Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студентки *Райгородської Дар'ї Олегівни*
(ПІБ)

академічної групи *122-20-1*
(шифр)

спеціальності *122 Комп'ютерні науки*
(код і назва спеціальності)

освітньої програми *Комп'ютерні науки*
(назва освітньої програми)

на тему: *Розробка веб-додатку для он-лайн магазину для IT-компанії на ASP.NET*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	*доц. Гуліна І.Г.*			
розділів:				
спеціальний	*доц. Гуліна І.Г.*			
економічний	*доц. Касьяненко Л.В.*			
Рецензент				
Нормоконтролер	*доц. Гуліна І.Г.*			

Дніпро
2024

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

М.О. Алексєєв
(підпис) (прізвище, ініціали)

« » 2024 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студентки *122-20-1* *Райгородської Дар'ї Олегівни*
(група) (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка веб-додатку для он-лайн магазину для IT – компанії на ASP.NET*

затверджена наказом ректора НТУ «ДП» від 23.05.2024 р. № 469-с

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів виробничої практики та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі.*** ***Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***01.06.2024 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***06.06.2024 р.***

Завдання видала _____ *доц. Гуліна І.Г.*
(підпис) (посада, прізвище, ініціали)

Завдання прийняла до виконання _____ *Райгородська Д.О.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2024 р.*

Термін подання кваліфікаційної роботи до ЕК: *10.06.2024 р.*

РЕФЕРАТ

Пояснювальна записка: 116 с., 60 рис., 3 дод., 41 джерел.

Об’єктом дослідження є веб-орієнтований додаток для надання послуг інтернет-компанії.

Мета кваліфікаційної роботи: створення застосунку, що дозволить пропонувати різноманітні сервіси, такі як хмарні та електронні продукти, в комерційних цілях, а також для всіх користувачів.

У вступі проводиться аналіз та описується сучасний стан проблеми, визначається мета кваліфікаційної роботи та сфера її застосування, обґрунтовується актуальність теми та деталізується завдання.

У першому розділі проаналізовано предметну галузь, визначено актуальність завдання та призначення розробки, сформульовано постановку завдання, зазначено вимоги до програмної реалізації, технологій та програмних засобів.

У другому розділі проаналізовані наявні рішення, обрано платформу для розробки, виконано проектування і розробка веб – орієнтованого додатку, описана робота системи, алгоритм і структура її функціонування, а також виклик та завантаження програми, визначено вхідні и вихідні дані, охарактеризовано склад параметрів технічних засобів.

В економічному розділі визначено трудомісткість розробленої системи, проведений підрахунок вартості роботи по створенню застосунку та розраховано час на його створення.

Практичне значення полягає в створенні веб – орієнтованого додатку інтернет – магазину, що забезпечує покращення процесів взаємодії між клієнтами та компанією шляхом автоматизації процесів, та дозволяє створювати замовлення через Інтернет.

Актуальність полягає у відсутності місцевих аналогів на ринку Україні, зростанні попиту на цифрові послуги та продукти, та підвищення кібербезпеки через локальне розміщення даних.

Список ключових слів: SQLITE, AJAX, ДОДАТОК, ЗАСТОСУНОК, ПАНЕЛЬ МОДЕРАТОРА, МОДЕРАТОР, АРХІТЕКТУРА, КЛІЄНТ, ДИЗАЙН.

ABSTRACT

Explanatory Note: 116 pages, 60 figures, 3 appendices, 41 sources.

Development object: a web-oriented application for providing services to an internet company.

The purpose of the qualification work is to create an application that offers various services, such as cloud and electronic products, for commercial purposes and for all users.

The introduction analyzes and describes the current state of the problem, defines the aim of the qualification work and its application field, justifies the relevance of the topic, and details the task.

The first section, the subject area is analyzed, the relevance of the task and the purpose of the development, formulates the task, and specifies the requirements for software implementation, technologies, and tools.

In the second section, available solutions are analyzed, the platform for development selected, performs the design and development of the web-oriented application, describes the system's operation, its algorithm and functional structure, as well as the program call and load, defines the input and output data, and characterizes the set of technical parameters.

In the economic section, the labor intensity of the developed system is determined, the cost of work for creating the application is calculated, and the time required for its creation is estimated.

The practical significance lies in the creation of a web-oriented application for an online store, which ensures improved interaction processes between clients and the company through process automation and allows orders to be placed via the internet.

The relevance lies in the absence of local analogs in the Ukrainian market, the growing demand for digital services and products, and increased cybersecurity through local data storage.

Keywords: SQLITE, AJAX, APPLICATION, SOFTWARE, MODERATOR PANEL, MODERATOR, ARCHITECTURE, CLIENT, DESIGN.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – база даних;

UML – Unified Modeling Language;

IT – інформаційні технології;

ОС – операційна система;

CSS – Cascading Style Sheets;

SQL – Structured Query Language;

HTML – HyperText Markup Language;

JS – JavaScript;

CMS – Content Management System;

FTP – File Transfer Protocol;

BEM- Block, Element, Modifier;

MVC- Model-View-Controller.

ВСТУП

Порівняно з традиційною торгівлею, компанії, що займаються інтернет-бізнесом, мають значно більший потенціал для зростання. Для інтернет-магазинів це означає можливість створення привабливих умов для покупок та пропозицій. Обсяги комерційних операцій в інтернет-бізнесі продовжують зростати. Традиційна торгівля передбачає взаємодію клієнта з фізичним середовищем підприємства, персоналом та послугами, які вони надають. Натомість електронні технології пропонують абсолютно інший клієнтський досвід, часто кардинально відмінний від традиційних форматів торгівлі. Розвиток галузі інтернет – покупок та сервісів, а саме створення платформи для цього, може сприяти підвищенню доступності до певного асортименту товарів та послуг. Крім того, це призводить до автоматизації та зменшення помилок, через зменшення людського втручання до процесів покупок.

В даній кваліфікаційній роботі розглянуто веб – орієнтований додаток, предметною областю якого є інтернет магазин товарів, що націлений на потреби ІТ компанії, для зручного представлення послуг. Головна ідея полягає в створенні платформи саме на території України, яка б надавала перелік продуктів та сервісів для користувачів. Це дозволить використати ресурси з найбільшою швидкістю, адже сервери будуть знаходитися в більш локальному регіоні, порівняно з закордонними аналогами.

Об’єктом дослідження є веб – орієнтований додаток для надання послуг інтернет – компанії.

Мета кваліфікаційної роботи полягає в створенні застосунку, що дозволить пропонувати різноманітні сервіси, такі як хмарні, та електронні продукти в комерційних цілях, а також для всіх користувачів.

Для досягнення поставленої мети є необхідним вирішення переліку наступних задач, які включають:

– розглянути конкурентів ринку IT в галузі продажу цифрових копій програмного забезпечення та сервісних послуг;

– визначити програмні та технологічні вимоги до створення платформи. Дані вимоги охоплюють список використаних технологій, а також функціональні характеристики для забезпечення відмовостійкості в роботі серверів розробленого веб – орієнтованого додатку;

– розробити веб – застосунок та описати його структуру;

– здійснити розрахунок вартості створення платформи, та її трудомісткість.

Актуальність розробки веб – орієнтованого додатку полягає:

– відсутності місцевих аналогів на ринку України, що пропонують комплексні сервіси для IT – компаній. Локальні рішення забезпечать кращу доступність і швидкість роботи завдяки розміщенню серверів у межах країни, що зменшить затримки та покращить користувацький досвід;

– зріст попиту на цифрові послуги та продукти. Таким чином, така платформа дозволить українським компаніям швидко реагувати на ринкові запити та залишатись конкурентоспроможними;

– підвищення кібербезпеки через локальне розміщення даних, та дозволить контролювати конфіденційність інформації.

Практичне значення у створенні додатку містить наступні пункти:

– покращення взаємодії між клієнтами та компанією шляхом автоматизації процесів;

– створення місцевих серверів для взаємодії з існуючими послугами та сервісами;

– забезпечення створення аналогу на ринку, та підвищення конкурентоспроможності українських компаній;

– створення зручного інтерфейсу керування можливостями, що пропонує веб – застосунок.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1. Загальні відомості з предметної галузі

1.1.1. Аналіз існуючих компаній та їх програмних рішень

З ціллю виявлення плюсів та недоліків в глобальних інтернет – магазинах компаній, задля забезпечення ефективності програмного виробу, було обрано такі крупні компанії та їх підрозділи, як AWS Amazon [1] та Microsoft Store [2].

Хмарні обчислення – це послуги, що надаються за запитом і дозволяють зберігати та легко отримувати дані і проекти. Amazon є провідним постачальником хмарних послуг у світі через свою платформу Amazon Web Services (AWS), яка дає можливість клієнтам зберігати дані у хмарі. Хмарні обчислення є важливими для малих і середніх підприємств, оскільки допомагають їм використовувати нові можливості та конкурувати на рівних з великими компаніями.

AWS надають великий перелік послуг, що може використовувати клієнт. Наприклад, це може бути EC2 екземпляри [3], або шлюзи для програмних інтерфейсів взаємодії з додатками [4], та багатий перелік послуг що може бути вільно використаним.

