Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка POS-системи для інтернет-магазину з використанням фреймворків Flask та React

Виконав: студент IV курсу, групи СП-42
спеціальності 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

Сеньків Ю. М.
(підпис) (прізвище та ініціали)

Керівник Михалик Д. М.
(підпис) (прізвище та ініціали)

Нормоконтроль Стоянов Ю. М.
(підпис) (прізвище та ініціали)

Завідувач кафедри Петрик М. Р.
(підпис) (прізвище та ініціали)

Рецензент
(підпис) (прізвище та ініціали)

Тернопіль
2024

АНОТАЦІЯ

Кваліфікаційна робота бакалавра на тему «Розробка POS-системи для інтернет-магазину з використанням фреймворків Flask та React» написана Сеньків Юрієм Михайловичем, студентом Тернопільського національного технічного університету імені Івана Пулюя, Факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра програмної інженерії, група СП-42.

Сеньків Ю. М. Розробка POS-системи для інтернет-магазину з використанням фреймворків Flask та React : кваліфікаційна робота на здобуття освітнього ступеня бакалавр за спеціальністю «121 – Інженерія програмного забезпечення» / Юрій Михайлович Сеньків. – Тернопіль: ТНТУ, 2024. Сторінок – 55, рисунків – 31, таблиць – 4, частин – 5, додатків – 4, посилань – 24, лістингів – 1.

Метою наукової роботи є створення ефективної POS-системи для інтернет-магазину, завдання якої полягає в автоматизації управління бізнес-процесами, зокрема обробки замовлень, управління інвентарем та збереженням даних про клієнтів. Виконання цієї задачі потребувало ретельного аналізу вимог, проектування архітектури, коструювання та тестування системи.

У підсумку, виконана робота не лише досягла поставленої мети, але й продемонструвала високу якість виконання та можливістю масштабування, добавленням нового функціоналу.

У розробці були використані передові технології та інструменти: мови програмування Python та JavaScript, фреймворки Flask та React, система керування базами даних MySQL, додаткові інструменти HeidiSQL та Postman.

ABSTRACT

Bachelor's qualification work on the topic "Development of a POS system for an online store using Flask and React" was written by Senkiv Yurii Mykhailovych, a student of Ivan Puluj Ternopil National Technical University, Faculty of Computer and Information Systems and Software Engineering, group SP-42.

Development of a POS system for an online store using Flask and React: qualification work for obtaining a bachelor's degree in Software Engineering (specialization 121) / Yurii Mykhailovych Senkiv. - Ternopil: TNTU, 2024. Pages – 55, figures – 31, tables – 4, sections – 5, appendices – 4, references – 24, listings - 1.

The aim of the research work is to create an effective POS system for an online store, which aims to automate business processes management, including order processing, inventory management, and customer data storage. Achieving this goal required thorough analysis of requirements, architecture design, system construction, and testing.

In conclusion, the completed work not only achieved its set objectives but also demonstrated high-quality implementation and scalability by adding new functionality.

The development utilized advanced technologies and tools: Python and JavaScript programming languages, Flask and React frameworks, MySQL database management system, along with additional tools such as HeidiSQL and Postman.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПЗ – програмне забезпечення.

POS-система (Point of Sale system) – поєднання програмного та апаратного забезпечення для автоматизації різного роду бізнес-процесів.

Фронтенд – клієнтська частина системи.

Бекенд – серверна частина системи.

Python – мова програмування, один з методів використання якої є конструювання веб-додатків.

Flask – фреймворк для розробки веб-додатків на мові програмування Python.

JavaScript – мова програмування, яка використовується для створення динамічних та інтерактивних елементів на веб-сторінках.

React (React.js) – фреймворк JavaScript для побудови інтерфейсів користувача.

Фреймворк – набір готових компонентів, бібліотек та інструментів, які спрощують і прискорюють розробку програмного забезпечення.

HeidiSQL – інструмент для систем керування базами даних.

Postman – інструмент для тестування.

MySQL – реляційна модель системи управління базами даних.

ЗМІСТ

ВСТУП

Розвиток сучасних технологій, зокрема в сфері електронної комерції, створює нові вимоги до програмного забезпечення, яке використовується для управління бізнес-процесами. Ефективне керування продажами, інвентарем та обробка даних клієнтів є ключовими факторами успішної діяльності інтернет-магазинів. У цьому контексті важливу роль відіграють POS-системи (Point of Sale systems), які забезпечують централізацію та автоматизацію цих процесів. Вони не лише спрощують роботу менеджерів та працівників, але й підвищують точність та швидкість обслуговування клієнтів.

POS-система – комплексне рішення, яке поєднує програмне та апаратне забезпечення для автоматизації різного роду бізнес-процесів. Відповідно до теми кваліфікаційної роботи дана система розглядається лише з програмної точки зору. Це включає розробку та інтеграцію програмних компонентів, що забезпечують ефективну взаємодію між різними модулями системи, такими яку управління інвентарем, обробка замовлень та збереження даних клієнтів.

Метою роботи є детальна демонстрація процесу створення ефективної POS-системи для інтернет-магазину з використанням сучасних технологій. Використовуючи передові фреймворки Flask та React, ця система буде спроектована для забезпечення високої продуктивності та надійності.

Завдання роботи полягає в забезпеченні автоматизації ключових бізнес-процесів, включаючи обробку замовлень, управління інвентарем та збереження даних про клієнтів. Для досягнення цієї мети необхідно виконати низку завдань, описаних у змісті кваліфікаційної роботи. Серед них:

- розробка архітектури системи;
- вибір технологій;
- проектування та створення бази даних;
- конструювання серверної та клієнтської частин додатку;

– тестування готового рішення.

Об’єктом дослідження є процеси управління та обслуговування інтернет-магазинів, які включають в себе комплексні завдання з прийому та обробки замовлень, управління інвентарем, зберігання даних тощо. Ці процеси вимагають інтеграцію різноманітних інформаційних систем для забезпечення безперебійного функціонування електронної комерції, а також максимальної задоволеності покупців.

Предмет дослідження – технології та методи автоматизації бізнес-процесів за допомогою програмного забезпечення, розробленого з використанням мов програмування Python та JavaScript, фреймворків Flask та React, системи керування базами даних MySQL. Ці технології забезпечують надійність, масштабованість та гнучкість розробленої системи, дозволяючи легко адаптувати її до потреб бізнесу та вимог ринку.

Інноваційність проекту полягає у використанні передових технологій і фреймворків, які дозволяють створити високопродуктивну та надійну POS-систему для інтернет-магазину. Використання фреймворків Flask та React дозволяють створити динамічні та інтерактивні веб-додатки, що підвищують зручність й ефективність роботи користувачів.

Значущість проекту полягає в його здатності підвищити ефективність бізнес-процесів в інтернет-магазинах через автоматизацію ключових операцій. Це сприятиме зменшенню кількості помилок, оптимізації управління ресурсами та покращенню якості обслуговування клієнтів. Крім того, інтеграція сучасних технологій забезпечить високу продуктивність та надійність системи, що в кінцевому підсумку призведе до зростання конкурентоспроможності підприємства.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ Й ОПИС ЕТАПІВ РОЗРОБКИ СИСТЕМИ

1.1 Аналіз предметної області й актуальність теми

POS (Point of Sale) система – інтегроване рішення, що складається з програмного та апаратного забезпечення, яке використовується для централізації бізнес-операцій, управління продажами, інвентаризації продукції, обробки даних клієнтів тощо [1].

Хоч термін POS-система часто асоціюється з касовими апаратами, сучасні додатки такого роду здебільшого функціонують на основі програмного забезпечення. Це дає можливість використовувати їх на різних пристроях, таких як планшети, смартфони, комп’ютери.

Тема автоматизації управління бізнес-процесами за допомогою POS-систем є надзвичайно актуальною через ряд причин, серед яких:

– підвищення ефективності за рахунок автоматизації рутинних завдань;
– мінімізація людських помилок;
– оптимізація управління інвентарем;
– покращення якості обслуговування клієнтів;
– гнучкість й масштабованість.

1.2 Постановка завдання та аналіз вимог до системи

Основною задачею POS-системи є можливість легко створювати замовлення, базуючись на даних про товари та інформацію про клієнтів. Функціонал додатку повинен чітко та коректно виконувати завдання, поставлені бізнесом та відповідати функціональним та нефункціональним

вимогам. Зокрема, також повинні бути враховані вимоги до захисту даних та доступу, вимоги до інтерфейсу системи, вимоги до роботи з даними.

Після детального аналізу поставленого завдання було виділено такі функціональні вимоги:

– наявність зручної навігації у системі;
– можливість створення нового продукту;
– можливість редагування продуктів;
– можливість видалення продуктів;
– можливість додавання інформації про нового клієнта;
– можливість редагування даних клієнта;
– можливість видалення інформації про клієнта;
– можливість створення замовлення;
– можливість редагування замовлення;
– можливість видалення замовлення;
– коректне й роздільне відображення кожної з сутностей системи: продуктів, клієнтів, замовлення;
– можливість фільтрації даних на сторінках замовлень, продуктів та клієнтів.

Нефункціональні вимоги:

– Продуктивність. Система повинна працювати швидко, без помилок.
– Надійність. Система повинна функціонувати безперебійно та бути доступною протягом доби.
– Зручність використання. Система повинна володіти адаптивним, інтуїтивно зрозумілим інтерфейсом.

Вимоги до захисту та доступу:

– система повинна обмежувати доступ до своїх ресурсів, дозволяючи підключення лише з IP-адрес, що знаходяться у білому списку. Всі спроби доступу з IP-адрес, які не включені до білого списку повинні бути

автоматично заблоковані. Білий список має бути керованим, з можливістю додавання, видалення або оновлення IP-адрес адміністратором системи.

Вимоги до інтерфейсу системи:

– інтерфейс має бути адаптивним, автоматично підлаштовуватись під різні розміри екрану, забезпечуючи належне відображення на будь-якому пристрої;

– дизайн повинен бути мінімалістичним з використанням монотонних кольорів та чітким текстом, який добре контрастує з фоном;

– інтерфейс повинен бути простим і зручним для користувача.

Вимоги до роботи з даними:

– Система повинна гарантувати, що всі дані є цілісними. Будь-яка операція з даними, включаючи створення, оновлення, видалення та зчитування, повинна зберігати внутрішню узгодженість даних.

– Всі операції, які змінюють стан даних, повинні виконуватися в рамках транзакції, що забезпечують атомарність, узгодженість, ізоляцію та довговічність (ACID-принципи).

– Всі дані, введені користувачами повинні проходити через систему валідації для перевірки коректності та відповідності встановленим правилам та форматам. Некоректні дані повинні бути відхилені з відповідним повідомленням.

1.3 Основні етапи розробки системи

Після детального аналізу предметної області, одним з найголовніших аспектів створення системи є важливість визначення правильної моделі розробки додатку. Серед основних методологій розробки ПЗ вирізняють каскадну, інкрементну, ітеративну, гнучку, спіральну та V-подібну моделі,

кожна з яких має свої переваги та недоліки і обирається індивідуально під кожен проект [2].

Варто зазначити, що незалежно від обраної методології розробки ПЗ, створення додатку безсуперечно повинне включати наступні етапи:

- Аналіз вимог. Важливо детально описати всі вимоги, що ставляться до проекту задля розуміння реалізації поставлених цілей.
- Проектування архітектури додатку. Цей етап повинен включати розробку структури системи, визначення її основних компонентів, їх взаємодії та розподіл функцій між ними.
- Розробка та програмування. Реалізація серверної логіки, яка відповідає за обробку запитів від клієнта, управління базою даних, забезпечення безпеки додатку тощо, розробка інтерфейсу користувача, який буде зручним та ергономічним для працівників.
- Тестування. Проведення тестів для перевірки правильності та надійності системи, виявлення та виправлення допущених помилок.
- Впровадження та підтримка. Система повинна бути розгорнутою і налагодженою для використання користувачем.

