентифікатор користувача. Ім’я, номер, адреса доставки товарів	Звернення до магазину. Валідація вхідних даних, оновлення існуючої інформації про користувача	Правильно

Завдяки успішному тестуванню веб-додатку на різних рівнях системи та різноманітним методам тестування було виявлено, що функції реалізовані

відповідно до функціональних вимог програмного забезпечення, тому спроектоване програмне забезпечення працездатне та може витримувати навантаження у вигляді звичайних користувачів магазину.

У цьому розділі вибрано та продемонстровано підхід до тестування системи. Була протестована система на різних рівнях за допомогою модульних та інтеграційних тестів. Обрано інструменти для реалізації тестування на різних рівнях системи та з використанням різних методологій. Таким чином, система реалізована згідно технічного завдання, а веб-додаток повністю функціональний і готовий до використання.

3.4. Керівництво для користувача

По-перше, щоб користувачі могли використовувати бота, їм потрібно знайти його в Telegram за допомогою вбудованої функції Search, а потім додати його до списку активних контактів. Взаємодія з чат-ботом починається з активації користувачем чат-бота команди /start. Після виконання цієї команди з’явиться стартова сторінка.

У стартовому вікні користувач може побачити функціонал перегляду товарів, переглянути свої активні замовлення та налаштувати чат-боти. Інтерфейс у наведеному прикладі англійський, тому активною мовою клієнта Telegram є англійська.

Щоб побачити товари в магазині, потрібно зайти в меню товарів (рис. 3.4). У цьому меню ви можете сортувати та фільтрувати товари за необхідними критеріями, такими як: стиль, виробник, механізм, ціна. Активні фільтри можна видалити в меню «Очистити фільтри». Виріб представлено у вигляді кнопки, а напис на кнопці - це назва товару. Якщо товарів багато, вони розбиті на сторінки,

які легко гортати за допомогою стрілок. Якщо всі товари поміщаються на одній сторінці, стрілки для позначення сторінок не відображатимуться. Вибрані фільтри можна очистити в пункті меню «Очистити фільтри».

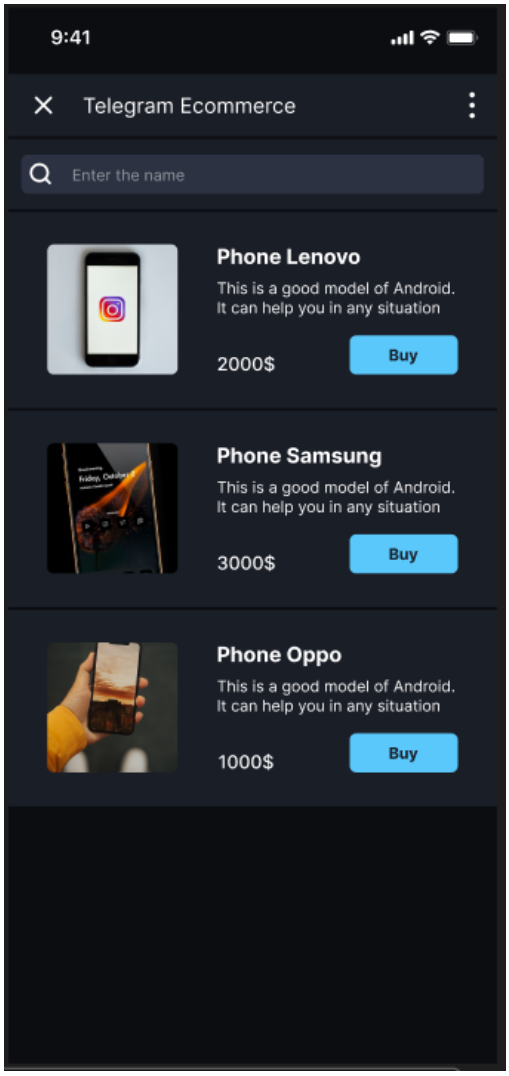

Рисунок 3.4 – Початкова взаємодія з чат-ботом

Користувачі можуть детально переглянути будь-який товар, доступний у магазині, натиснувши назву товару. Інформація про товар відображається в інформаційному повідомленні. Для зручності додана кнопка додавання товару в

кошик. Після цього товар додається в кошик - замовлення можна переглянути у відповідному меню. Також у цьому меню можна переглянути інформацію про активність щодо адреси доставки, номера телефону та імені одержувача замовлення.