Розглянувши веб – орієнтований інтерфейс, що зображено на рис. 1.1 – 1.2, можна прийти до висновку, що графічний інтерфейс AWS Amazon надає багатий перелік елементів керування, що включають в себе:

- меню вибору вкладок сервісів;
- ресурси для поточного регіону, що обраний в списку регіонів в шапці;
- елементи управління екземпляром;
- інформація про стан сервісу;
- шапку сайту, з перемикачами та профілем;
- футер сайту з мінімальною кількість елементів;

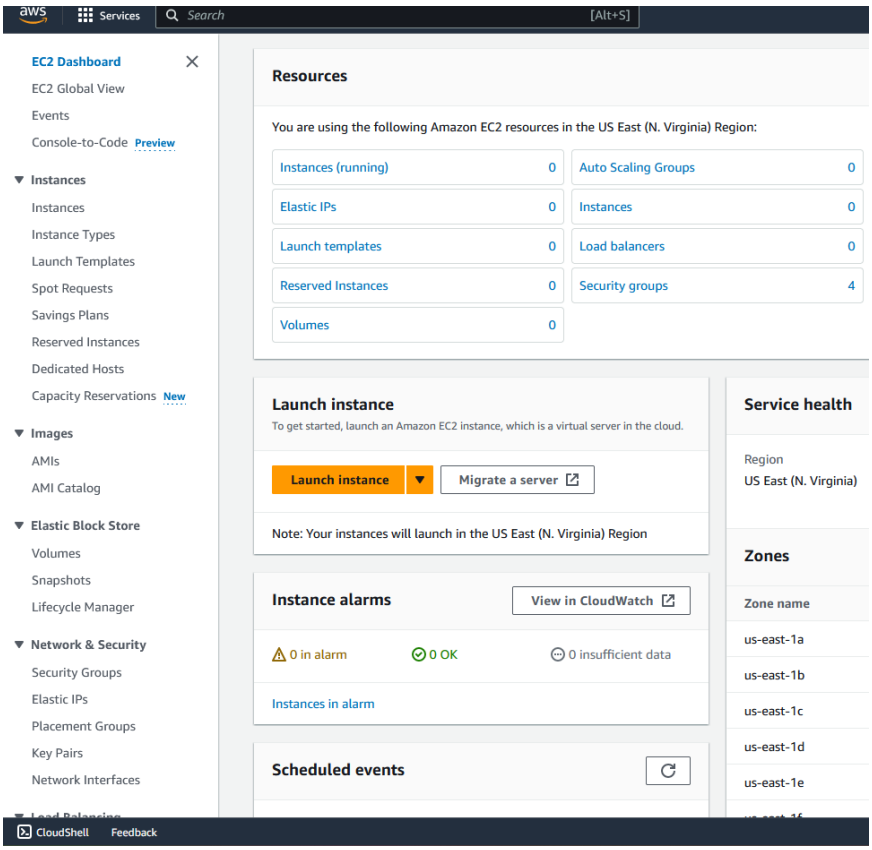

Рис. 1.1. Меню та ресурси управління

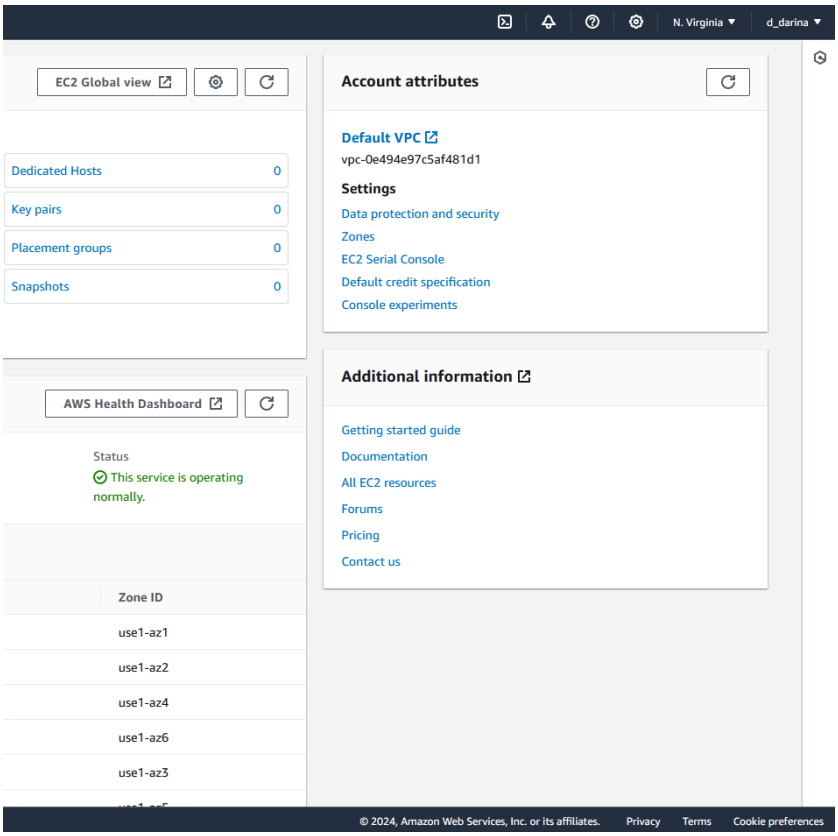

Рис. 1.2. Додаткова інформація

Аналізуючи сайт хмарних сервісів Amazon, можна прийти висновку, що він пропонує досить складний графічний інтерфейс з великим переліком елементів управління. Це збільшує час, який користувач може витратити на освоєння основних інструментів управління сервісами, та освоєння технічних аспектів застосування. Хоч ця платформа має досить велику документацію, але існує ще багатий перелік інформації та підрозділів, які включає в себе AWS, що вимагає додаткового, окрім зазначеного, часу на освоєння.

Іншим програмним рішенням, що розглянуто, є магазин Microsoft Store (рис. 1.3, 1.4). Це рішення пропонує більш спрощений інтерфейс керування і перегляду продуктів, що робить покупки більш інтуїтивними.

Цей додаток є онлайн ринком, де споживачі купують та завантажують різні товари. Магазин дозволяє користувачам купувати апаратне забезпечення, зокрема персональні комп'ютери та консолі, або завантажувати програмне забезпечення та цифровий вміст, включаючи програми, ігри.

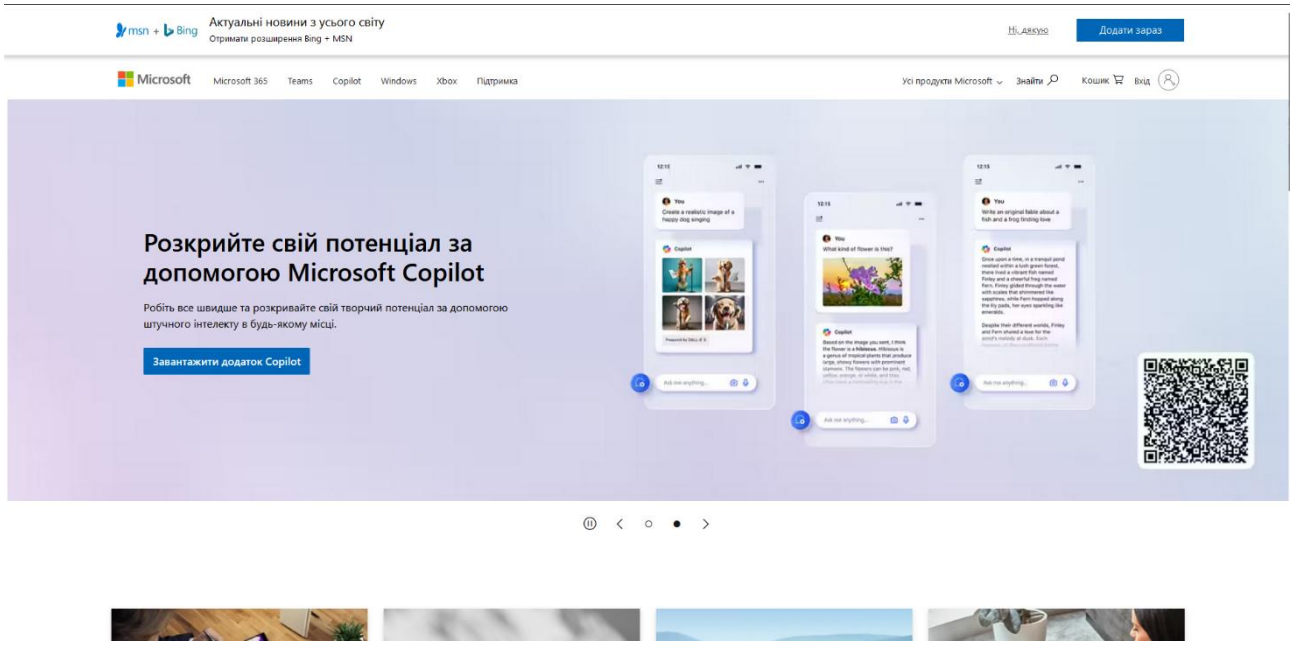

Рис. 1.3. Стартова сторінка сайту Microsoft

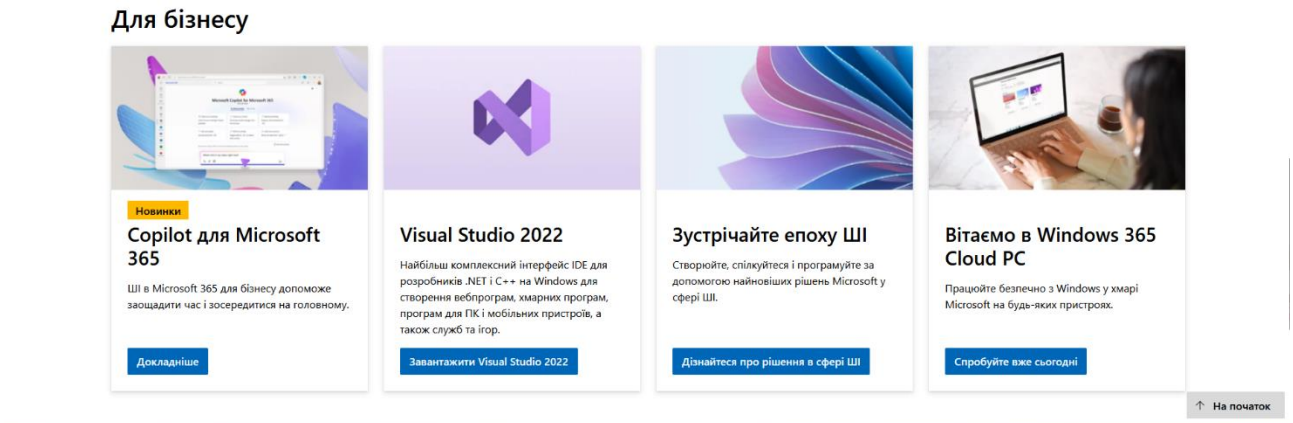

Рис. 1.4. Продукти Microsoft

Як можна побачити, шапка сайту Microsoft пропонує можливості пошуку та перегляду усіх продуктів. Також, наявні елементи профілю і посилань на інші розділи.