Після дослідження різних методологій розробки програмного забезпечення було вибрано каскадну модель (ориг. Waterfall Model). Це рішення було прийняте завдяки основній особливості даного підходу – структурованості і послідовності, що у свою чергу забезпечує чіткий розподіл етапів розробки проекту від аналізу вимог до впровадження та підтримки. Даний підхід особливо підходить для цієї системи, оскільки її вимоги є добре визначеними, що дозволяє мінімізувати можливі ризики і гарантувати високу якість кінцевого продукту.

2 ПРОЕКТУВАННЯ АРХІТЕКТУРИ POS-СИСТЕМИ

2.1 Пошук акторів та варіантів використання

Ретельно проаналізувавши предметну область системи та визначивши всі її вимоги, можна виділити єдиного головного актора – Користувач. Користувач є центральною фігурою системи, від взаємодії якого залежить успішне виконання бізнес-процесів. Для даного актора доступні всі варіанти використання, наведені на діаграмі, зображеній на рисунку 2.1.

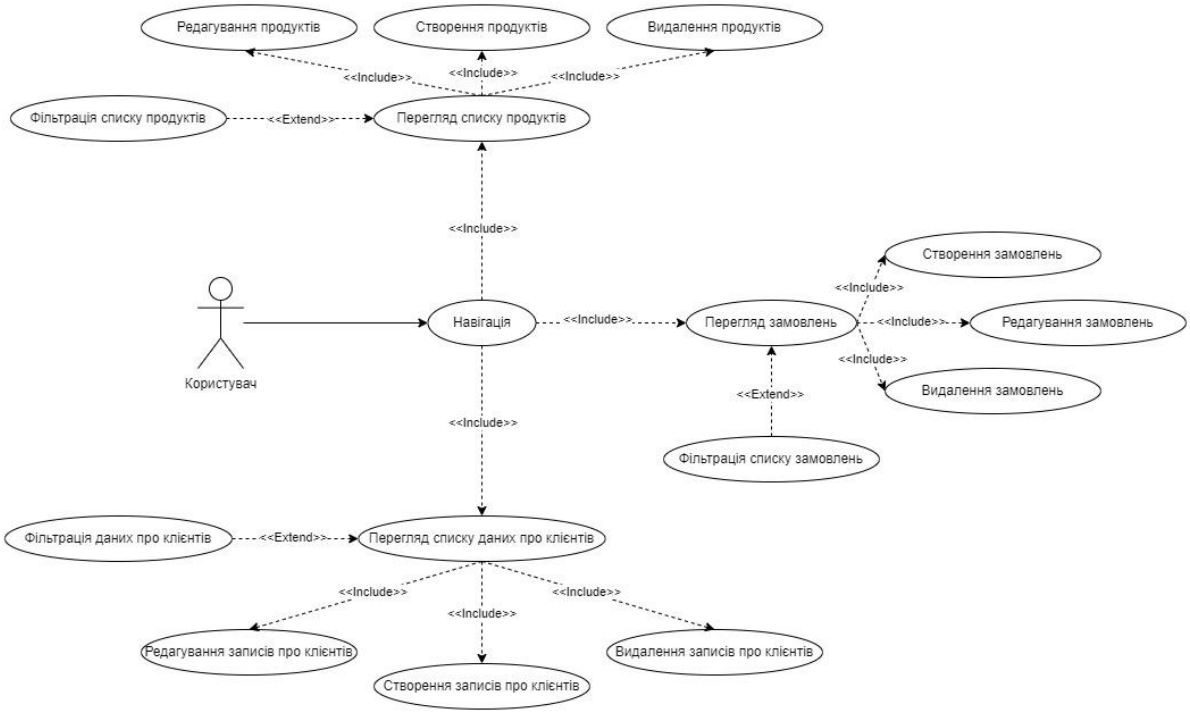

Рисунок 2.1 – Діаграма варіантів використання

2.2 Опис варіантів використання

Проаналізувавши діаграму варіантів використання, зображену на рисунку 2.1, подальший крок передбачає детальне розглядання кожного ВВ системи. Для кращого розуміння кожного сценарію взаємодії кожен варіант

буде розглянуто окремо. Цей аналіз дозволить глибше зрозуміти різноманітність можливостей використання системи та їх вплив на її функціональність.

Навігація – даний варіант використання передбачає надання можливості користувачу зручно та ефективно переміщатися по різних частинах системи: сторінки зі списком продуктів, замовлень та інформації про клієнтів.

Перегляд списків продуктів, замовлень та інформації про клієнтів – три різні варіанти використання, схожі за функціональними можливостями, представлення кожної сутності системи на окремій сторінці, однак означають різні властивості елементів додатку.

Створення, редагування та видалення продукту, замовлення чи запису з інформацією про клієнта – набір базових операцій для кожної з сутностей системи, що забезпечують функціонування додатку та реалізовують бізнес-вимоги.

Фільтрація даних на сторінках списків продуктів, замовлень та інформації про клієнтів – варіанти використання, які реалізовують зручність у пошуку потрібної інформації на сторінці потрібного елементу системи.

2.3 Архітектура системи

POS-система для інтернет-магазину побудована на основі клієнт-серверної архітектури, що дозволяє забезпечити розподіл функціональності між сервером та клієнтськими додатками. Ця архітектура обрана через низку переваг, які роблять її найкращим рішенням для такого типу додатків.

Серед основних переваг:

– Масштабованість. Клієнт-серверна архітектура дозволяє легко масштабувати систему, додаючи нові клієнтські додатки або збільшуючи потужність серверів. Це важливо для інтернет-магазину, який може швидко зростати та збільшувати обсяг оперованих даних;

– Централізоване управління даними. Всі дані зберігаються на сервері, що забезпечує централізоване управління, оновлення та безпеку даних. Це знижує ризик втрати даних і полегшує їхнє резервне копіювання та відновлення;

– Безпека. Серверна частина може реалізовувати різні заходи безпеки, включаючи шифрування даних, аутентифікацію користувачів та контроль доступу, що підвищує загальну безпеку системи. Клієнти взаємодіють з сервером через захищені протоколи, що мінімізує ризики перехоплення або маніпуляції даними;

– Розділення обов’язків. Клієнтська частина відповідає за відображення інтерфейсу користувача, тоді як сервер обробляє запити, виконує бізнес-логіку та управління даними;

– Гнучкість і адаптивність. Клієнт-серверна архітектура дозволяє використовувати різні клієнтські платформи (веб, мобільні додатки) без зміни серверної логіки. Це забезпечує гнучкість у виборі та розширенні користувацького інтерфейсу, адаптуючи систему до потреб різних груп користувачів;

– Полегшене тестування та обслуговування. Зрозуміле розділення між клієнтською та серверною частинами спрощує тестування та обслуговування системи. Помилки можуть бути швидше локалізовані та виправлені, що підвищує надійність та стабільність роботи системи [3].

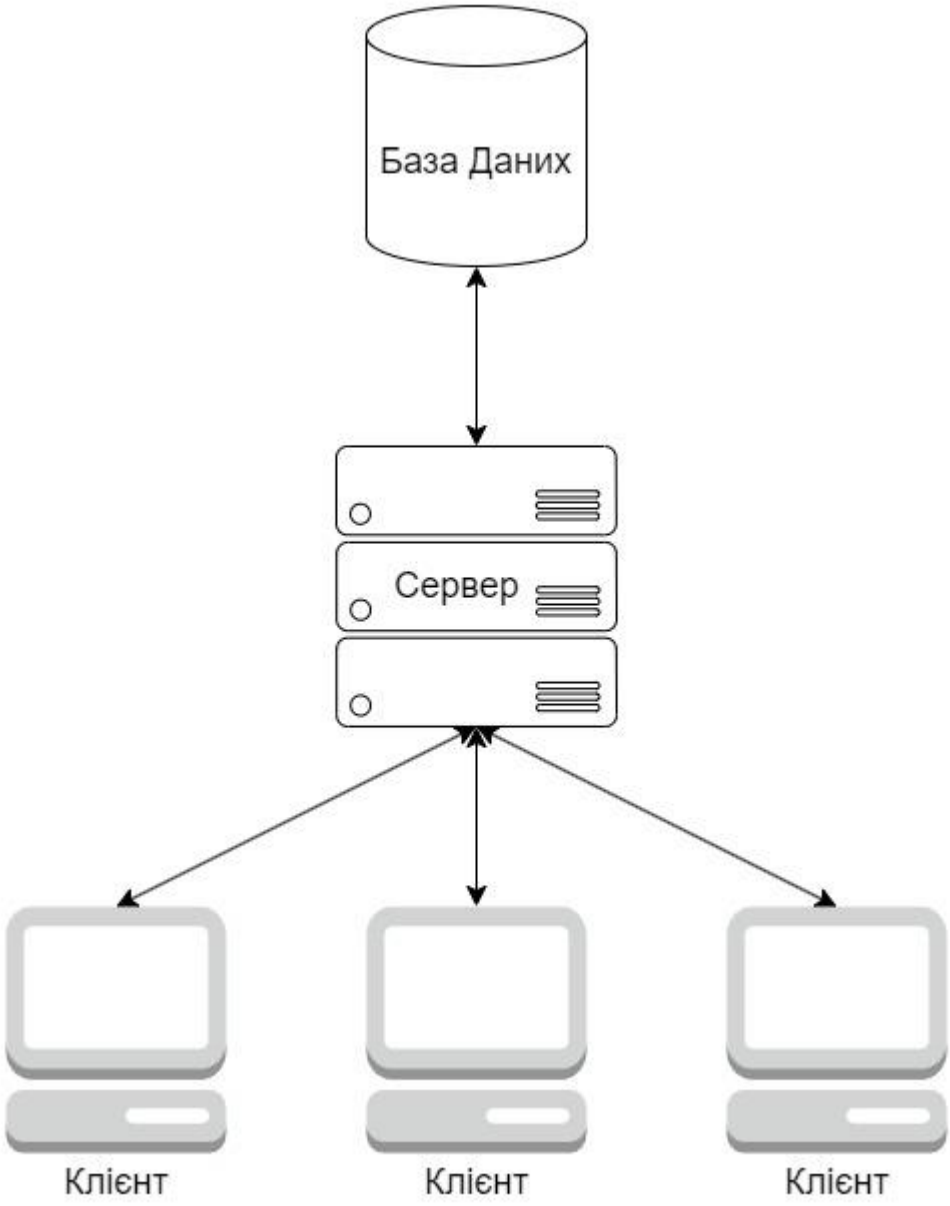

Рисунок 2.2 – Клієнт-серверна архітектура

Завдяки клієнт-серверній архітектурі, POS-система здатна ефективно обробляти численні запити, забезпечувати високий рівень безпеки та надавати зручний інтерфейс користувачам. Таких підхід дозволяє створити надійне, масштабоване та легке у підтримці рішення, яке відповідає вимогам сучасного інтернет-магазину.

2.4 Вибір технологій та інструментальних засобів розробки системи

Вибір технологій та інструментальних засобів для розробки системи є критичним етапом, який визначає ефективність і продуктивність процесу створення програмного продукту, а також його кінцеву якість і здатність відповідати поставленим вимогам.

Зважаючи на визначену архітектуру системи, підбір технологій та інструментальних засобів був розбитий на 3 етапи окремо для кожного ланки системи: розробка серверної частини, розробка клієнтської частини, створення бази даних.

Для серверної частини додатку було обрано мову програмування Python й фреймворк Flask. Таке рішення було зумовлене простотою синтаксису Python й великою кількістю доступних бібліотек, що полегшують розробку веб-додатків [4]. Flask в свою чергу був обраний через легкість і гнучкість його налаштування й можливість розширення його функціональних можливостей за рахунок додаткових модулів [6].

Для розробки клієнтської частини було обрано мову програмування JavaScript і фреймворк React. React дозволяє створювати швидкі та інтерактивні інтерфейси користувача завдяки своїй компонентній архітектурі [15, 16].