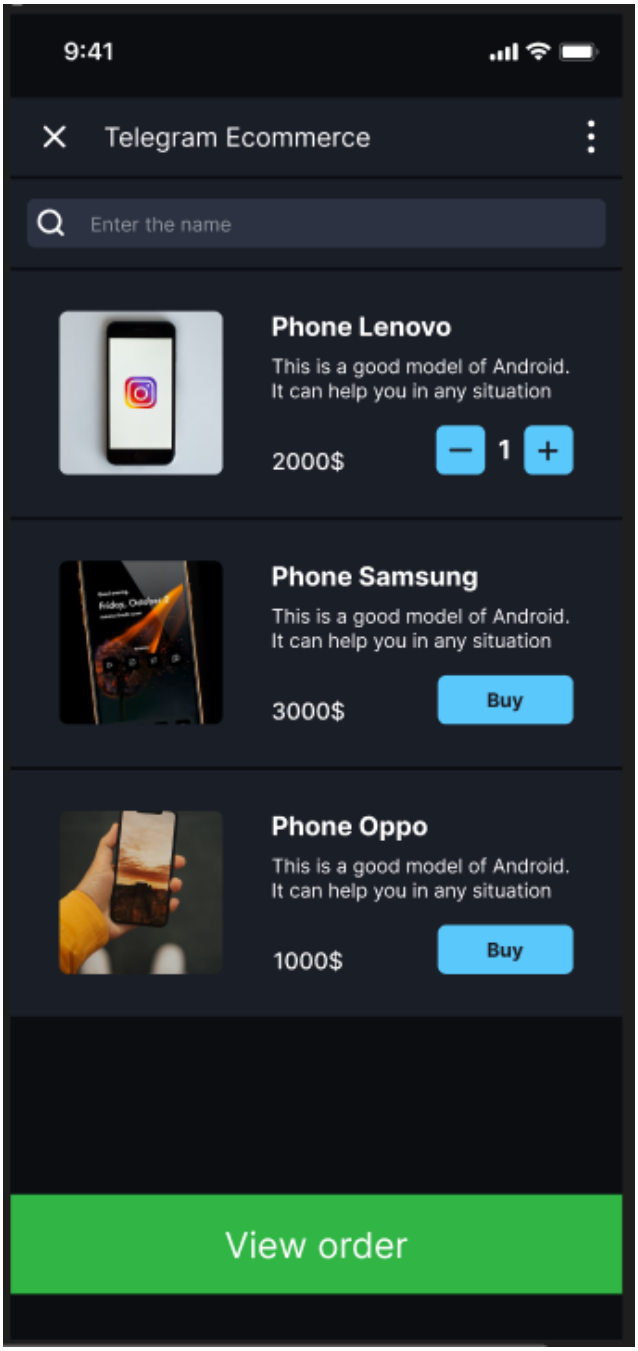

Рисунок 3.5 – Перегляд інформації про замовлення

Та загальної вартості (рис. 3.6)