Проаналізувавши компанії та їх програмні рішення, слід зазначити, що Microsoft має наступні характеристики графічного дизайну:

– мінімалістичний дизайн з акцентом на простоту та зрозумілість;

– використання переважно синіх та білих кольорів для створення професійного вигляду;

– великі заголовки та читабельний текст, стандартні шрифти;

– іконки та зображення високої якості, використання великих банерів і слайдерів для привернення уваги до ключових продуктів та послуг;

– система швидкого пошуку необхідної інформації.

Розглядаючи функціональність, можна описати наступні елементи:

– інформаційні блоки з описами продуктів, відео – демонстраціями та клієнтськими історіями успіху;

– знаходження ресурсів підтримки, документів та навчальних матеріалів.

Сайт Microsoft має цільову аудиторію в вигляді широкого спектру користувачів, включаючи як технічних спеціалістів, так і звичайних користувачів.

З іншого боку, AWS Amazon має наступні елементи графічного дизайну:

– сайт має більш технічний та функціональний дизайн, спрямований на фахівців в галузі IT;

– основні кольори – помаранчевий та білий, з акцентами на чорний та сірий;

– вузькі шрифти, часто дрібний текст;

– інфографіка, діаграми та схеми пояснення технологічних концепцій.

Функціональність складає:

– розширені ресурси підтримки, включаючи форуми та документацію;

– детальний технічний опис кожного продукту або сервісу, та їх посібники;

– можливість керування хмарними сервісами та мониторинг використання.

Отже, сайт Microsoft має більш дружній до користувачів дизайн та структуру, що робить можливим його використання широким колом користувачів. AWS Amazon фокусується та технічних аспектах, та пропонує детальну інформацію для фахівців й забезпечує необхідними ресурсами.

Тому, виходячи з вище зазначеного висновку, необхідно створити додаток, що об’єднує переваги обох сервісів:

– дружній до користувача дизайн та структуру;

– адаптивність версій сайту для декількох платформ;

– інтуїтивне меню, та можливість управління продуктами;

– розділ технічної підтримки як пункт меню для найбільш частих питань.

1.1.2. Аналіз розроблених програмних рішень в Україні

В якості існуючих рішень, що пропонує суспільство розробників в Україні, обрано сервіси GigaCloud [5] та DeNovo [6] з ціллю порівняння функціоналу, та

запровадження необхідних функцій. Зображення сторінок сайтів наведено на рис. 1.5 – 1.6.

Виходячи з проаналізованих статей, що стосуються опису запропонованих компаній [7], GigaCloud є провідним українським хмарним провайдером, з великою кількістю клієнтів, а також понад 150 глобальних партнерів. Компанія обслуговує як великі корпорації, так і державні установи від малих до середніх підприємств.

DeNovo є провайдером хмарних сервісів, технологічність і надійсність якого підтверджено світовими лідерами VMware[8] та SAP [9]. Керування інфраструктурою відбувається в Україні.

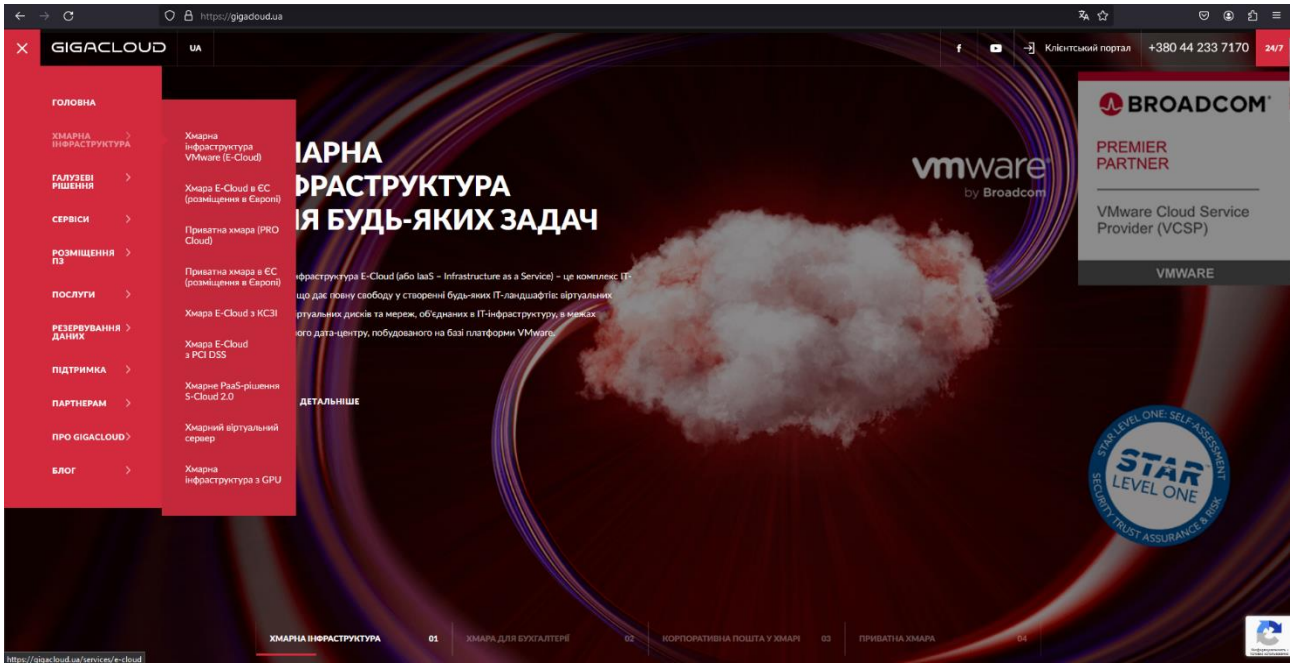

Рис. 1.5. GigaCloud – головна сторінка

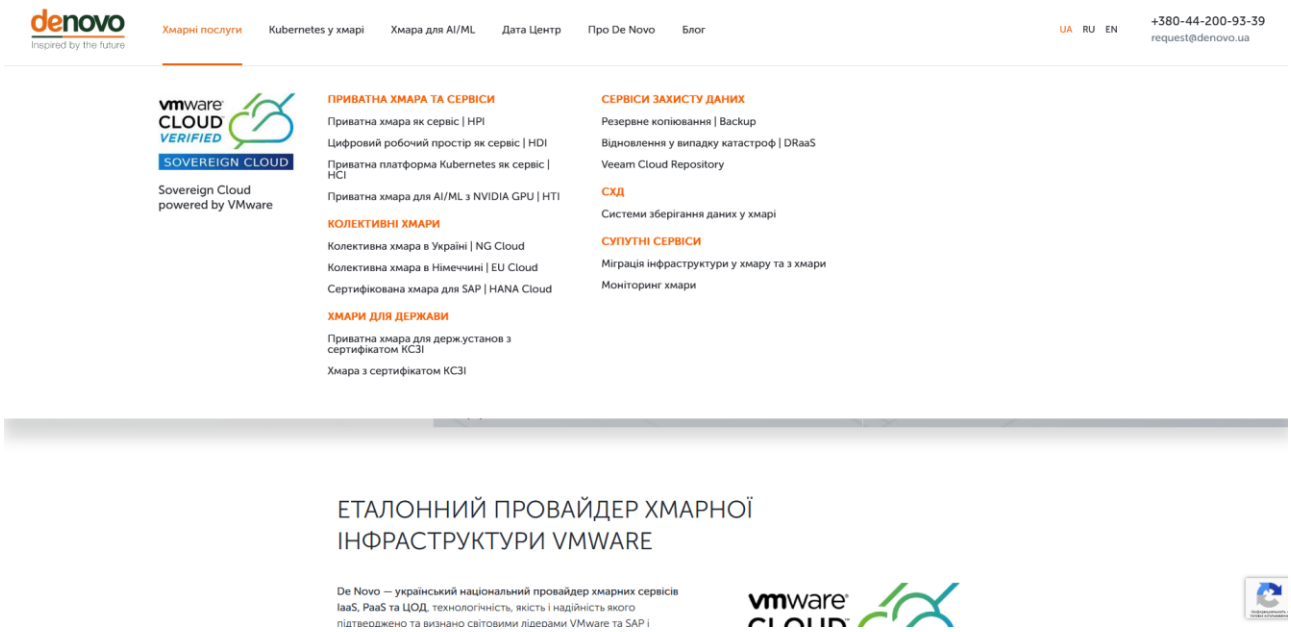

Рис. 1.6. DeNovo – головна сторінка, послуги

Розглянувши інтерфейс обох сайтів, слід зазначити риси дизайну та функціональності. DeNovo має:

– інтуїтивна навігація;

– головна сторінка містить короткий огляд основних послуг компанії (рис 1.7);

– чіткі категорії та дружнє для користувача меню;

– інфраструктуру, яка включає віртуальні машини, мережі рішення, та сховища даних;

– розробку та розгортання додатків;

– консультаційні послуги з міграції в хмару.