MySQL була обрана як основна система керування базами даних завдяки своїй надійності, високій продуктивності та здатності обробляти великі обсяги даних [8]. MySQL також підтримує потужні засоби для роботи з транзакціями, що важливо задля збереження цілісності даних у системі.

Для написання коду та управління проектом використовуються інтегроване середовище розробки Visual Studio Code, яке забезпечує зручний інтерфейс й підтримку різних плагінів [17].

В ролі інструменту для управління базами даних було обрано доступне програмне забезпечення HeidiSQL, яке вирізняється зручним графічним

інтерфейсом, широким функціоналом для адміністрування БД. Крім того, HeidiSQL надає можливість легко створювати та редагувати таблиці, виконувати складні SQL-запити, експортувати та імпортувати дані, а також забезпечує високу швидкість роботи та стабільність, що значно спрощує процес розробки та обслуговування баз даних [9].

Після розгляду різних інструментів тестування було вирішено використовувати застосунок «Postman» як основний засіб для тестування серверної частини POS-системи. Такий вибір був обґрунтований наступними причинами:

- зручний інтерфейс;
- можливість автоматизації тестів;
- підтримка різних типів запитів;
- можливість перевірки відповідей [10].

2.5 Визначення класів системи

На основі аналізу системи, її вимог і складових було визначено класи додатку й продемонстровано у діаграмі класів (рис. 2.2) [12].

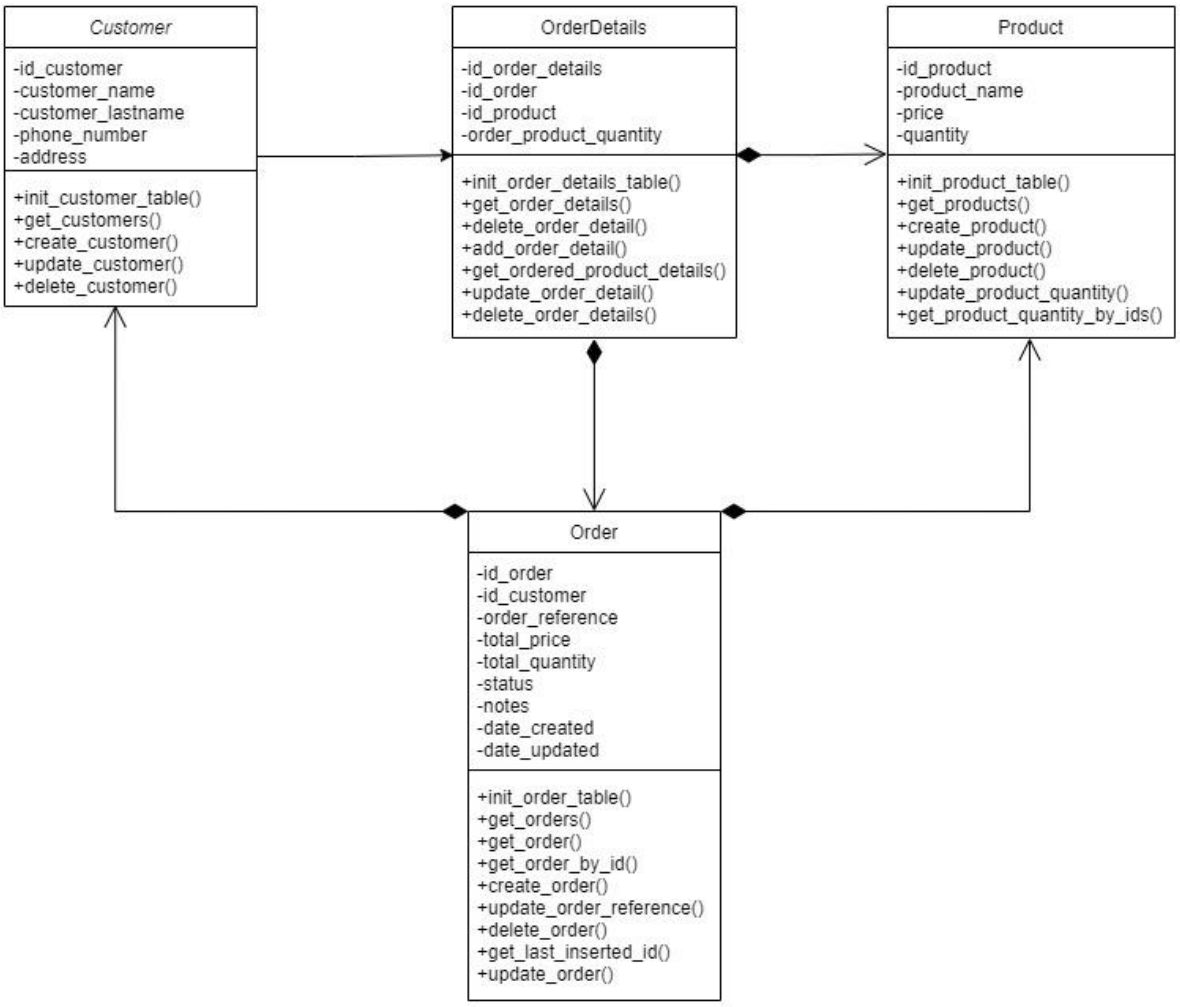

Рисунок 2.3 – Діаграма класів системи

З діаграми видно, що кожен описаний клас має свої набори атрибутів та методів.

Основними класами системи є Product та Customer, вони є незалежними один між одним. Між класами Product та Order й Customer та Order існує зв’язок композиція, оскільки головні класи є основною складовою класу Order та об’єкт останнього не може існувати без визначених об’єктів основних класів.

Клас OrderDetails є доповненням класу Order та Product, об’єкт якого також не може існувати без визначених об’єктів останніх двох класів.

2.6 Керування даними

База даних є невід’ємною складовою будь-якої POS-системи. Як зазначалось раніше під час вибору технологій та інструментальних засобів, на основі аналізу вимог було вирішено використовувати реляційну модель бази даних, а саме MySQL, за рахунок її переваги у зручності користування, швидкості роботи й розгортання та масштабованості, особливо у даній системі.

Оскільки реляційна модель БД передбачає збереження даних у табличному форматі, зважаючи на вимоги інформаційної системи, було визначено ряд структур, які окреслюють кожну з сутностей системи [13].

Таблиця «product» - відповідає за збереження інформації про продукти інтернет-магазину.

Таблиця 2.1 – Таблиця «product»

Назва поля	Тип поля	Ключ	Додаткові параметри
id_product	INT(10)	PRIMARY KEY	AUTO_INCREMENT
product_name	VARCHAR(255)	-	-
price	DECIMAL(12, 2)	-	-
quantity	INT(10)	-	-

Таблиця «customer» - відповідає за збереження інформації про клієнтів інтернет-магазину.

Таблиця 2.2 – Таблиця «customer»

Назва поля	Тип поля	Ключ	Додаткові параметри
id_customer	INT	PRIMARY KEY	AUTO_INCREMENT

Продовження таблиці 2.2

customer_name	VARCHAR(255)	-	-
customer_lastname	VARCHAR(255)	-	-
phone_number	INT(10)	-	-
address	VARCHAR(255)	-	-

Таблиця «orders» - відповідає за збереження інформації про замовлення інтернет-магазину.

Таблиця 2.3 – Таблиця «orders»

Назва поля	Тип поля	Ключ	Додаткові параметри
id_order	INT(10)	PRIMARY KEY	AUTO_INCREMENT
id_customer	INT(10)	FOREIGN KEY	-
order_reference	VARCHAR(255)	-	-
total_price	DECIMAL(10, 2)	-	-
total_quantity	DECIMAL(10, 2)	-	-
status	VARCHAR(255)	-	-
notes	VARCHAR(255)	-	-
date_created	DATETIME	-	-
date_updated	DATETIME	-	-

Таблиця «order_details» - слугує доповненням таблиці «orders» та відповідає за збереження розширеної інформації про замовлення інтернет-магазину.

Таблиця 2.4 – Таблиця «order_details»

Назва поля	Тип поля	Ключ	Додаткові параметри
id_order_details	INT(10)	PRIMARY KEY	AUTO_INCREMENT
id_order	INT(10)	FOREIGN KEY	-
id_product	INT(10)	FOREIGN KEY	-
order_product_quantity	INT(10)	-	-

3 РОЗРОБКА POS-СИСТЕМИ

3.1 Створення бази даних та таблиць

Процес створення та наповнення бази даних є ключовим етапом у розробці будь-якої інформаційної системи, оскільки від його правильного виконання залежить ефективність зберігання, обробки та доступу до інформації.

Після успішного проектування бази даних та визначення її структури, було виконане її безпосереднє створення. За допомогою інструменту HeidiSQL було встановлене з’єднання з локальним сервером, після чого ініціалізовано створення пустої БД «royalbee» через виконання запиту, представленого в лістингу 3.1.

Лістинг 3.1.1 – Запит на створення бази даних «royalbee»

```
CREATE DATABASE `royalbee`;
```

Оскільки структура бази даних є визначеною, було виконано SQL-запити на створення відповідних таблиць, лістинг який наведений на рисунках 3.1.1, 3.1.2, 3.1.3 та 3.1.4.

```
1 CREATE TABLE `product` (
2         `id_product` INT(10) NOT NULL AUTO_INCREMENT,
3         `product_name` VARCHAR(255) NULL DEFAULT NULL,
4         `price` DECIMAL(10,2) NULL DEFAULT NULL,
5         `quantity` INT(10) NULL DEFAULT NULL,
6         PRIMARY KEY (`id_product`)
7 );
```

Рисунок 3.1.1 – Запит на створення таблиці «product»

```sql
CREATE TABLE `customer` (
        `id_customer` INT(10) NOT NULL AUTO_INCREMENT,
        `customer_name` VARCHAR(255) NULL DEFAULT NULL,
        `customer_lastname` VARCHAR(255) NULL DEFAULT NULL,
        `phone_number` INT(10) NULL DEFAULT NULL,
        `address` VARCHAR(255) NULL DEFAULT NULL,
        PRIMARY KEY (`id_customer`)
);

```

Рисунок 3.1.2 – Запит на створення таблиці «customer»

```sql
CREATE TABLE `orders` (
        `id_order` INT(10) NOT NULL AUTO_INCREMENT,
        `id_customer` INT(10) NULL DEFAULT NULL,
        `order_reference` VARCHAR(255) NULL DEFAULT NULL,
        `total_price` DECIMAL(10,2) NULL DEFAULT '0.00',
        `total_quantity` INT(10) NULL DEFAULT '0',
        `status` VARCHAR(255) NULL DEFAULT 'in_progress',
        `notes` VARCHAR(255) NULL DEFAULT NULL,
        `date_created` DATETIME NULL DEFAULT CURRENT_TIMESTAMP,
        `date_updated` DATETIME NULL DEFAULT CURRENT_TIMESTAMP,
        PRIMARY KEY (`id_order`),
        CONSTRAINT `FOREIGN KEY` FOREIGN KEY (`id_customer`)
        REFERENCES `customer` (`id_customer`) ON UPDATE NO ACTION ON DELETE NO ACTION
);

```

Рисунок 3.1.3 – Запит на створення таблиці «orders»

```sql
CREATE TABLE `order_details` (
        `id_order_details` INT(10) NOT NULL AUTO_INCREMENT,
        `id_order` INT(10) NULL DEFAULT NULL,
        `id_product` INT(10) NULL DEFAULT NULL,
        `order_product_quantity` INT(10) NULL DEFAULT NULL,
        PRIMARY KEY (`id_order_details`),
        CONSTRAINT `FOREIGN KEY` FOREIGN KEY (`id_order`)
        REFERENCES `orders` (`id_order`) ON UPDATE NO ACTION ON DELETE NO ACTION,
        CONSTRAINT `FOREIGN KEY 2 ` FOREIGN KEY (`id_product`)
        REFERENCES `product` (`id_product`) ON UPDATE NO ACTION ON DELETE NO ACTION
);

```

Рисунок 3.1.4 – Запит на створення таблиці «order_details»

Після успішно виконаних операцій було отримано готову для роботи базу даних з усіма потрібними таблицями. Результат виконаної роботи зображений на рисунку 3.1.5.