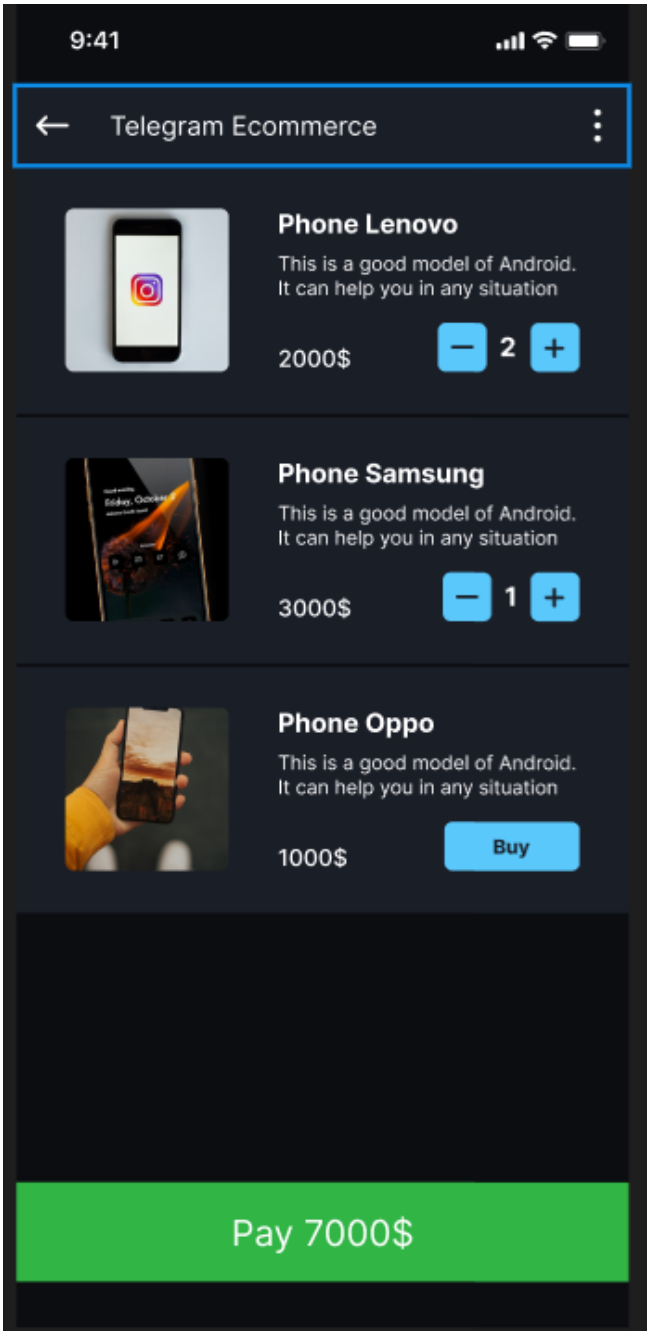

Рисунок 3.6 – загальна вартість товарів

Ви також можете змінити свою контактну інформацію та підтвердити своє замовлення в цьому меню. Для цього необхідно вибрати спосіб оплати. Активні замовлення можна переглянути в Замовленнях (рис. 3.7).