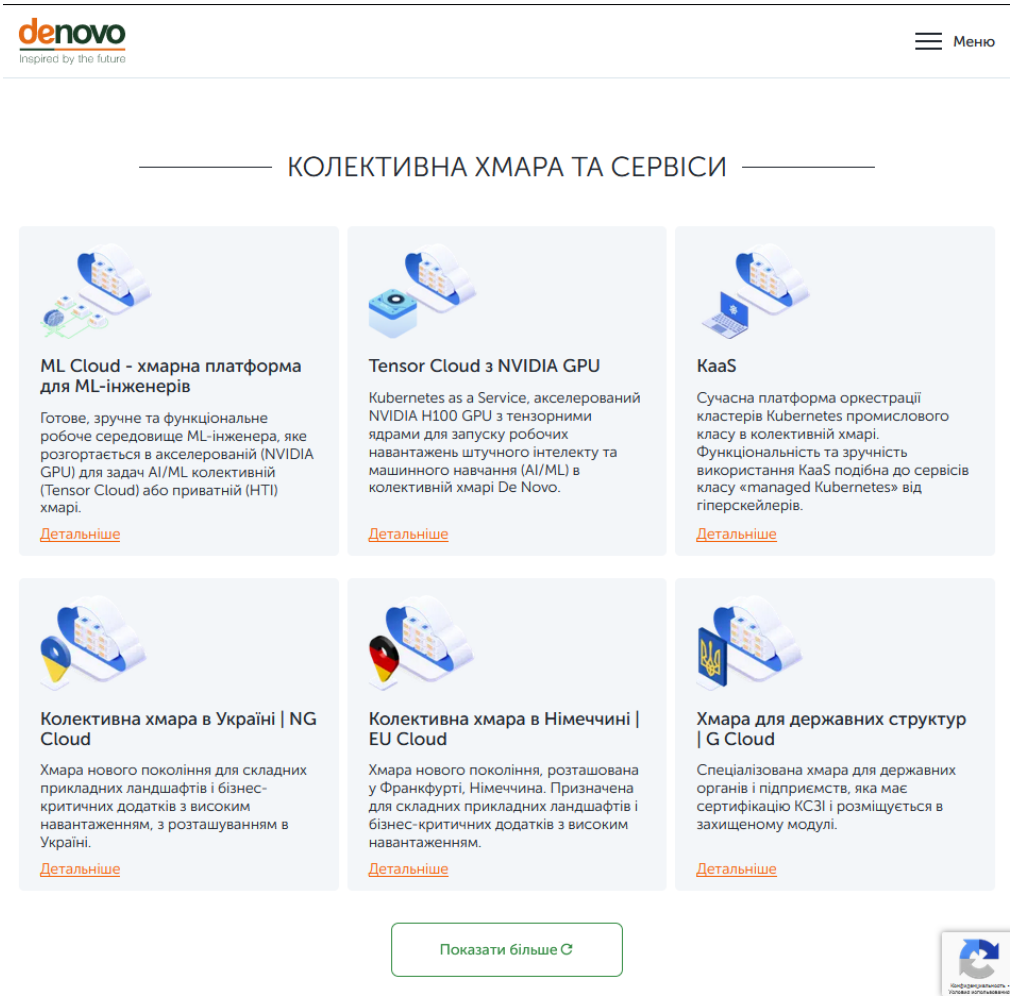

Рис. 1.7. DeNovo, список послуг

Навпроти, GigaCloud включає характеристики:

– дизайн з акцентом на швидкість доступу до послуг;
– чітко структуроване меню;
– управління хмарними ресурсами та підтримка;
– платформа для управління великими даними та аналітикою.

Отже, обидві платформи пропонують хмарні сервіси. Основна різниця полягає в інтерфейсі і функціональних можливостях, які можуть бути більш підходящими в залежності від потреб. Проте, DeNovo орієнтований більше на корпоративний сегмент.

В результаті даного аналізу, прийнято рішення виконати веб – орієнтований додаток поточного продукту для всіх користувачів, який буде структурованим і матиме мінімалістичний дизайн.

1.2. Призначення розробки та галузь застосування

В межах цієї кваліфікаційної роботи розглядається веб – орієнтований застосунок для IT – компанії «HTech», яка призначена для надання хмарних послуг та можливості придбання цифрових ліцензійних копій програмного забезпечення.

Загальною причиною появи застосунку є необхідність створення аналогу крупних компаній для використання в межах території України, та можливість купівлі забезпечення або сервісів для різноманітного призначення галузі IT.

Призначення розробки полягає в:

- продажу ліцензійного програмного забезпечення;
- створення онлайн – платформи хмарних обчислень;
- зручність покупок та використання інструментів;
- забезпечення інструментів керування списками товарів модераторам сайту;
- підтримка каталогу товарів та послуг;

Причинами створення цього застосунку є:

- відсутність місцевих аналогів. Українські IT – компанії потребують спеціалізованої платформи для забезпечення послуг то продажу продуктів через мережу, оскільки наявні рішення мають не повну відповідність з зарубіжними аналогами;
- забезпечення доступності та зручності клієнтам, що шукають хмарні обчислювальні сервіси та програмне забезпечення;
- підвищення конкурентоспроможності на ринку електронної комерції.

Області застосування можуть буть:

- продаж ліцензійного забезпечення та цифрових продуктів;
- хмарні послуги та сервіси;
- маркетплейс для IT – послуг та консультацій;
- платформа для навчання та сертифікації;

– система управління проектами;

– технічна підтримка та обслуговування.

Основна термінологія містить значення з предметної області, а саме:

– програмний інтерфейс – набір правил і протоколів для взаємодії між різними програмними компонентами;

– бекенд – це серверна сторона застосунку, яка обробляє запити від користувачів, та виконує певну логіку;

– користувацький інтерфейс – візуальне представлення даних, а також елементи, з якими взаємодіє користувач;

– система знижок – система, за якою здійснюється надавання знижок при придбанні товарів на сайті магазину;

– панель керування – система, з використанням якої здійснюється керування товарами та інформації до них;

– брандмауер – система безпеки, що фільтрує мережевий трафік.

Так, основними ключовими словами будуть: інтернет-магазин, додаток, застосунок, кошик, платіжна система, знижки, акції, веб – дизайн, база даних.

1.3. Підстава для розробки

Підставами для розробки (виконання кваліфікаційної роботи) є:

- освітня програма спеціальності 122 «Комп’ютерні науки»;
- навчальний план та графік навчального процесу;
- наказ ректора Національного технічного університету «Дніпровська політехніка» № 469-с від 23.05.2024 р;
- завдання на кваліфікаційну роботу на тему: «Розробка веб – додатку для он-лайн магазину для IT – компанії на ASP.NET».

1.4. Постановка завдання

Метою поточної кваліфікаційної роботи є створення веб – орієнтованого застосунку для IT – компанії, призначеного для надання хмарних послуг та можливості придбання цифрових ліцензійних копій програмного забезпечення. Цей додаток повинен забезпечити доступ до продуктів і послуг для компанії та клієнтів.

Призначення полягає в:

- продаж ліцензійного програмного забезпечення;
- створення онлайн – платформи для хмарних обчислень;
- забезпечення зручності покупок та використання сайту;
- надання інструментів керування списками товарів модераторам сайту;
- підтримці каталогу товарів та послуг;
- створенні навігації для пошуку інформації по сайту;
- надання документації для швидкого освоєння керування інструментарієм додатку.

Технічно – економічна сутність завдання: надати послуг хмарних обчислень, сервісів та програмного забезпечення, шляхом створення веб – орієнтованого додатку – магазину, та впровадити в масове використання. При цьому, має місце створення автоматизації процесу продажу послуг і забезпечення доступності сервісів для широкого кола користувачів, шляхом організації дружнього інтерфейсу користувача. Обґрунтування доцільності рішення базується на потребі ринку у локалізованих програмних рішеннях.

Перелік об’єктів інформаційної системи складається з:

- клієнти компанії;
- модератори та адміністратори платформи;
- сервери та інфраструктура, що необхідні для коректного функціонування застосунку.

Таким чином, структура об’єктів наступна:

– бекенд – сервер для обробки запитів та управління даними;

– графічна частина клієнта для взаємодії взаємодії клієнтів та модераторів із системою;

– бази даних для зберігання інформації про продукти, користувачів та транзакції.

Показники, що повинні відслідковуватися – це товари та їх детальний опис і структура, а також пов’язана з ними документацію до застосування. Адже опис товару це важливий компонент структури та організації сайту з інтернет - продажів.

Вихідною інформацією є дані про продукти та послуги, що надаються. Для модераторів це буде інформація про зареєстрованих клієнтів на платформі, а також інформація про додані до кошику продукти та здійснені транзакції. Також, дані про транзакції і статус оплати на сайті, після чого користувачі отримують бажану послугу або ліцензію програмного забезпечення. Окрім цього, вихідною інформацією буде:

– документація до продуктів;

– список існуючих товарів для модератора у вигляді таблиці;

– опис існуючих товарів;

– дані про клієнтів в вигляді таблиці даних;

– список елементів за результатами пошуку.

Розподіл функцій між персоналом поділяється на права до панелі адміністратор, та функціонал для всіх користувачів:

– адміністратори керують серверною інфраструктурою;

– модератори додають та редагують інформацію про продукти, клієнтів;

– користувачі здійснюють покупки та користуються сервісами.

Повний список функцій користувача включає:

– перегляд каталогу продуктів та послуг;

– додавання продукту до корзинки;

– здійснення покупок і замовлень;

– управління обліковими даними і даними корзини;

– перегляд деталей про кожен продукт;

– детальний перегляд інформації про кожне замовлення;

– розширена інформації про весь перелік обраних товарів для покупки.

Список функцій модератора:

– додавання та редагування товару каталогу;

– редагування опису товарів та структури сторінок для кожного продукту, його розділів;

– перегляд таблиці користувачів і продуктів;

– менеджмент і перегляд замовлень і транзакцій.

З ціллю кращого розуміння можливостей управління інформацією модератором, на рис. 1.8 зображено загальну концепцію макету, що відображає управління продуктами та панель адміністратора, яка виконана в середовищі «Figma».