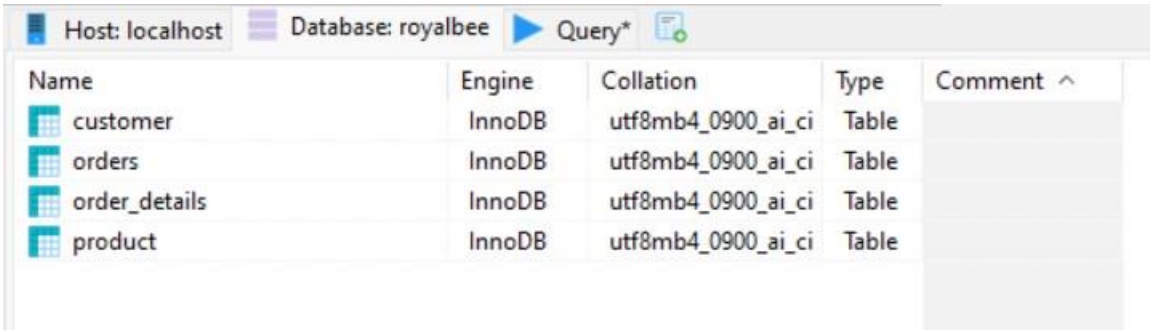

Рисунок 3.1.5 – Створена база даних «royalbee»

3.2 Організація середовища розробки серверної частини

Як зазначалось раніше, для розробки серверної частини була використана мова програмування Python, зокрема її фреймворк Flask.

Оскільки мова програмування Python для роботи у проекті вимагає наявність окремого віртуального середовища, на початку роботи у гілці серверної частини додатку було ініціалізовано створення віртуального середовища за допомогою команди: «python -m myenv /project/flask_server» [5, 14].

Після успішного виконання команди, у терміналі Visual Studio Code було активовано віртуальне середовище Python відкриттям файлу activate, що знаходиться за шляхом ./myenv/Scripts/activate.

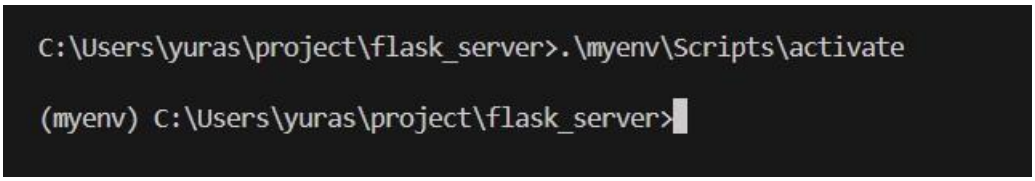

Рисунок 3.2.1 – Активація віртуального середовища у терміналі Visual Studio Code

Далі була здійснена інсталяція фреймворку Flask через відкрите віртуальне середовище за допомогою команди «pip install Flask» [7].

```
(myenv) C:\Users\yuras\project\flask_server>pip install flask
```

Рисунок 3.2.2 – Інсталяція фреймворку Flask

По завершенню інсталяції фреймворку було створено файл server.py для конструювання серверної частини.

3.3 Етапи розробки серверної частини

При конструюванні серверної частини було визначено кілька ключових етапів:

– Створення базового додатку з використанням Flask.

– Підключення бази даних.

– Створення класів під кожну сутність системи з потрібним набором атрибутів та методів.

– Розробка маршрутизації та функціональної частини серверної частини системи.

Такий підхід дозволяє послідовно реалізовувати функціональність системи задля досягнення поставлених цілей.

3.4 Конструювання серверної частини

Для реалізації поставлених цілей першочергово було створено базовий додаток з використанням Flask, лістинг якого зазначений на рисунку 3.4.1 [7].

```python
from flask import Flask

app = Flask(__name__)

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

```

Рисунок 3.4.1 – Базовий додаток з використанням Flask

Далі було додано логіку з’єднання із базою даних, зображену на рисунку 3.4.2.

```python
from DatabaseConnection import DatabaseConnection

db = DatabaseConnection(host='****', user='****', password='****', database='royalbee')
```

Рисунок 3.4.2 – Реалізація з’єднання з базою даних

Для кожної сутності системи було створено класи й розділено по окремих файлах з назвами, що відповідають назвам класів. Далі класи було підключено до файлу server.py та створено їх стандартні об’єкти для подальшого використання.

```
7   from product import Product
8   from customer import Customer
9   from order import Order
10  from order_details import OrderDetails
11
12
13
14  product_obj = Product('', '', '', '')
15  order_details_obj = OrderDetails('', '', '', '')
16  order_obj = Order('', '', '', '', '', '', '', '', '')
17  customer_obj = Customer('', '', '', '', '')
18
```

Рисунок 3.4.3 – Підключення класів до головного файлу серверної частини та створення стандартних об’єктів

Фінальним етапом конструювання серверної частини інформаційної системи стала розробка методів маршрутизації та функціональної складової додатку.

На основі аналізу функціональних вимог було визначено основні функції та маршрути системи, наведені на рисунку 3.4.1.

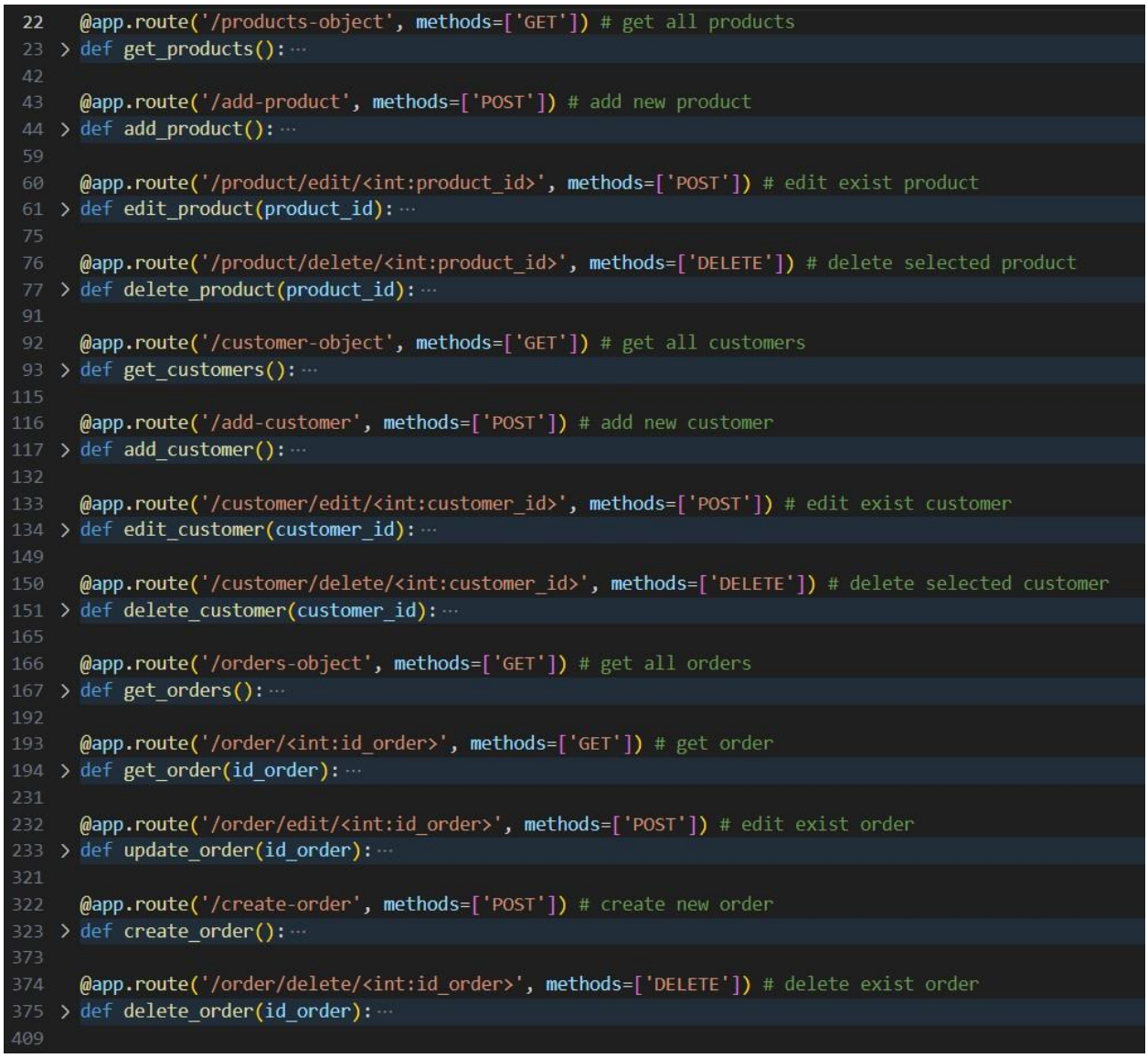

Рисунок 3.4.1 – Основні функції та маршрути серверної частини системи

Кожна із зображених на рисунку 3.4.1 функцій реалізовує окремі варіанти використання системи. Керування даними відбувається за рахунок прямих запитів до бази даних, кожен з яких є частиною транзакції, що реалізовується методами db.connection.begin() та db.connection.commit().

Однією з основних можливостей системи є функція, наведена на рисунку 3.4.4, що реалізовує логіку створення замовлення. Її особливість полягає у одночасній роботі з даними формату JSON, перевизначенні атрибутів стандартних об’єктів класів сутностей, динамічному формуванні значення атрибуту «order_reference» та закритті з’єднання з базою даних після завершення роботи функції, незалежно від результату її виконання.

```python
@app.route('/create-order', methods=['POST']) # create new order
def create_order():
    try:
        data = request.json
        db.connection.begin()
        cursor = db.connection.cursor()

        order_obj.id_customer = data.get('id_customer')
        order_obj.total_price = data.get('total_price')
        order_obj.total_quantity = data.get('total_quantity')
        order_obj.status = data.get('status')
        order_obj.notes = data.get('notes')
        # create order
        co_query, co_params = Order.create_order(order_obj)
        cursor.execute(co_query, co_params)
        # generate order reference
        cursor.execute(Order.get_last_inserted_id())
        result = cursor.fetchone()
        order_id = result[0]
        order_increment = ''

        for i in range(7 - len(str(order_id))):
            order_increment += '0'

        order_obj.order_reference = 'RB' + order_increment + str(order_id)
        order_obj.id_order = order_id

        uor_query, uor_params = Order.update_order_reference(order_obj)
        cursor.execute(uor_query, uor_params)

        products_data = data.get('productData')
        for product in products_data:
            # add order detail
            order_details_obj.id_order = order_id
            order_details_obj.id_product = product[0]
            order_details_obj.order_product_quantity = product[1]
            aod_query, aod_params = OrderDetails.add_order_detail(order_details_obj)
            cursor.execute(aod_query, aod_params)

            # update product quantity on order create
            product_obj.id_product = product[0]
            product_obj.quantity = product[1]
            upq_query, upq_params = Product.update_product_quantity_on_order_create(product_obj)
            cursor.execute(upq_query, upq_params)

        db.connection.commit()
        return jsonify({'message': "Order has been successfully added!"})
    except Exception as e:
        return jsonify({'error:': 'Failed to fetch order details!'})
    finally:
        db.connection.close()

```

Рисунок 3.4.4 – Реалізація функції створення замовлення

3.5 Етапи розробки клієнтської частини системи

Як і у випадку з бекендом, конструювання клієнтської частини системи було поділено на декілька ключових етапів:

– Організація робочого середовища, інсталяція та запуск базового додатку на React.