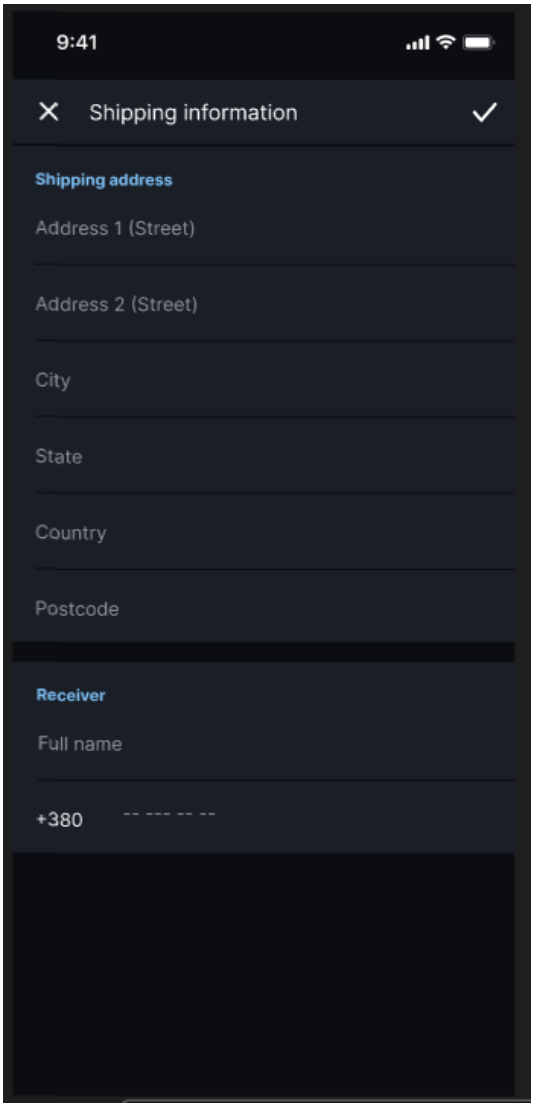

Рисунок 3.7 – оформлення замовлення.

ВИСНОВКИ

Підсумовуючи дипломну роботу, можна сказати, що розроблений телеграм-бот дозволить власнику значно збільшити продажі товарів і послуг, розширити кількість покупців, дати можливість скоротити період часу. Весь процес від надання продукту до його продажу, покращення управління телеграм-боту, зниження витрат на оренду або купівлю торгових приміщень та найму численного персоналу.

У магазині стало можливим висвітлити його основні функції та послідовність взаємодії з користувачами, також розроблено структуру та алгоритм роботи цього телеграм-боту та виділено його основні функції.

На сьогоднішній день існує величезна кількість технологій розробки телеграм-ботів, але необхідно використовувати і застосовувати новітні сучасні інструменти розробки.

Цей веб-сервіс розроблено з використанням Java, Spring Boot, Thymeleaf, Hibernate, JavaScript, MySQL, Docker, Gitlab + Gitlab CI.

Метою дипломної роботи була розробка веб-платформи електронної комерції на основі Telegram. У ході досягнення цієї мети було вирішено наступні завдання:

1) аналіз предметної області;

2) проаналізовано новітні технології розробки телеграм-ботів;

3) проаналізовано особливості розвитку сучасних веб-платформи з точки зору проектування дизайну телеграм-боту;

4) вивчено сучасні засоби розробки телеграм-боту;

5) реалізовано необхідний функціонал;

6) на віртуальному хостингу розміщено телеграм-бот.

Розроблений телеграм-бот має перспективи для подальшого розвитку та вдосконалення. Отже поставлені завдання виконані.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Комп'ютерна система PLATO [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/PLATO.
2. Talkomatic – Just think of it [Електронний ресурс] – Режим доступу: https://just.thinkofit.com/talkomatic/.
3. AIM messenger changed the way we communicate [Електронний ресурс] – Режим доступу: https://www.vox.com/culture/2017/12/15/16780418/aim-aol-instant-messenger-shutdowncultural-impact.
4. History of Friendster [Електронний ресурс] – Режим доступу: https://socialnetworking.lovetoknow.com/history-friendster.
5. Facebook [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Facebook.
6. Хто та як використовує месенджери в Україні? [Електронний ресурс] – Режим доступу : https://minfin.com.ua/2018/04/14/33167841/.
7. Рейтинг мобільних додатків за березень 2020 [Електронний ресурс] – Режим доступу : https://tns-ua.com/news/reyting-mobilnih-dodatkiv-za-berezen-2020.
8. Messina, C. 2016 will be the year of conversational commerce [Електронний ресурс] / C. Messina // Medium. – 2016. – Режим доступу: https://medium.com/chris-messina/2016-will-be-the-year-of-conversationalcommerce-1586e85e3991.
9. MTPoto Mobile Protocol [Електронний ресурс] – Режим доступу: https://core.telegram.org/mtproto. Дата звернення – травень 2020 р.
10. Козлов А. А. Телеграм-бот як простий та зручний спосіб отримання інформації [Електронний ресурс] / А. А. Козлов, А. В. Батищев // Територія науки. – 2017. – №5. – с. 55-64.