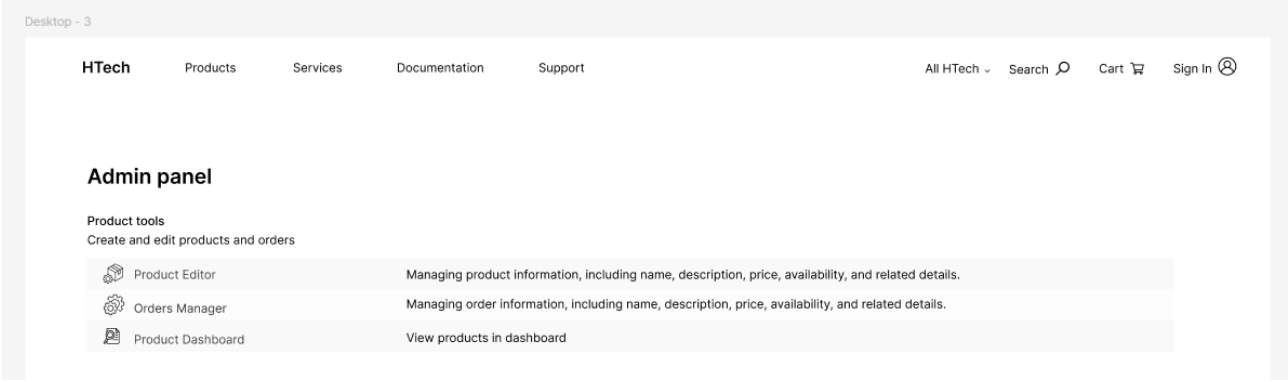

Рис. 1.8. Панель модератора, розділ управління продуктами

Створення й розробка додатку містить наступні етапи:

– аналіз вимог до веб – орієнтованого додатку;

– аналіз аналогів, їх функцій та недоліків;

– проектування таблиць баз даних для збереження її цілісності і коректної працездатності додатку;

– розгортання додатку та відповідних сервісів на серверах;

Зв’язок даної системи з іншими задачами включає пункти обробки платежів та фінансових транзакцій. Необхідна реалізація інтеграції з зовнішніми системи для забезпечення хмарних послуг, зокрема інтеграції системи платежів в додаток.

Існують умови, при яких припиняється розв’язання завдання автоматизованим способом. Це здійснюється у разі:

– технічних збоїв або недоступності серверів;

– відсутності актуальних даних про продукти або користувачів;

– невідповідність прав доступу до функцій, до котрих здійснюється запит;

– виявляення критичних помилок у роботі системи.

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

Вхідна інформація повинна оброблятися при надсиланні даних з клієнтської форми. З клієнтської частини можуть надсилати інформацію як модератори так і звичайні користувачі. Ця інформація включає в себе список продуктів та їх опис, список клієнтів і інформацію про них. Дані, що були оброблені клієнтом, надсилаються до відповідного контролеру. У випадку невідповідності даних, з’являється відповідне повідомлення на клієнтській частині.

Організація вихідних даних повинна відбуватися за допомогою сервісів програми, що заповнюють дані в об’єкти та передають їх до представлень, які повертаються користувачам на їх запити до конкретних ресурсів.

Виконувані функції мають включати:

– перегляд каталогу програмного забезпечення;

– додавати обрані продукти до кошика;

– здійснювати оплату і отримувати ліцензійні ключи або завантажувальні файли;

– забезпечувати доступ до хмарних обчислювальних ресурсів;

– модератори повинні мати можливість додавати нові товари, редагувати існуючи, змінювати їх опис та налаштування категорії товарів;

Вимоги до часових характеристик:

– Обробка запитів користувачів. Застосунок повинен забезпечувати обробку в режиме реального часу, з мінімально можливою затримкою.

– Оновлення даних. Дані про продукти та послуги повинні оновлюватися в режимі реального часу після внесення змін модераторами.

Представлення даних має наступні характеристики:

– Каталог продуктів. Дані відображаються у вигляді карток продуктів з детальним описом. Для кожного окремого продукту існує унікальна структура сторінки опису, з кнопками додавання до кошика.

– Дані для модератора надаються у вигляді таблиць, з детальною інформацією про кожний елемент.

1.5.2. Вимоги до інформаційної безпеки

Аналізуючи джерела на предмет інформаційної безпеки, має місце твердження, що інформаційна безпека – це практика, що зосереджена на захисті даних від несанкціонованого доступу, використання, розголошення, порушення, зміни або їх знищення [10 – 12]. При чому, дані можуть бути як електронними, так і фізичними.

Основною метою інформаційної безпеки є забезпечення конфіденційності, цілісності та доступності даних.

В поточному проекті, обов’язковою є реалізація переліку методів захисту інформації, а саме:

– Процес аутентифікації. Здійснюється з метою, щоб підтвердити особу користувача, та здійснити вхід до системи під його обліковими даними та потенційними привілеями.

– Авторизація. Цей процес повинен виконуватися кожного разу, коли користувач здійснює запит до чутливих даних, або до тих даних, де наявний певний рівень доступу. Наприклад, якщо користувач намагається здійснити запит до панелі адміністратора, то необхідно здійснити перевірку відповідності його прав до виконуваної дії.

З ціллю забезпечення додаткових елементів безпеки для кожного користувача, необхідно ввести:

– обмеження кількості мінімальних символів у паролі;

– шифрування паролів в базі даних користувачів, та надання їх у вигляді хешів;

– обов’язкова наявність символів верхнього та нижнього регістрів у паролі.

Хешування паролів є важливим кроком для забезпечення цілісності системи, та захисту користувачів від несанкціонованого доступу. В цілому хешування – це процес перетворення паролю у фіксовану довжину рядка, який не можна перетворити назад в оригінальний пароль [13]. При цьому, для даного процесу використовуються алгоритми хеш -функції, такі як bcrtpy, Argon2 або scrypt. Також, гарною практику захисту буде соління [14], тобто процес, коли пароль поєднується з випадковим значенням перед хешуванням. Це надасть додатковий рівень захисту в системі.

З ціллю безпеки чутливих даних, дані користувачів та транзакції мають зберігатися на окремому сервері, та мати обмежений доступ з мережі.

Важливим пунктом є приведення таблиць бази даних до третьої форми нормалізації задля збереження цілісності даних, та уникання можливих невідповідностей даних.

1.5.3. Вимоги до складу та параметрів технічних засобів

Для коректного функціонування всіх компонентів системи, необхідно використати наступне обладнання для серверів:

– Процесор. Мінімум 8 ядер, частота не менше 3.0 ГГц.

– Оперативна пам’ять. Мінімум 32 ГБ DDR4.

– Накопичувач. SSD з об’ємом не менше 1 ТБ.

– Мережевий інтерфейс: Gigabit Ethernet, підтримка 10 Gigabit Ethernet є бажано.

Тоді мережеве обладнання повинно мати:

– Маршрутизатори. Підтримка протоколів IPv4/IPv6, можливість налаштування VPN.

– Комутатори. Підтримка VLAN, мінімум 24 порти Gigabit Ethernet.

– Брандмауери. Підтримка NAT, контроль доступу, можливість налаштування фільтрації трафіку.

Система зберігання даних:

– NAS. Підтримка RAID, об’єм не менше ніж 10 ТБ.

– SAN. Підтримка протоколів iSCSI, FC, об’єм не менше 20 ТБ.

В свою чергу, клієнтське обладнання має мати наступні характеристики:

– Процесор. Мінімум 4 ядра, частота не менше 2.5 ГГц.

– Оперативна пам’ять. 2 ГБ і вище.

– Накопичувач. HDD або SDD з об’ємом, необхідним для функціонування операційної системи.

– Графічний процесор. Відеокарта з підтримкою OpenGL або DirectX, мінімум 2 ГБ відеопам’яті.

1.5.4. Вимоги до інформаційної та програмної сумісності

Додаток розроблено на платформі ASP.NET, що забезпечує сумісність з іншими продуктами на базі Microsoft та підтримує інтеграцію з різними базами даних та зовнішніми сервісами. Взаємодія між клієнтом і сервером здійснюється у форматі JSON, конфігураційні файли використовують формат XML.

Основною мовою програмування для серверної частини є C#, а для взаємодії з базою даних використовується Entity Framework. Клієнтська частина використовує JavaScript для динамічних функцій та взаємодії з сервером через AJAX, а також HTML та CSS для створення структури та стилізації веб-сторінок.

Таким чином, основні вимоги:

- .NET Core 6.0;
- C# 8.0 або новіша версія;
- JavaScript (для клієнтської частини) та сучасний браузер для виконання скриптів;
- Entity Framework 6.0 для роботи з базою даних

Вимоги до операційної системи включають наявність OS Linux для серверу з вільними дистрибутивами, що підтримують вище зазначені характеристики.

Для клієнтською частини бажано використовувати операційну систему Windows 10 або вище.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ТА РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1. Функціональне призначення системи

Веб – орієнтований застосунок інтернет магазину для IT компанії призначений для діяльності компанії, та надання послуг онлайн. Виходячи з цього, функціональне призначення додатку включає:

- ведення бази даних про товари, включаючи інформацію про їхні характеристики, розділи, опис, наявність на складі;
- управління каталогом товарів, що дозволяє користувачам переглядати нові продукти, а модераторам додавати їх і переглядати детальну інформацію;
- забезпечення аутентифікації та авторизації користувачів у системі;
- реалізація функцій кошика для покупок та оформлення замовлень, включаючи інтеграцію з платіжними системами для безпечної оплати онлайн;
- управління замовленнями та відстеження їхнього статусу, отримання актуальної інформації про закази;
- ведення документації та сторінок з описом кожного окремого продукту.

Експлуатаційне призначення системи включає:

- редактор для додавання, редагування та видалення товарів, а також для управління описами і розділами продуктів та сервісів;
- інтерфейс керування покупками, включаючи можливість роботи з корзинкою та оформлення платежів;
- автоматизування процесу управління сервісами;
- забезпечення перегляду даних про користувачів за їх замовлення використовуючи панель модератора;
- можливості ведення профілю і зміни даних користувача, перегляду історії покупок;

2.2. Опис застосованих математичних методів

Оскільки специфіка розв’язуваної задачі не передбачає використання математичних методів, у процесі розробки веб – орієнтованого застосунку для інтернет - магазину ІТ компанії математичні методи не застосовуються.

2.3. Опис використаних технологій та мов програмування

2.3.1. Клієнт – серверна архітектура

В даному проекті використовується клієнт - серверна архітектура. Користувач надсилає HTTP [15] запити до контролерів з клієнтів (браузерів), сервер (бекенд) обробляє ці запити, та повертає HTTP – відповідь у вигляді сторінки з даними.

Модель «клієнт-сервер» описує взаємодію програм, які співпрацюють у додатку. Серверний компонент надає функції або послуги клієнтам, які надсилають запити на ці послуги. Сервери можуть бути різних типів, залежно від послуг, які вони надають: наприклад, веб-сервери обслуговують веб-сторінки, а файлові сервери зберігають і обробляють файли. Серверні ресурси можуть включати програмне забезпечення, дані, процесори та пристрої зберігання [16].