– Підключення клієнтської частини до серверної.

– Розробка навігації системи.

– Конструювання сторінок та окремих компонентів додатку.

Такий підхід до розподілу задач забезпечив більш ефективну та послідовну роботу над проектом

3.6 Організація робочого середовища, інсталяція та запуск базового додатку на React.

Розробка клієнтської частини системи реалізована з використанням мови програмування JavaScript й фреймворку React.

Підготовка робочого середовища включає інсталяцію базового додатку React до папки проекту. Після успішного виконання команди «npx create-react-app myapp» у терміналі Visual Studio Code у кореневій папці системи з’являються усі необхідні для роботи файли, зображені на рисунку 3.6.1.

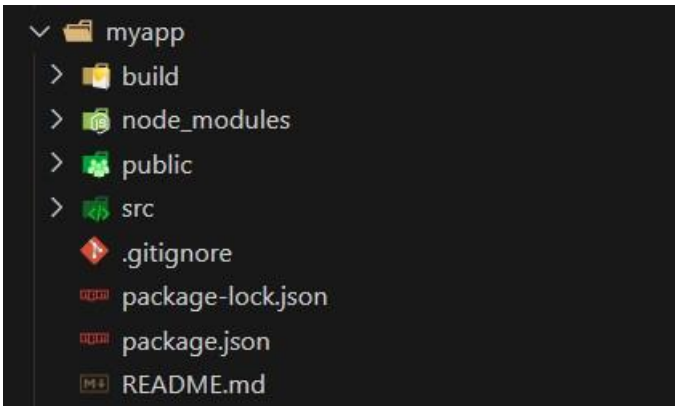

Рисунок 3.6.1 – Файлова структура базового додатку React

Для запуску інстальованого додатку було виконано команду «npm start», результат якої зображено на рисунку 3.6.2 [16].

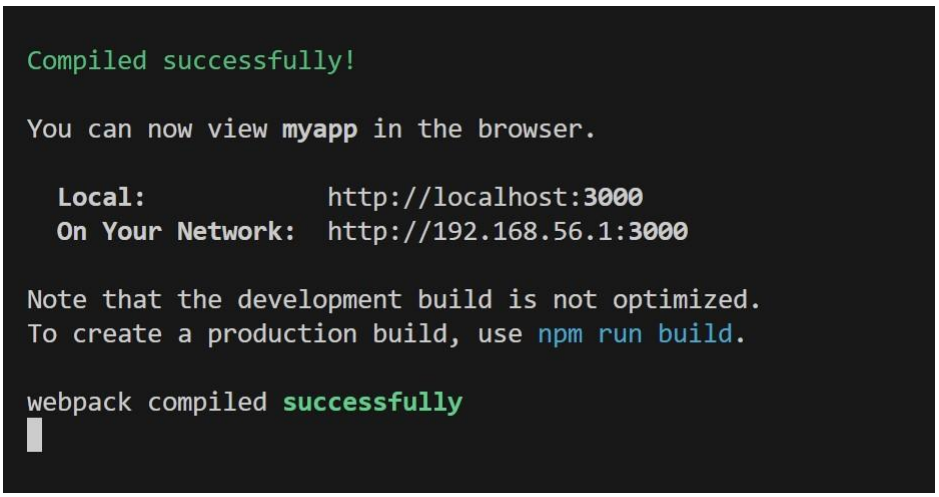

Рисунок 3.6.2 – Запуск додатку на React

3.7 Підключення клієнтської частини додатку до серверної

Оскільки за замовчуванням, інтеграція додатків на Flask та React не є передбачуваною. Для того, щоб підключити клієнтську частину системи до серверної, у файлі package.json було визначено параметр «proxy» й надано йому значення «http://localhost:5000».

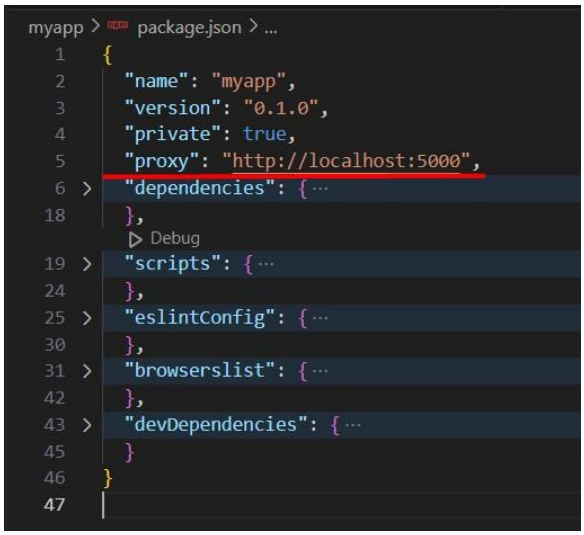

Рисунок 3.7.1 – Підключення клієнтської частини до серверної

Завдяки таким змінам стало можливо напряму звертатись до API серверної частини з клієнтської частини системи, що значно спростило процес обміну даними між фронтендом і бекендом й забезпечило більш зручний і швидкий робочий процес при розробці проекту.

3.8 Розробка навігації системи

На основі аналізу вимог до системи, навігаційну систему було поділено на 4 маршрути:

– «Home» - стартова пуста сторінка системи

– «Orders» - сторінка зі списком замовлень інтернет-магазину й іншим потрібним функціоналом.

– «Products» - сторінка з списком продуктів інтернет-магазину та пов’язаними функціями.

– «Customers» - сторінка з списком клієнтів інтернет-магазину та відповідними функціями.

Для реалізації такої системи було використано бібліотеку React Router, яка дозволяє легко визначити маршрути для різних сторінок додатку.

Навігаційну структуру, зображену на рисунку 3.8.1, було визначено в основному файлі додатку App.js, де кожен маршрут відповідає певному компоненту, що відображає відповідну сторінку. Це забезпечує чітку та зрозумілу систему навігації, що полегшує роботу користувачів з додатком.

myapp > src > JS App.js > ...

```
import "./App.css"
import { Link, Route, Routes } from "react-router-dom";
import { Orders } from "./pages/Orders/orders";
import { Home } from "./pages/home";
import { Products } from "./pages/Products/products";
import { Customers } from "./pages/Customers/customers";
import { Order } from "./pages/Order/order"
import { CreateOrder } from "./pages/CreateOrder/createOrder"

function App() {

  return (
    <>
      <div className="navigation">
        <nav>
          <ul>
            <li><Link to="/">Home</Link></li>
            <li><Link to="/orders">Orders</Link></li>
            <li><Link to="/products">Products</Link></li>
            <li><Link to="/customers">Customers</Link></li>
          </ul>
        </nav>
      </div>

      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/orders" element={<Orders />} />
        <Route path="/order/:id_order" element={<Order />} />
        <Route path="/order/create-new-order" element={<CreateOrder />} />
        <Route path="/products" element={<Products />} />
        <Route path="/customers" element={<Customers />} />
      </Routes>
    </>
  );
}

export default App;
```

Рисунок 3.8.1 – Реалізація навігаційної системи додатку

3.9 Конструювання сторінок та компонентів клієнтської частини додатку

Для конструювання окремих сторінок, що вимагається структурою проекту, у репозиторії клієнтської частини системи було створено папку «pages». У цій папці розміщено файли сторінок формату «.js» з назвами, що описують їх суть. Після цього було розроблено розмітку кожної сторінки, інтегровано функціонал, передбачений вимогами проекту, й додано спливаючі вікна, реалізовані з використанням бібліотеки «sweetalert», для сповіщення користувача про успішність чи невдачу виконаної операції.

Одним із найважливіших завдань системи є створення замовлення. Ця частина додатку є найбільш навантаженою, оскільки для її функціонування використовуються всі інші сутності інформаційної системи. Під час створення замовлення здійснюється калькуляція цін та кількості продуктів, а також динамічно формується унікальний ідентифікатор замовлення. Для полегшення цього навантаження, було вирішено реалізувати функціонал створення замовлення на новій сторінці «/order/create-new-order».

Першочерговим етапом створення замовлення є вибір існуючого або створення нового клієнта. Така реалізація передбачає використання вибірки масиву всіх існуючих клієнтів магазину за допомогою функції, зазначеної на рисунку 3.9.1.

```
const fetchData = async () => {
    try {
        const customerResponse = await fetch("/customer-object");
        const customerData = await customerResponse.json();

        setCustomers(customerData);
        setLoading(false);
    } catch (error) {
        console.error("Error fetching data:", error);
    }
};
```

Рисунок 3.9.1 – Вибірка масиву існуючих клієнтів

У разі необхідності створення нового клієнта на сторінці передбачена кнопка, після натискання на яку викликається компонент «AddNewCustomer», реалізація якого зображена на рисунку 3.9.2.

```
{addCustomerButtonVisible && !nextButtonClicked && (
    <button className="add-new-customer-button" onClick={openAddCustomerModal}>
        Add New Customer
    </button>
)}
```

Рисунок 3.9.2 – Виклик компоненту «AddNewCustomer» при натисканні на відповідну кнопку.

За умови якщо клієнта було обрано, на сторінці створення замовлення з’являється кнопка «Next», натиск на яку передбачає виклик компоненту «CreateOrderModal». Даний компонент містить нові набори змінних для заповнення користувачем такі як: статус замовлення, нотатки, список продуктів, які необхідно додати до замовлення. Поля сумарної ціни та кількості продуктів у замовленні розраховуються автоматично на стороні клієнтської частини системи. Поле статусу замовлення є розкривним списком із вже визначеними значеннями опцій (рис. 3.9.3).

```
<div className="order-status">
    <label htmlFor="order-status">Order Status</label>
    <select id="order-status" name="order-status" onChange={(e) => setStatus(e.target.options[e.target.selectedIndex].innerText)}>
        <option value="">Select Status</option>
        <option value="in-progress">In Progress</option>
        <option value="shipped">Shipped</option>
        <option value="completed">Completed</option>
    </select>
</div>
```

Рисунок 3.9.3 – Реалізація поля статусу замовлення

Поле вибору продуктів є також розкривним списком, кожна опція якого є елементом масиву відфільтрованих даних всіх створених продуктів у системі (рис. 3.9.4).

```
<div className="add-product-to-order">
    {!isDropdownOpen ? (
        <button className="add-product-to-order-button" onClick={handleAddProductToOrderClick}>Add Product To Order</button>
    ) : (
        <div>
            <select value={selectedProduct} onChange={handleProductDropdownChange}>
                <option value="">Select a product</option>
                {filteredProducts.map(product => (
                    <option key={product.id_product} value={product.product_name}>{product.product_name}</option>
                ))}
            </select>
            <button className="add-product-to-order-button" onClick={handleAddProductToOrder}>Add</button>
        </div>
    )}
</div>
```

Рисунок 3.9.4 – Реалізація додавання продуктів до замовлення

Фільтрація продуктів реалізована таким чином, що у списку відсутні продукти з кількістю 0 або ті, які вже додані до замовлення (рис. 3.9.5).

```
const applyFilter = () => {
    const filteredProductData = products.filter(product => {
        const isAlreadyAdded = orderProductDetails.some(orderProduct => orderProduct.product_name === product.product_name);
        return !isAlreadyAdded && product.quantity > 0;
    });
    setFilteredProducts(filteredProductData);
};
```

Рисунок 3.9.5 – Реалізація фільтрації продуктів при додаванні їх до замовлення

Завершальним етапом створення замовлення є натискання кнопки «Create», яка ініціалізує виконання функції «handleCreateOrder», суть якої полягає у надсиланні всіх необхідних даних до серверної частини системи й обробку результату операції. Реалізація цієї функції зображена на рисунку 3.9.6.

```
const handleCreateOrder = async () => {
    let productsToAdd = orderProductDetails.map(product => [product.id_product, product.order_product_quantity]);
    const dataToAdd = {
        id_customer: customerToAdd.customerId,
        productData: productsToAdd,
        total_price: parseFloat(totalPrice),
        total_quantity: totalQuantity,
        notes: notes,
        status: status
    };

    try {
        if (dataToAdd.status === '') {
            Swal.fire({···
            })
        } else if (dataToAdd.productData.length === 0) {
            Swal.fire({···
            })
        } else {
            const response = await fetch(`/create-order`, {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json'
                },
                body: JSON.stringify(dataToAdd)
            });

            if (response.ok) {···
            } else {···
            }
        }

    } catch (error) {
        console.error('Error creating order:', error);
    }
};
```

Рисунок 3.9.6 – Реалізація функції «handleCreateOrder»

4 ТЕСТУВАННЯ ТА ВІЗУАЛІЗАЦІЯ POS-СИСТЕМИ

4.1 Процес та результати тестування

Для проектів такого роду існує безліч засобів та методі тестування починаючи від написання та виконання unit-тестів, закінчуючи ручним тестуванням додатку. Ціль перевірки POS-системи полягає у перевірці роботи маршрутів серверної частини, оскільки саме у цьому компоненті зосереджена найважливіша частина функціоналу системи. Серверна частина обробляє запити, керує бізнес логікою, взаємодіє з базою даних, забезпечує безпеку даних і відповідає за стабільність і надійність всього додатку.