11. Ebot one – редактор ботів для Telegram [Електронний ресурс] – Режим доступу: https://ebot.one/.
12. Manybot – платформа для створення ботів [Електронний ресурс] – Режим доступу: https://manybot.io/.
13. How to make a responsive telegram bot [Електронний ресурс] – Режим доступу: https://www.sohamkamani.com/blog/2016/09/21/making-atelegram-bot/.
14. Васильєв О. Python на прикладах. Практичний курс по програмуванню / Васильєв Олексій., 2016. – С. 432.
15. Введення до модулю os [Електронний ресурс] – Режим доступу : https://docs.python.org/3/library/os.html.
16. Використання модулю subprocess [Електронний ресурс] – Режим доступу: https://docs.python.org/3/library/subprocess.html?highlight=subprocess#module-subprocess.
17. Використання пакету logging [Електронний ресурс] – Режим доступу: https://docs.python.org/3/library/logging.html?highlight=logging#module-logging..
18. Приклади ботів для Telegram [Електронний ресурс] – Режим доступу: https://github.com/TelegramBots/telegram.bot.examples.
19. WeChat – додаток– Режим доступ: https://ain.ua/special/wechat-veon/
20. Етапи створення чат ботів [Електронний ресурс] – Режим доступу: https://apix-drive.com/ru/blog/ecommerce/chat-bot
21. Інтернет магазин Lamoda [Електронний ресурс] – Режим доступу: https://www.lamoda.ru
22. Встановлення реляційної бази даних MySQL [Електронний ресурс] – Режим доступу: https://uk.education-wiki.com/1727095-mysql-relational-database

23.20 найпопулярніших інструментів модульного тестування в 2021 році. [Електронний ресурс] – Режим доступу: https://uk.myservername.com/20-most-popular-unit-testing-tools-2021

ДОДАТКИ

Додаток А (Діаграма класів)