Комп’ютер може бути клієнтом, сервером або обома одночасно, залежно від потреб програми. Наприклад, один комп’ютер може одночасно працювати як веб-сервер і файловий сервер, надаючи різні дані клієнтам з різними запитами. Клієнтські програми також можуть обмінюватися даними з серверними програмами на одному комп’ютері. Зв’язок між серверами для синхронізації даних іноді називають міжсерверною комунікацією.

Загальний концепт містить твердження, що клієнт – це комп’ютерна програма, наприклад веб – бразуер, що працює на локальному пристрої користувача, такому як комп'ютер, смартфон чи інший гаджет, і підключається до сервера, коли це потрібно. Операції можуть виконуватися на клієнтському боці,

оскільки вони потребують доступу до даних або функцій, що є на клієнті, але не на сервері. Це може бути зумовлено тим, що користувач повинен мати змогу спостерігати за процесом або вводити дані, або ж через недостатню потужність сервера для своєчасного обслуговування всіх клієнтів. Окрім цього, якщо операції можна виконати на клієнтському пристрої без передачі даних через мережу, це може зекономити час, зменшити використання пропускної здатності і підвищити безпеку.

Якщо сервер надає дані за стандартними протоколами, такими як HTTP або FTP [17], користувачі можуть використовувати різні клієнтські програми (наприклад, будь-який сучасний веб-браузер може отримувати дані через HTTP чи FTP). У випадках, коли програми є спеціалізованими, розробники можуть створювати власні сервери, клієнти та протоколи для взаємодії між ними.

Нижче наведену схему клієнт – серверної архітектури (рис. 2.1).

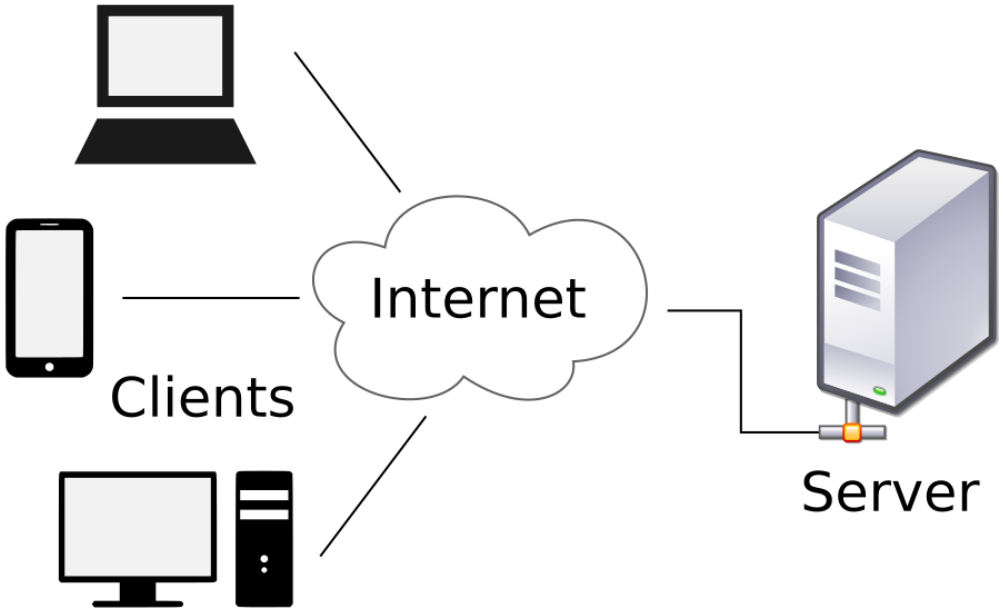

Рис. 2.1. Схема мережі комунікації клієнтів з сервером

2.3.2. Фреймворк ASP.NET

Фреймворк, що використовується для розробки веб – орієнтованого в поточному проекті – ASP.NET [18]. Він має основне призначення для розробки веб – застосунків та динамічних сторінок, використовуючи HTML, CSS та JavaScript. Також, цей фреймворк є кросплатформенним, тобто може використовуватися для створення веб – застосунків, які працюють на різних операційних системах, таких як Windows, macOS та Linux.

ASP.NET пропонує три структури для створення веб-додатків: веб-форми, ASP.NET MVC і веб-сторінки ASP.NET. Усі три фреймворки є стабільними, і надають можливість створювати веб-додатки. Кожна структура націлена на інший стиль розробки.

Усі три фреймворки ASP.NET базуються на .NET Framework, та мають спільні основні функції .NET і ASP.NET. Наприклад, усі три структури пропонують модель безпеки входу, засновану, і використовують можливості для керування запитами, обробки сеансів, які є частиною основної функціональності платформи. Крім того, ці фреймворки не є повністю незалежними, і вибір одного не виключає використання іншого. Ці інструменти можуть співіснувати в одній веб-програмі, та бути окремими компонентами програм, написані з використанням різних фреймворків. Наприклад, частини програми, орієнтовані на клієнта, можуть бути розроблені в MVC [19], тоді як частини доступу до даних і адміністративні частини розроблені у веб-формах, щоб скористатися перевагами керування даними та простого доступу до даних.

В даній роботі для досягнення поставлених задач використовується MVC інструментарій ASP.NET для створення додатку. Структуру фреймворку зображено на рис. 2.2.

Процес починається, коли користувач переходить на веб-сайт або вводить URL-адресу у своєму браузері. Навігація за адресою в браузері надсилає запит із комп’ютера користувача на сервер, на якому розміщено веб-програму, за допомогою протоколу HTTP. Запит може бути отриманий і повторно переданий

на кількох маршрутизаторах по дорозі, але лише коли він дійде до сервера, пов’язаного з ім’ям хоста, запит обробляється. Залежно від запиту, контролер оброблює дані що були отримані, та генерує сторінку, що буде відображена користувачу, на стороні серверу.

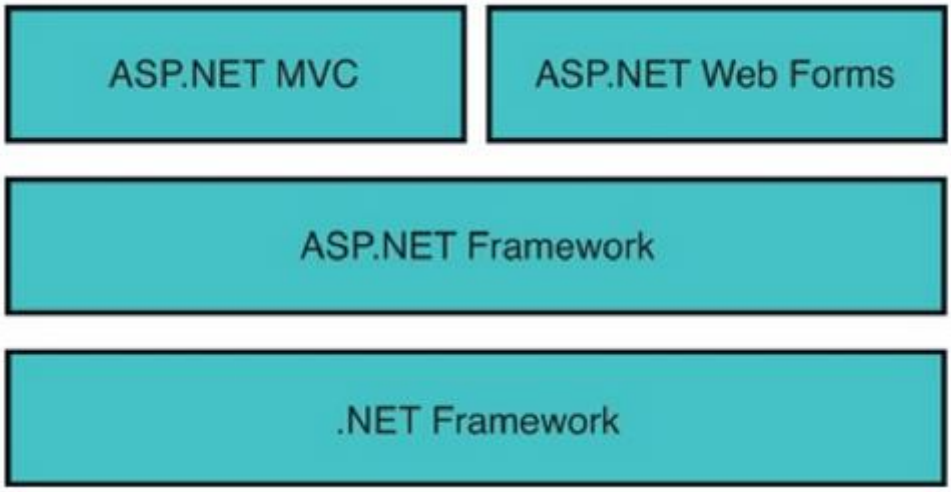

Рис. 2.2. ASP.NET фреймворки

2.3.3. Архітектурний шаблон MVC

В даному контексті, шаблон проектування MVC (Model – View - Controller) використовується в рамках платформи та інструментів ASP.NET – технології для побудови веб -застосунків. Даний патерн працює за наступним принципом:

– Модель. Містить логіку даних всього додатку, що не має представлення або контролер. Часто об’єкти моделі отримують та зберігають стан моделі в базі даних. Наприклад, в випадку поточного проекту, об’єкт Product може отримувати дані з бази даних та оперувати ними. У невеликих програм модель є концептуальним розділенням замість фізичного. У випадку, якщо програма лише зчитує дані та надсилає їх до представлення, програма не має рівня фізичної моделі, та пов’язаних класів. Тоді, набір даних бере на себе роль модельного об’єкту.

– Представлення. Це компоненти, які відображують інтерфейс користувача. До представлення передаються дані, які повинні бути відображені.

– Контролери. Опрацьовують взаємодіє користувача з програмним додатком, а також працюють з моделю даних. Обирають представлення для відображення інтерфейсу користувача.

Згідно концепції MVC (рис. 2.3), представлення можуть тільки відображувати інформацію вигляді, що передається [20].

На додаток до керування складністю, шаблон MVC спрощує тестування програм у порівнянні з тестуванням веб-додатків ASP.NET, що базуються на веб-формах. У веб-додатку ASP.NET, побудованому на веб-формах, один клас відповідає як за відображення вихідних даних, так і за обробку введення користувача. Це ускладнює написання автоматизованих тестів, оскільки для тестування окремої сторінки потрібно створити екземпляр класу сторінки, всі його дочірні елементи керування та додаткові залежні класи. Через необхідність створення великої кількості класів для запуску сторінки важко писати тести, які зосереджуються на окремих частинах програми. Тому тести для веб-додатків ASP.NET на основі веб-форм можуть бути складнішими у впровадженні, ніж тести у додатку MVC. Крім того, тести для програм ASP.NET на основі веб-форм вимагають наявності веб-сервера. Фреймворк MVC розділяє компоненти та активно використовує інтерфейси, що дозволяє тестувати окремі компоненти незалежно від решти фреймворку.