В процесі тестування за допомогою «Postman» було створено колекцію запитів для перевірки основних маршрутів серверної частини POS-системи (рис. 4.2.1):

– отримання списку продуктів, маршрут «/products-object»;

– додавання нового продукту, маршрут «/add-product»;

– редагування існуючого продукту, маршрут «/product/edit/<int:product_id>»;

– видалення продукту, маршрут «/product/delete/<int:product_id>»;

– отримання списку клієнтів, маршрут «/customer-object»;

– додавання нового клієнту, маршрут «/add-customer»;

– редагування існуючого клієнта, маршрут «/customer/edit/<int:customer_id>»;

– видалення клієнта, маршрут «/customer/delete/<int:customer_id>»;

– отримання списку замовлень, маршрут «/orders-object»;

– отримання деталей замовлення, маршрут «/order/<int:id_order>»;

– створення нового замовлення, маршрут «/create-order»;

– редагування існуючого замовлення, маршрут «/order/edit/<int:id_order>»;

– видалення замовлення, маршрут «/order/delete/<int:id_order>» [11].

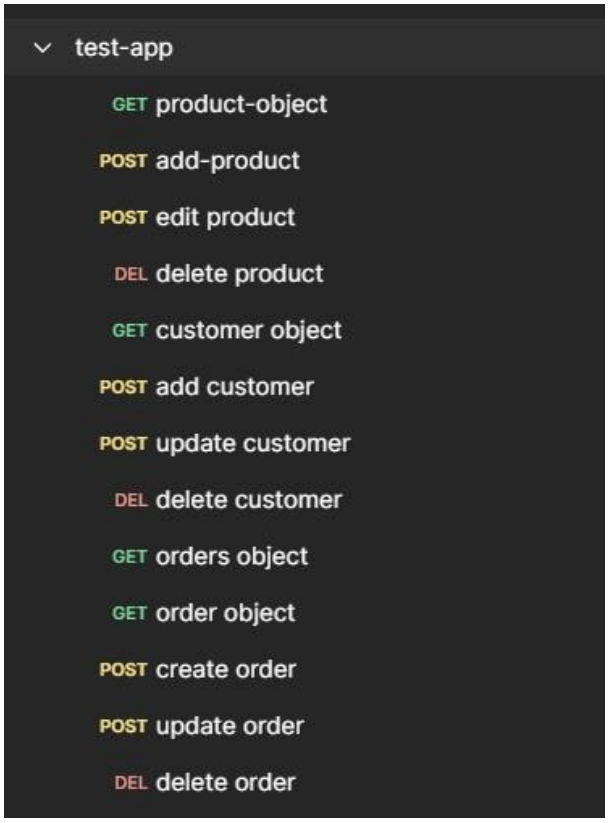

Рисунок 4.2.1 – Колекція запитів для тестування

Для кожного з перерахованих запитів було автоматично згенеровано тести, та виконано процедуру перевірки, результат якої зображений на рисунку 4.2.2.

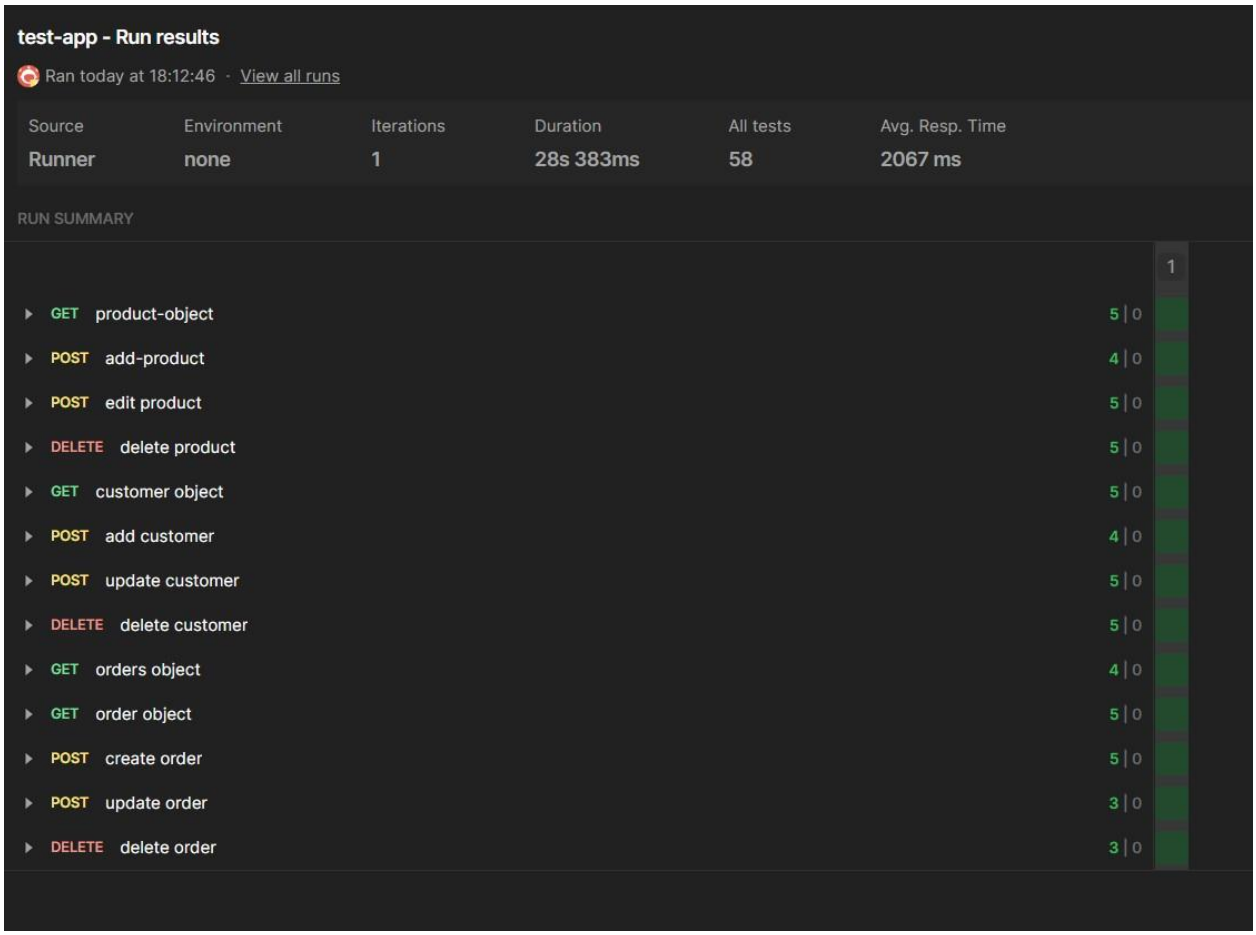

Рисунок 4.2.2 – Результат тестування системи

4.2 Візуалізація сторінок системи

Візуалізація сторінок системи є ключовим аспектом розробки POS-системи, оскільки саме інтерфейс є тим, що користувач бачить у першу чергу. Інтерфейс повинен бути інтуїтивно-зрозумілим, чітким і простим, естетично привабливим, адаптивним й доступним. Дотримуючись таких принципів було сконструйовано сторінки системи, декілька з яких було зображено на рисунках 4.3.1, 4.3.2 й 4.3.3.

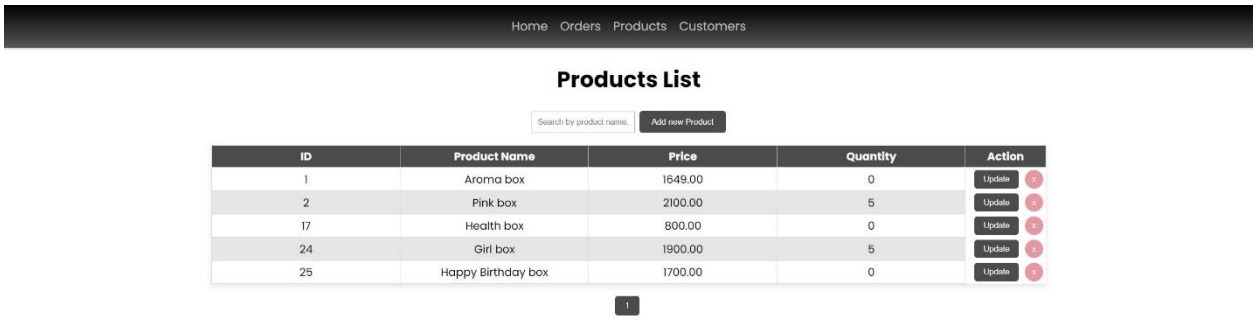

Рисунок 4.3.1. – Візуалізація сторінки перегляду списку продуктів

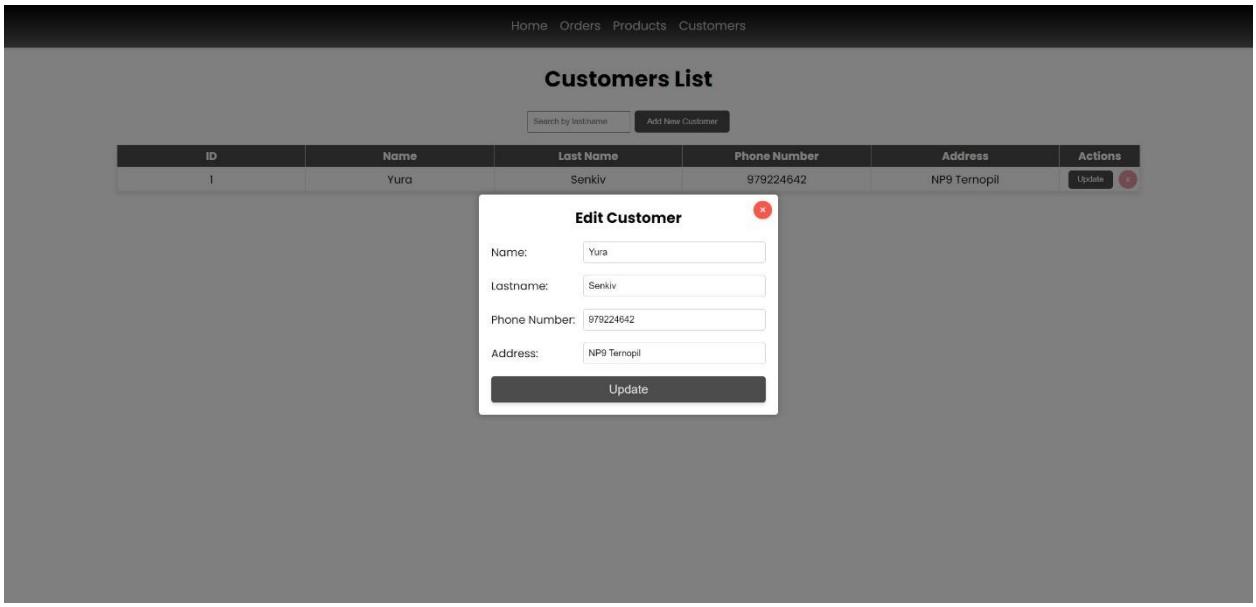

Рисунок 4.3.2 – Візуалізація сторінки «Customers» з відкритим вікном редагування даних клієнта