Додаток Б (лістинг коду)

```
@Configuration
public class AwsS3Configuration {

    @Bean
    public TransferManager transferManager(AmazonS3 amazonS3) {
        return
TransferManagerBuilder.standard().withS3Client(amazonS3).build();
    }

@EnableWebSecurity
public class WebSecurityConfiguration {

    @Bean
    public AuthenticationManager authenticationManager(
        TelegramWebAppDataAuthenticationProvider
telegramAuthenticationProvider,
        TelegramUpdateAuthenticationProvider
telegramUpdateAuthenticationProvider
    ) {
        return      new      ProviderManager(telegramAuthenticationProvider,
telegramUpdateAuthenticationProvider);
    }

    @Bean
    public SecurityFilterChain filterChain(
        HttpSecurity http,
        AuthenticationManager authenticationManager,
        TelegramWebAppDataAuthenticationFilter telegramAuthenticationFilter
    ) throws Exception {
        return http
            .csrf().disable()
            .addFilterBefore(telegramAuthenticationFilter,
BasicAuthenticationFilter.class)
            .authenticationManager(authenticationManager)

.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)
```

```
            .and()
            .authorizeHttpRequests()
            .antMatchers(POST, PRODUCTS).hasRole(SELLER)
            .antMatchers(PUT, PRODUCTS + "/*").hasRole(SELLER)
            .antMatchers(DELETE, PRODUCTS + "/*").hasRole(SELLER)
            .antMatchers(GET, TEST_CLIENT).permitAll()
            .anyRequest().authenticated()
            .and()
            .build();
    }

    @Bean
    public WebSecurityCustomizer webSecurityCustomizer() {
        return                          webSecurity                          ->
webSecurity.ignoring().antMatchers("/css/**/*.css", "/js/**/*.js");
    }
}

@Configuration
public class TelegramUpdateHandlerConfiguration {

    @Bean
    public TelegramUpdateHandlingChainElement buyerStartHandler(
        SecurityContextAuthorityChecker securityContextAuthorityChecker
    ) {
        return                                                          new
StartCommandHandlerForAuthority(securityContextAuthorityChecker, BUYER);
    }
}

@Configuration
public class WebAppAddressesConfiguration {

    @Bean
    public URL webAppRootUrl(@Value("${telegram.bot.web-app.address}") String
webAppRootUrlRaw)
        throws MalformedURLException {
```

```
        return new URL(webAppRootUrlRaw);
    }

    @Bean
    @Profile(Profiles.TEST_CLIENT)
    public    URL    testClientWebAppUrl(@Qualifier("webAppRootUrl")    URL
webAppRootUrl) throws MalformedURLException {
        return new URL(webAppRootUrl, TEST_CLIENT);
    }

    @Bean
    public URL webAppCartUrl(@Qualifier("webAppRootUrl") URL webAppRootUrl)
throws MalformedURLException {
        return new URL(webAppRootUrl, "/cart");
    }

    @Bean
    public     URL     webAppProductsUrl(@Qualifier("webAppRootUrl")     URL
webAppRootUrl) throws MalformedURLException {
        return new URL(webAppRootUrl, "/products");
    }
}

@NoArgsConstructor(access = AccessLevel.PRIVATE)
public class AwsConstants {

    public static final String S3_FOLDERS_SEPARATOR = "/";

    public static final String S3_PRODUCTS_FOLDER = "products";

    public static final String S3_PICTURES_FOLDER = "pictures";
}

@NoArgsConstructor(access = AccessLevel.PRIVATE)
public class Headers {

    public static final String TG_DATA_CHECK_STRING = "data_check_string";
```

```
    public static final String TG_HASH = "hash";

    public static final String TG_CHAT_ID = "chat_id";
}

@NoArgsConstructor(access = AccessLevel.PRIVATE)
public class Paths {

    public static final String PRODUCTS = "/products";

    public static final String TEST_CLIENT = "/test-client";
}

@NoArgsConstructor(access = AccessLevel.PRIVATE)
public class Profiles {

    public static final String TEST_CLIENT = "test-client";
}

@Controller
@RequestMapping(Paths.TEST_CLIENT)
@Profile(Profiles.TEST_CLIENT)
public class TestClientController {

    @GetMapping
    public ModelAndView getTestClientView() {
        return new ModelAndView("test/test-client");
    }
}

@Controller
@RequiredArgsConstructor
@RequestMapping(path = PRODUCTS)
public class ProductController {

    private final ProductService productService;
```

```
    @PostMapping
    public ResponseEntity<ProductDto> createProduct(@Valid @RequestBody
ProductCreateDto productCreateDto) {
        return
ResponseEntity.ok(productService.createProduct(productCreateDto));
    }

    @GetMapping(path = "/{productId}")
    public ResponseEntity<ProductDto> getProduct(@PathVariable Long
productId) {
        return ResponseEntity.of(productService.findProductById(productId));
    }

    @GetMapping
    public ResponseEntity<Collection<ProductDto>> getAllProducts(Pageable
pageable) {
        return
ResponseEntity.ok(productService.findAllProductsAsList(pageable));
    }

    @PutMapping("/{productId}")
    public ResponseEntity<ProductDto> updateProduct(
        @PathVariable Long productId, @Valid @RequestBody ProductUpdateDto
productUpdateDto
    ) {
        return ResponseEntity.of(productService.updateProductById(productId,
productUpdateDto));
    }

    @DeleteMapping("/{productId}")
    public ResponseEntity<ProductDto> deleteProduct(@PathVariable Long
productId) {
        return
ResponseEntity.of(productService.deleteProductById(productId));
    }
}
```