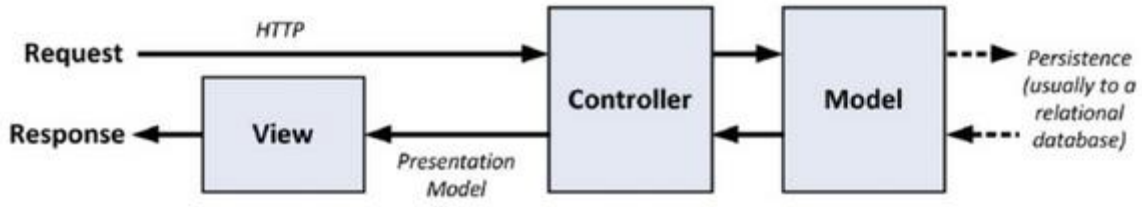

Рис. 2.3. Взаємодії в архітектурі MVC

2.3.4. Сховище даних та Entity Framework

В якості сховища даних було обрано базу даних SQLite [21], що є вбудованою реляційною базою даних з відкритим кодом. Це сховище розроблено для забезпечення зручності керування даними без накладних витрат, які часто приходять із виділеними системами керування базами даних. Замість того, щоб працювати як окремий процес, він інтегрується безпосередньо в додаток, який обслуговує, і виконується в його процесі. Код бази даних стає невід'ємною частиною програми Програма виконує свої функції та керує даними без додаткових вказівок на те, як це відбувається. Всередині працює повноцінна база даних. Одна з переваг такого підходу полягає в тому, що немає потреби конфігурувати або адмініструвати мережу. Це значно полегшує роботу: не потрібно турбуватися про брандмауери, мережеві адреси або складні дозволи та привілеї. Клієнт і сервер працюють разом в одному процесі, що зменшує накладні витрати на мережеві виклики, спрощує адміністрування бази даних і полегшує розгортання додатку. Усі необхідні компоненти включені прямо в програму.

В контексті поточного проекту, Entity Framework (EF Core) дозволяє маніпулювати даними в SQLite, надаючи зручні методи для роботи з базою даних.. EF Core виступає в якості ORM (Object-Relational Mapper), що дозволяє працювати з даними, використовуючи об’єкти та класи .NET [22 - 24].

Об’єктно – реляційне відображення (ORM) – це програмний шаблон, який використовується для приховування складнощів взаємодії з реляційної базою даних з об’єктно – орієнтованої програми [25].

ORM робить це, шляхом створення «віртуальної об’єктної бази даних», яка представляє реляційну базу даних у вигляді об’єктів. Це дозволяє писати код, який використовує об’єкти для доступу до даних в БД, не застосовуючи SQL – запити.

ORM пропонують такі функції:

– Відповідність. Автоматичне відображення таблиць БД на класи та стовпці таблиць на властивості класів.

– Надає методи для створення, читання, оновлення та видалення даних.

– Запити. Дозволяє розробникам писати запити до даних за допомогою об’єктно – орієнтованої мови, при цьому не використовуючи SQL.

– Автоматично завантажує пов’язані дані з бази даних, коли вони потрібні.

Переваги даного методу полягають у:

– зменшенні програмного коду;

– полегшенні розробки та обслуговування програмного забезпечення, зокрема через покращення читабельності коду, таким чином зменшуючи ймовірність помилок.

Однак недоліками є швидкість роботи. Ручне використання SQL може бути швидшим. Також, приховуються деталі роботи бази даних, що ускладнює її налагодження.

Аналізуючи джерела на предмет порівняння працездатності Entity Framework[26], логіка, яка переводить методи в еквівалентний SQL код потребує пам’яті. Наприклад EF Core використовує відображення, тому перший виклик буде повільнішім і займатиме більше часу ніж решта, через використання кешування. Інший аналог, а саме Depper [27], що використовувався для порівняння, не використовував відображення, але має логіку для зіставлення результатів SQL, який був згенерований та запущений, а також очікуваних сутностей. Ще один аналог ORM, а саме – Hibernate [28] також використовує перший рівень кешу для оптимізації, але відображення за лаштунками позначать споживання пам’яті. Виходячи з досліджень, у якому перший сценарій представлений метод вставки даних, що передбачає вставлення сутностей, які мають зв’язки один – до – одного, а другий – представлений методом вставки, що окрім один – до – одного, мають зв’язок один – до – багатьох, результати яких зображені на рис. 2.4. На даній діаграмі, EF Core знаходиться на другому місці

по використанню пам’яті при здійснені операцій, порівняно з його конкурентами, що є гарним показником.

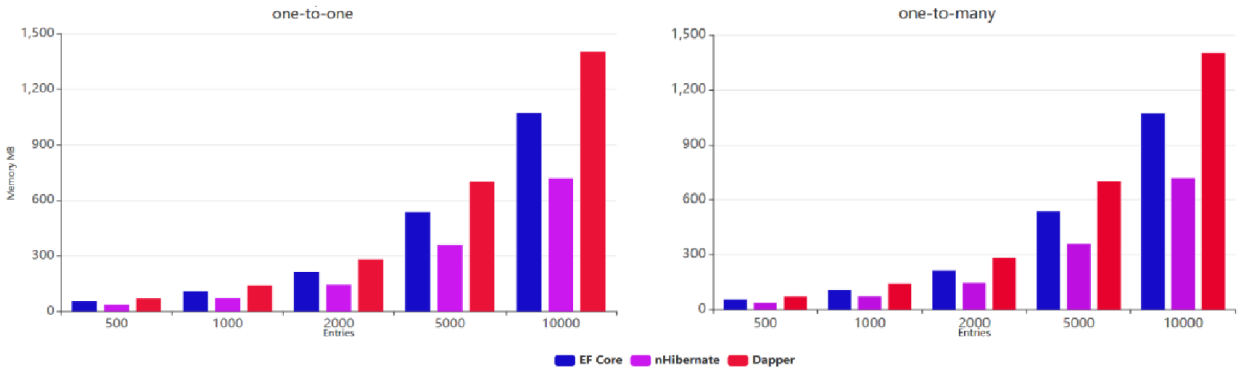

Рис. 2.4. Показники затраченої пам’яті при використанні операції INSERT

Таким чином, використання Entity Framework є ефективним рішенням, порівняно з іншими конкурентами на ринку, в поточному контексті.

2.3.4. Використані мови програмування

При розробці використано переважно мову програмування C#, що реалізує архітектуру системи шаблона проектування MVC.

C# є мовою програмування загального призначення, яка підтримує множинну парадигму [29]. Дана мова має зв’язок з сімейством мов C та підтримує компонентно – орієнтоване програмування. Ключовим для компонентів є те, що вони представляють модель програмування з властивостями, методами та подіями; вони мають атрибути, які надають декларативну інформацію про компонент; і вони включають власну документацію. C# надає конструкції для прямої підтримки цих концепцій.

Наступні функції C# допомагають у створенні надійних та стійких програм:

– Збірник сміття (Garbage collection) автоматично звільняє пам’ять, зайняту невикористовуваними об’єктами.

– Обробка вийнятків надає структурований та розширюваний підхід до виявлення помилок та відновлення.

– Типобезпечний дизайн мови робить неможливим читання з неініціалізованих змінних, індексування масивів за межами їхніх кордонів, або виконання непровірених приведень типів.

C# має уніфіковану систему типів, та всі типи, включаючи примітивні (такі як int або double) успадковують від єдиного кореневого об’єктного типу. Таким чином, всі типи мають набір спільних операцій, а значення будь – якого типу можна зберігати, транспортувати та оперувати узгодженим способом. Крім того, мова підтримує визначені користувачем типи посилань, так і типи значень, що дозволяє динамічно виділяти об’єкти, а також зберігати легкі структури в рядку.

Окрім цієї мови, для динамічних елементів на сторінці представлення, окрім частини яка генерується сервером, було використано мову програмування JavaScript.

JavaScript є сценарною мовою програмування, яка дозволяє впроваджувати складні функції на веб – сторінках – щоразу, коли веб-сторінка не просто відображує статичну інформацію, але й своєчасно оновлює вміст, інтерактивні карти, або анімовані елементи. В цілому JavaScript дозволяє динамічно оновлювати контент, контролювати мультимедіа, анімувати картинки

Окрім цього, в проекті використовуються стилі CSS – декларативна мова для контроля вигляду сторінка в веб – браузері.

Браузер приймає стилі до обраних елементів, щоб відобразити їх описаними правилами. Декларація містить властивості та їх значення. CSS є ключовою технологію в веб – розробці між HTML та JavaScript

Для розробки маркування сторінок, використовувався HTML 5.

2.3.5. Технологія Bootstrap

У межах цієї кваліфікаційної роботи, Bootstrap [30] використовується частково, переважно для реалізації слайдерів, або розбиття контейнерів на блоки, з ціллю зниження кількості коду та приведення його до певного стиля, що сприяє кращому розумінню і ефективному процесу розробки.

Bootstrap можна використовувати в проекті як локально, так і через посилання на онлайн ресурси (CDN). В цьому проекті, використовується другий варіант. Основними його перевагами є швидке завантаження, оскільки фали можуть бути кешованими у користувача, а також зручна інтеграція через відсутність необхідності завантаження файлів.

Цей фреймворк надає глобальні стилі, та змінення стандартного відображення документу. В набір цих інструментів входить як компоненти CSS, так і JavaScript, а також утиліти і іконки [31].

Основною перевагою є система сітки та гнучкий дизайн. Сіткова система Bootstrap використовує контейнери, рядки та стовпці для компонування та вирівнювання контенту. Вона побудована на основі flexbox і повністю адаптивна.

Реагуючий дизайн – це підхід, що дозволяє оптимізувати відображення контенту на різних пристроях. Наприклад, для настільних комп'ютерів контент може бути оптимізований для великих екранів, які часто підключаються до комп'ютерів. Планшети отримують макет, оптимізований для портретної або альбомної орієнтації. На смартфонах використовується вужчий макет, що відповідає їх ширині екрана.

Для налаштування на різні ширини екранів Bootstrap використовує медіа-запити CSS, які вимірюють ширину вікна браузера і умовно завантажують відповідні стилі. Це дозволяє оптимізувати контент за допомогою властивостей min-width та max-width.

Основою Bootstrap є п’ять різних макетів, кожен з яких базується на медіа-запитах CSS. Найширший макет має стовпці шириною 70 пікселів, порівняно з 60 пікселями у стандартному макеті. Макет для планшетів має стовпці шириною

42 пікселі, а на ще вужчих пристроях стовпці стають гнучкими та займають повну ширину екрана, складаючись вертикально. (рис 2.5).