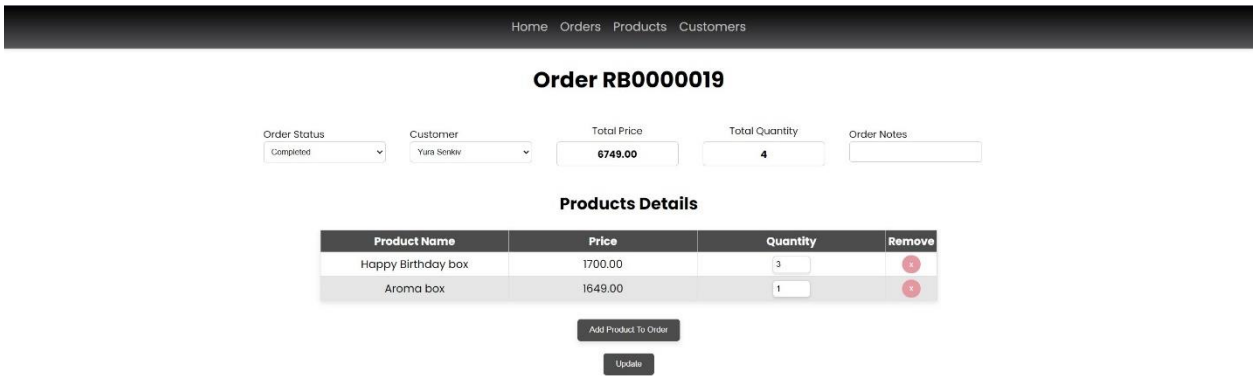

Рисунок 4.3.3 – Візуалізація сторінки редагування існуючого замовлення

5 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ ТА ОСНОВИ ОХОРОНИ ПРАЦІ

5.1 Таксонометрія небезпек

Кваліфікаційна робота освітнього рівня «Бакалавр» присвячена розробці POS-системи для інтернет-магазину з використанням фреймворків Flask та React. В процесі функціонування інтернет-магазину і його обслуговування відбуваються різні види діяльності. Тому в параграфі «Безпека життєдіяльності» доцільно розглянути таксонометрію небезпек.

З метою визначення небезпек та спрямування заходів щодо попередження їх прояву, зниження розміру негативних наслідків складають таксономію, квантифікацію та ідентифікацію небезпек.

Таксономія небезпек є процесом класифікації та організації явищ, процесів, інформації та об’єктів, які можуть спричинити шкоду [18, с.17]. Небезпеки класифікують таким чином:

– за походженням (природні, техногенні, соціально-політичні, комбіновані);

– за локалізацією (космічні атмосферні, літосферні, гідросферні);

– за наслідками (захворювання, травми, загибель, пожежі, забруднення);

– за шкодою (соціальні, технічні, екологічні);

– за сферою прояву (побутові, виробничі, спортивні, дорожньо-транспортні);

– за часом проявлення (імпульсні, кумулятивні);

– за характером дії на людину (активні і пасивні (останні активізуються за рахунок енергії, носієм якої є сама людина, що наражається на гострі, нерухомі елементи, ями, ухили, нерівності поверхні тощо)).

Квантифікація небезпек – кількісна оцінка збитків, заподіяна небезпекою. Така оцінка залежить від багатьох чинників, серед яких: кількість

людей, що знаходились у небезпечній зоні, кількість та якість матеріальних (в тому числі і природніх) цінностей, що перебували там, природні ресурси, перспективності зони тощо [18, с.142].

Ідентифікація небезпек – це визначення типу небезпеки та встановлення її характеристик, що є необхідними для розробки заходів з усунення або мінімізації її наслідків. Цей процес здійснюється відносно об’єктів господарювання щодо визначення потенційної небезпеки та потенційно-небезпечних об’єктів з присвоєння відповідного класу підвищеної небезпеки [18, с.174].

У процесів ідентифікації розглядаються:

– внутрішні фактори, що характеризують небезпечність будов, споруд, обладнання, технологічних процесів суб’єкта господарської діяльності та небезпечних речовин, що виготовляються, переробляються, зберігаються чи транспортуються на його території;

– зовнішні фактори, що безпосередньо не пов’язані з функціонуванням суб’єкта господарської діяльності, але можуть ініціювати виникнення аварійних надзвичайних ситуацій;

– небезпечні події, в перелік яких можуть входити катастрофи, аварії, пожежі, стихійні лиха, епідемії тощо, які при певних умовах можуть призвести до НС.

Найбільш вдалою класифікацією небезпек є класифікація за джерелами походження, згідно з якою всі небезпеки поділяються на групи: природні, техногенні, соціальні, політичні та комбіновані [19, п.1.3.2].

Природні небезпечні джерела – це об'єкти, явища та стихійні лиха, що можуть становити загрозу для життя або здоров'я людей, включаючи землетруси, вулкани, повені, снігові лавини, урагани, ожеледиці, блискавки, сонячне та космічне випромінювання, а також небезпечні рослини, тварини, бактерії і віруси.

Техногенні небезпеки – це ризики, пов'язані з використанням легкозаймистих та вибухонебезпечних речовин, процесів, що відбуваються за підвищених температур і тисків, транспортних засобів, електричної енергії, хімічних речовин і різних видів випромінювання.

До соціальних джерел небезпек належать небезпеки, викликані низьким духовним та культурним рівнем людей: п’янство, алкоголізм, наркоманія, бродяжництво, злочинність тощо.

Джерелами політичних небезпек є конфлікти на міжнаціональному та міждержавному рівнях, політичний тероризм, ідеологічні, міжпартійні, міжконфесійні та збройні конфлікти, війни.

Небезпеки комбінованого характеру зустрічаються найчастіше.

5.2 Інженерно-технічні рішення з охорони праці

Охорона праці є невід'ємною складовою будь-якого виробничого процесу та життєдіяльності підприємств. Її головною метою є забезпечення безпечних та здорових умов праці, мінімізація ризиків виникнення нещасних випадків та професійних захворювань. У сучасному суспільстві неабияка увага приділяється не тільки дотриманню законодавчих норм і стандартів, але й впровадженню інженерно-технічних заходів, що є запорукою ефективної охорони праці.

Для підприємства, що займається розробкою POS-систем було визначено ряд інженерно-технічних рішень з:

- організації робочих місць;
- забезпечення безпеки електрообладнання;
- мікрокліматичних норм;
- навчання та інструктажу з охорони праці;

Далі кожне зазначене рішення розглядається окремо, базуючись на переліку офіційних документів, які відповідають вимогам чинного законодавства України та нормативним актам з безпеки та охорони праці.

Згідно з Законом України «Про охорону праці», працівникам необхідно забезпечувати безпечні та здорові умови праці, що включає організацію ергономічних робочих місць, освітлення відповідно до ДБН В.2.5-28:2018 та використання безпечних електроустановок згідно з Правилами безпечної експлуатації електроустановок споживачів, що включає встановлення розеток із захисними кришками, розподільчих щитків з автоматичними вимикачами й регулярний технічний огляд та обслуговування електрообладнання [20, 21, 22].

З метою забезпечення мікрокліматичних норм відповідно до ДСН 3.3.6.042-99 приміщення повинні відповідати допустимим умовам мікроклімату, зображеним на рисунку 5.2.1 [23].

Період року	Категорія робіт	Температура повітря	Відносна вологість	Швидкість руху, м/сек.
Холодний період року	Легка Іа	22 - 24	60 - 40	0,1
	Легка Іб	21 - 23	60 - 40	0,1
	Середньої важкості IIа	19 - 21	60 - 40	0,2
	Середньої важкості IIб	17 - 19	60 - 40	0,2
	Важка III	16 - 18	60 - 40	0,3
Теплий період року	Легка Іа	23 - 25	60 - 40	0,1
	Легка Іб	22 - 24	60 - 40	0,2
	Середньої важкості IIа	21 - 23	60 - 40	0,3
	Середньої важкості IIб	20 - 22	60 - 40	0,3
	Важка III	18 - 20	60 - 40	0,4

Рисунок 5.2.1 – Допустимі умови мікроклімату

Відповідно до типового положення про порядок проведення навчання і перевірки знань з питань охорони праці (НПАОП 0.00-4.12-05) працівникам було забезпечене регулярне проведення інструктажів та перевірки знань з питань охорони праці [24].

ВИСНОВКИ

У процесі виконання даної роботи було здійснено повний цикл розробки POS-системи, починаючи з постановки завдання та аналізу вимог, закінчуючи тестуванням та візуалізацією системи. Це дозволило досягти значного професійного зростання та здобути цінний досвід у сфері інженерії програмного забезпечення.

Реалізація проєкту сприяла глибокому розумінню сучасних технологій та інструментів розробки програмного забезпечення. Це включає знання архітектури систем, баз даних, технологій для конструювання серверної та клієнтської частин додатку. Використання фреймворків Flask та React дозволило оволодіти передовими методами веб-розробки та забезпечити високу продуктивність та масштабованість системи.

Інтеграція різних компонентів системи та забезпечення їх коректної взаємодії значно підвищила компетентність у проектуванні та розробці складних систем. Використання мови програмування Python для серверної частини та JavaScript для клієнтської частини дозволило створити ефективний та надійний продукт. Процес інтеграції включав налаштування системи керування базами даних MySQL, що дозволило забезпечити надійне зберігання та швидкий доступ до даних.

Виконання завдань потребувало ефективного планування та організації робочого процесу, що включало управління часом, розподіл ресурсів та моніторинг прогресу. Використання методів гнучкої розробки (Agile) дозволило оптимізувати робочий процес, швидко адаптуватися до змін та забезпечити високий рівень продуктивності команди.

Поглиблений аналіз вимог та їх реалізація в системі сприяли кращому розумінню потреб бізнесу та кінцевих користувачів. Це дозволило створити інтуїтивно-зрозумілий інтерфейс, який відповідає очікуванням користувачів та забезпечує високу зручність використання. Вивчення принципів UX/UI-

дизайну дозволило створити ефективні та привабливі інтерфейси, що підвищили загальну зручність та задоволеність користувачів.

Застосування принципів дизайну орієнтованого на користувача забезпечило створення ефективних та привабливих інтерфейсів. Ретельне дослідження поведінки користувачів та їхніх потреб дозволило забезпечити високий рівень зручності та доступності системи.

Проведення ретельного тестування забезпечило перевірку системи, що дозволило виявити потенційні проблеми та гарантувати високу якість кінцевого продукту. Використання різних методів тестування, таких як юніт-тести, інтеграційні тести та тестування користувацького інтерфейсу, дозволило забезпечити надійність та стабільність роботи системи.

Розробка даної системи може служити основою для подальших досліджень та впроваджень, спрямованих на покращення існуючих та створення нових інформаційних систем. Використання новітніх технологій та методів розробки дозволяє створювати високоякісні програмні продукти, що відповідають сучасним вимогам ринку.

Таким чином, виконана робота не лише продемонструвала практичні навички розробки програмного забезпечення, але й зробила вагомий внесок у професійний розвиток. Оволодіння новими технологіями та методами, а також набуття досвіду в управлінні проєктами, сприятиме подальшому розвитку кар'єри та відкриває нові можливості у сфері інформаційних технологій.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Що таке POS-система? [Електронний ресурс] – Режим доступу до ресурсу: https://magefan.com/ua/blog/shtcho-take-pos-systema#shtcho-take-pos-systema.
2. Ключові методології розробки програмного забезпечення [Електронний ресурс] – Режим доступу до ресурсу: https://wezom.com.ua/ua/blog/metodologija-razrabotki-programmnogo-obespechenija.
3. What is Client Server Architecture? [Електронний ресурс] – Режим доступу до ресурсу: https://u-next.com/blogs/cyber-security/what-is-client-server-architecture/.
4. Python for Web Development [Електронний ресурс] – Режим доступу до ресурсу: https://djangostars.com/blog/python-web-development/.
5. Python Developer’s Guide [Електронний ресурс] – Режим доступу до ресурсу: https://devguide.python.org/getting-started/.
6. Introduction to Web development using Flask [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/python-introduction-to-web-development-using-flask/.
7. Документація Flask [Електронний ресурс] – Режим доступу до ресурсу: https://flask.palletsprojects.com/en/3.0.x/quickstart/.
8. Документація MySQL [Електронний ресурс] – Режим доступу до ресурсу: https://dev.mysql.com/doc/.
9. Особливості інструменту HeidiSQL [Електронний ресурс] – Режим доступу до ресурсу: https://www.heidisql.com/#featurelist.
10. What is Postman [Електронний ресурс] – Режим доступу до ресурсу: https://www.postman.com/product/what-is-postman/.

11. Документація Postman [Електронний ресурс] – Режим доступу до ресурсу: https://learning.postman.com/docs/introduction/overview/.

12. Елементи UML [Електронний ресурс] – Режим доступу до ресурсу: https://docs.kde.org/trunk5/uk/umbrello/umbrello/uml-elements.html.

13. Як організовані реляційні бази даних [Електронний ресурс] – Режим доступу до ресурсу: https://data-b-i.com/uk/article/yak-organyzoveny-relyatsiyni-bazy-danykh.html.

14. Creation of virtual environments [Електронний ресурс] – Режим доступу до ресурсу: https://docs.python.org/3/library/venv.html.

15. Огляд React [Електронний ресурс] – Режим доступу до ресурсу: https://react.dev/.

16. Документація React [Електронний ресурс] – Режим доступу до ресурсу: https://react.dev/learn.

17. Огляд Visual Studio Code [Електронний ресурс] – Режим доступу до ресурсу: https://code.visualstudio.com/.

18. Зацарний В.В., Зацарна О.В., Землянська О.В., Праховнік Н.А.. «Безпека Життєдіяльності». Навчальний посібник. – Київ: НТТУ «КПІ», 2016. – 227 с.

19. Атаманчук П.С., Мендерецький В.В., Панчук О.П., Чорна О.Г.. «Безпека Життєдіяльності». Навчальний посібник. – Київ: Центр учбової літератури, 2011. – 276 с.

20. Закон України «Про охорону праці» [Електронний ресурс] – Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/2694-12#Text.

21. ДБН В.2.5-28:2018 [Електронний ресурс] – Режим доступу до ресурсу: https://e-construction.gov.ua/laws_detail/3074958732556240833?doc_type=2.

22. Правила безпечної експлуатації електроустановок споживачів [Електронний ресурс] – Режим доступу до ресурсу: https://antifire.ua/dbn/10.pdf.

23. ДСН 3.3.6.042-99 [Електронний ресурс] – Режим доступу до ресурсу: https://zakon.rada.gov.ua/rada/show/va042282-99#Text.

24. НПАОП 0.00-4.12-05 [Електронний ресурс] – Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/z0231-05#n32.

ДОДАТКИ

Додаток А – Тези конференції

УДК 004.41
Сеньків Ю. – ст. гр. СП-42
Тернопільський національний технічний університет імені Івана Пулюя

РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ІНТЕРНЕТ-МАГАЗИНУ З ВИКОРИСТАННЯМ REACT ТА FLASK

Науковий керівник: д. ф.-м. н., професор Михалик Д. М.

Senkiv Y.
Ternopil Ivan Puluj National Technical University

DEVELOPMENT OF SOFTWARE FOR AN ONLINE-SHOP USING REACT AND FLASK

Supervisor: PhD in Mathematical Sciences, Professor Mykhalyk D. M.

Ключові слова: POS-система, React, Flask, ефективність, користувацьке задоволення
Keywords: POS-system, React, Flask, efficiency, user satisfaction

Моя наукова робота спрямована на створення веб-додатку, POS-системи, для інтернет-магазину за допомогою React та Flask. Ця робота має на меті поліпшення процесів обробки замовлень та збільшення ефективності користувачів.

Початково, я проводжу аналіз вимог до POS-системи, щоб зрозуміти, які функції потрібно включити до веб-додатку. На основі цього аналізу я розробляю архітектуру застосунку та обираю React та Flask як основні технології для реалізації. React відомий своєю швидкістю та зручністю в розробці інтерфейсу, тоді як Flask забезпечує легкість та гнучкість у розробці серверної частини додатку.

Основні функції веб-додатку включають обробку даних клієнтів, створення замовлень, відслідковування кількості товарів. Додаток також надає адміністративні можливості для управління продуктами, замовленнями та іншими аспектами POS-системи.

Завдяки цьому веб-додатку, підприємство зможе оптимізувати роботу з даними клієнтів, забезпечити зручну та ефективну систему обробки замовлень і підвищити загальне задоволення від користування POS-системою.

Література

1. React Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://react.dev/learn.
2. Flask Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://flask.palletsprojects.com/en/3.0.x/.

Додаток Б – Лістинг функції «createOrder» додавання нового замовлення

```
def create_order():
    try:
        data = request.json
        db.connection.begin()
        cursor = db.connection.cursor()

        order_obj.id_customer = data.get('id_customer')
        order_obj.total_price = data.get('total_price')
        order_obj.total_quantity = data.get('total_quantity')
        order_obj.status = data.get('status')
        order_obj.notes = data.get('notes')
        # create order
        co_query, co_params = Order.create_order(order_obj)
        cursor.execute(co_query, co_params)
        # generate order reference
        cursor.execute(Order.get_last_inserted_id())
        result = cursor.fetchone()
        order_id = result[0]
        order_increment = ''

        for i in range(7 - len(str(order_id))):
           order_increment += '0'

        order_obj.order_reference = 'RB' + order_increment +
str(order_id)
        order_obj.id_order = order_id

        uor_query, uor_params =
Order.update_order_reference(order_obj)
        cursor.execute(uor_query, uor_params)

        products_data = data.get('productData')
        for product in products_data:
            # add order detail
            order_details_obj.id_order = order_id
            order_details_obj.id_product = product[0]
            order_details_obj.order_product_quantity =
product[1]
            aod_query, aod_params =
OrderDetails.add_order_detail(order_details_obj)
            cursor.execute(aod_query, aod_params)

            # update product quantity on order create
            product_obj.id_product = product[0]
```

Продовження лістингу Додатку Б

```
            product_obj.quantity = product[1]
            upq_query, upq_params =
Product.update_product_quantity_on_order_create(product_obj)
            cursor.execute(upq_query, upq_params)

        db.connection.commit()
        return jsonify({'message': "Order has been successfully
added!"})
    except Exception as e:
        return jsonify({'error:': 'Failed to fetch order
details!'})
    finally:
        db.connection.close()
```

Додаток В – Лістинг коду сторінки створення замовлення

```
import { useEffect, useState } from "react";
import AddNewCustomer from
"../Components/addNewCustomerComponent";
import CreateOrderModal from
"../Components/createOrderComponent";
import '../Order/productDetails.css'
import '../Order/order.css'

export function CreateOrder() {
    const [customers, setCustomers] = useState([]);
    const [loading, setLoading] = useState(true);
    const [customer, setCustomer] = useState("");
    const [customerName, setCustomerName] = useState("");
    const [isCreateOrderModalOpen, setCreateOrderModalOpen] =
useState(false);
    const [isNextButtonVisible, setNextButtonVisible] =
useState(true);
    const [customerToAdd, setCustomerToAdd] = useState({
        customerId: "",
    });
    const [nextButtonClicked, setNextButtonClicked] =
useState(false);
    const [addCustomerButtonVisible,
setAddCustomerButtonVisible] = useState(true);

    useEffect(() => {
        fetchData();
    }, []);

    const fetchData = async () => {
        try {
            const customerResponse = await fetch("/customer-
object");
            const customerData = await customerResponse.json();

            setCustomers(customerData);
            setLoading(false);
        } catch (error) {
            console.error("Error fetching data:", error);
        }
    };

    const [addCustomerModalOpen, setAddCustomerModalOpen] =
useState(false);
```

Продовження лістингу Додатку В

```
const [newCustomerData, setNewCustomerData] = useState({
        name: "",
        lastName: "",
        phoneNumber: "",
        address: ""
    });

    const openAddCustomerModal = () => {
        setAddCustomerModalOpen(true);
    };
    const closeAddCustomerModal = () => {
        setAddCustomerModalOpen(false);

        setNewCustomerData({
            name: "",
            lastName: "",
            phoneNumber: "",
            address: ""
        });
    };
    const confirmAddCustomer = async () => {
        await fetch('/add-customer', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
            },
            body: JSON.stringify(newCustomerData),
        });
        closeAddCustomerModal();
        fetchData();
    };
    const handleCustomerChange = (event) => {
        const customerId = event.target.value;
        const selectedCustomer = customers.find(customer =>
customer.id_customer === parseInt(customerId));
        setCustomer(customerId);
        setCustomerName(selectedCustomer ?
`${selectedCustomer.customer_name}
${selectedCustomer.customer_lastname}` : "");
        setCustomerToAdd({
            ...customerToAdd,
            customerId: customerId,
        });

        setNextButtonVisible(!!customerId);
    };
```

Продовження лістингу Додатку В

```
    const closeCreateOrderModal = () => {
        setCreateOrderModalOpen(false);
        setNextButtonVisible(true);
        setCustomer("");
        setCustomerName("");
    };

    const openCreateOrderModal = () => {
        setCreateOrderModalOpen(true);
        setNextButtonVisible(false);
        setNextButtonClicked(true);
        setAddCustomerButtonVisible(false);
    };

    return (
        <>
            <div className="h1-header">
                <h1>Create New Order</h1>
            </div>
            {loading ? (
                <div className="spinner-container">
                    <div className="loading-spinner"></div>
                </div>
            ) : (
                <div className="precontainer">
                    <div className="container">
                        <div className="customer-info">
                            <span className="customer-span-
block">Customer: </span>
                            {nextButtonClicked ? (
                                <span>{customerName}</span>
                            ) : (
                                <select
                                    id="customer-select"
                                    value={customer}
                                    onChange={handleCustomerChan
ge}
                                >
```

Продовження лістингу Додатку В

```
                                        <option value="">Select
Customer</option>
                                        {customers.map((customer) =>
(
                                            <option
                                                key={customer.id_cus
tomer}
                                                value={customer.id_c
ustomer}
                                            >
                                                {`${customer.custome
r_name} ${customer.customer_lastname}`}
                                            </option>
                                        ))}
                                    </select>
                                )}
                                {addCustomerButtonVisible &&
!nextButtonClicked && (
                                    <button className="add-new-
customer-button" onClick={openAddCustomerModal}>
                                        Add New Customer
                                    </button>
                                )}
                            </div>
                        </div>
                        <div className="next-step">
                            {customer && isNextButtonVisible && (
                                <button
                                    className="next-step-button"
                                    onClick={openCreateOrderModal}
                                >
                                    Next
                                </button>
                            )}
                        </div>
                    </div>
                )}
                <AddNewCustomer
                    isOpen={addCustomerModalOpen}
                    onClose={closeAddCustomerModal}
                    onConfirm={confirmAddCustomer}
                    onChange={setNewCustomerData}
                    newData={newCustomerData}
                />
                <CreateOrderModal
                    isOpen={isCreateOrderModalOpen}
                    onClose={closeCreateOrderModal}
```

Продовження лістингу Додатку В

```
        customerToAdd={customerToAdd}
                />
            </>
        );
}
```

Додаток Г – Диск з роботою