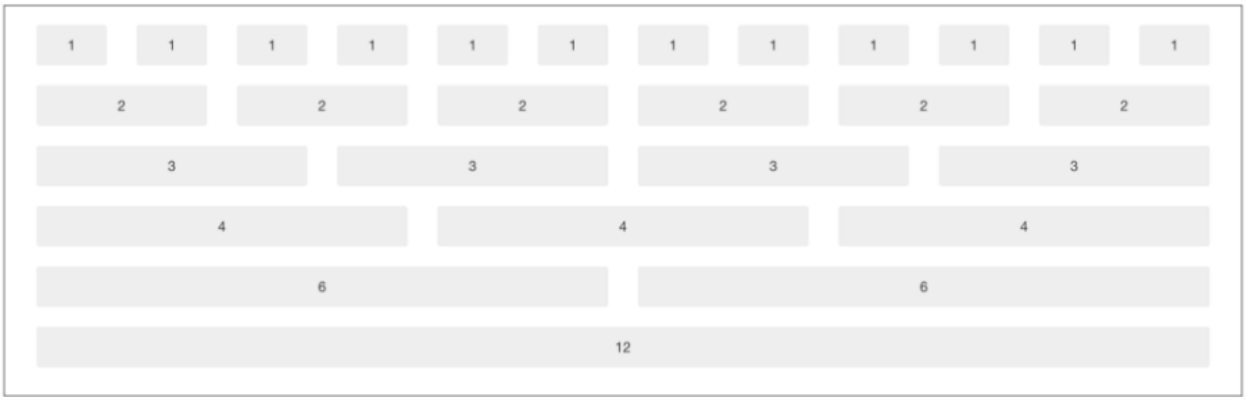

Рис. 2.5. Стандартна сітка

2.3.5. Інтеграція платіжної системи PayPal

PayPal – це онлайн-сервіс електронних платежів, що функціонує як обліковий запис, дозволяючи переказувати гроші за допомогою електронної пошти. PayPal може використовуватися як міжнародна платіжна система для підприємств і споживачів, що здійснюють торгові операції між різними країнами [32].

З аналізу зібраних даних [33], що PayPal, як електронна платіжна система, є однією з провідних компаній на ринку міжнародних онлайн-платежів. Вона ефективно вирішує питання споживачів і торговців щодо доставки, митних зборів, податків, безпечних методів оплати та захисту міжнародних торговельних операцій. PayPal також пропонує доступну та недорогу інфраструктуру для управління готівкою, що робить її привабливою для невеликих компаній, дозволяючи їм швидко та ефективно комерціалізувати свої продукти на світовому ринку.

До березня 2022 року українські користувачі PayPal могли лише відправляти платежі з карток VISA та MasterCard, випущених українськими банками, та оплачувати товарі та послуги в інтернеті. Проте, отримувати платежі

на рахунки через PayPal було неможливо, оскільки сервіс не підтримував надходження коштів в Україну [34].

17 березня 2022 року компанія розпочала діяльність в Україні, з метою надання підтримки громадянам. З цього моменту, існує можливість здійснювати P2P перекази та здійснювати вивід коштів на картки українських банів, які прив'язані до особистого рахунку у PayPal.

В даній роботі використовується інтеграція платіжного сервісу PayPal, а саме GooglePay [35], з метою здійснення оплати продуктів інтернет – магазину.

2.3.6. Методологія BEM

Слід зазначити, що в проекті використовується BEM [36] організації класів CSS та структури HTML файлів.

BEM (Block, Element, Modifier) — це компонентний підхід до веб-розробки. Ідея полягає в тому, щоб розділити інтерфейс користувача на незалежні блоки. Це робить розробку інтерфейсу легкою та швидкою навіть із складним інтерфейсом користувача, а також дозволяє повторно використовувати наявний код без копіювання та вставки.

Методологія BEM використовує підхід, який розповсюджується на структуру файлів проекту. Кожен блок, елемент і модифікатор має свій власний технологічний файл, що дозволяє нам з'єднувати їх окремо.

Основні особливості цього підходу:

– Кожен блок має свій власний каталог.

– Назва блоку і каталогу співпадає. Наприклад, блок заголовка знаходиться у каталозі header/, а меню у каталозі menu/.

– Реалізація блоку розділена на окремі технологічні файли, наприклад, header.css і header.js.

– Каталог блоку є основним для підкаталогів його елементів і модифікаторів.

– Назви підкаталогів елементів починаються з подвійного підкреслення.

– Назви підкаталогів модифікаторів починаються з одного підкреслення.

– Реалізації елементів і модифікаторів також розділені на окремі технологічні файли.

Блоки у BEM є головними компонентами конструкції. Вони представляють собою окремі частини, які містять всю необхідну HTML і CSS для свого функціонування. Наприклад, блоком може бути заголовок, навігаційне меню або секція з відповідним вмістом [37].

Назва блоку повинна відображати його призначення і бути унікальною в рамках проекту.

Блоки ідентифікуються за допомогою класу, який починається з імені блоку і має два підкреслення.

Завдяки умовам іменування BEM легко зрозуміти, що робить кожен клас і як він пов’язаний з іншими елементами на сторінці. Це полегшує роботу з кодом.

2.4. Опис структури системи та алгоритмів її функціонування

2.4.1. Алгоритми та структура проекту

З ціллю опису загальної структури було використано декомпозицію. Даний метод дозволяє розбити систему проекту на підсистеми, та на більш малі елементи структури, і візуалізує основні компоненти та взаємозв’язки між ними. Кількість рівнів визначена згідно усіх можливих компонентів та підсистем застосунку. Відображення здійснено за допомогою таблиці (табл. 2.1).

Таблиця 2.1

Декомпозиція системи

<table>
<tr><td rowspan="14">Інтернет-магазин</td><td rowspan="5">Продукти</td><td rowspan="4">Каталог</td><td rowspan="4">Сервіс або продукт</td><td rowspan="4">Сторінка продукту</td><td>Придбання</td></tr>
<tr><td>Реєстрація</td></tr>
<tr><td>Документація</td></tr>
<tr><td>Навігація</td></tr>
<tr><td colspan="4">Опис товарів</td></tr>
<tr><td colspan="4">Пошук продуктів/сервісів</td></tr>
<tr><td rowspan="9">Кошик</td><td rowspan="3">Товари</td><td>Деталі товару</td><td colspan="2">Сторінка деталей про продукт / сервіс</td></tr>
<tr><td colspan="3">Кількість товару</td></tr>
<tr><td colspan="3">Сумарна вартість за кількістю</td></tr>
<tr><td rowspan="2">Знижки</td><td colspan="3">Промо коди</td></tr>
<tr><td colspan="3">Активація знижок</td></tr>
<tr><td rowspan="4">Система платежу</td><td rowspan="4">Інтеграція PayPal</td><td colspan="2">GooglePay(VISA, Mastercard)</td></tr>
<tr><td rowspan="3">Акаунт продавця</td><td>Ідентифікатор реквізитів</td></tr>
<tr><td>Секретний код</td></tr>
<tr><td>Ідентифікатор профілю</td></tr>
</table>

Продовж. табл. 2.1

<table>
<tr><td></td><td></td><td></td><td></td><td colspan="3">Система здійснення та перевірки оплати</td></tr>
<tr><td rowspan="13"></td><td rowspan="5">Вхід в систему / реєстрація</td><td colspan="5">Видалення товарів</td></tr>
<tr><td>Редагування товару</td><td>Сторінка редагування</td><td colspan="3">Кількість одиниць</td></tr>
<tr><td rowspan="2">Аутентифікація</td><td colspan="4">Перевірка даних форми</td></tr>
<tr><td>Підтвердження особи</td><td colspan="3">Логін, пароль, емейл</td></tr>
<tr><td>Авторизація</td><td>Перевірка ролей</td><td colspan="3">Роль користувача</td></tr>
<tr><td></td><td></td><td colspan="3">Роль модератора</td></tr>
<tr><td rowspan="8">Адміністративна панель</td><td rowspan="4">Створення користувача</td><td rowspan="2">Клієнт</td><td rowspan="2">Сформовані дані</td><td>Логін</td></tr>
<tr><td>Пароль</td></tr>
<tr><td>Емейл</td></tr>
<tr><td>AJAX запит</td><td>Дані користувача</td></tr>
<tr><td rowspan="3">Сервер</td><td rowspan="3">Прийом запиту</td><td>Валідація даних</td></tr>
<tr><td>Створення користувача</td></tr>
<tr><td>Відповідь</td><td>Результат</td></tr>
<tr><td>Авторизація</td><td colspan="3">Перевірка ролей</td></tr>
<tr><td rowspan="6">Менеджер продуктів</td><td rowspan="6">Додавання /Оновлення продукту</td><td colspan="2">Назва товару</td></tr>
<tr><td colspan="2">Опис</td></tr>
<tr><td colspan="2">Опис в каталозі</td></tr>
<tr><td colspan="2">Статус товару</td></tr>
<tr><td colspan="2">Тип продукту</td></tr>
<tr><td colspan="2">Ціна продукту</td></tr>
</table>

Закінчення табл. 2.1

			Додавання/ Оновлення опису сторінки товару	Ідентифікатор продукту
				Назва секції
				Порядок секції
				Опис секції
		Менеджер замовлень	Таблиця замовлень	
		Продукти	Таблиця продуктів	
		Менеджер користувачів	Додавання/ Оновлення користувача	Логін, пароль
				Емейл
				Ідентифікатор користувача
				Роль
		Рольові групи	Таблиця ролей	
		Користувачі	Таблиця користувачей	
	Пошук	Система пошуку та відображення		

Алгоритм дії програми проекту відповідає архітектурі MVC. Користувач виконує запити сторінок, таким чином ініціюючи контролери викикати дії відповідні до сторінки, якої звернувся користувач.

Так, наприклад, аутентифікація виконується наступним чином:

```
[HttpPost]
public IActionResult Authorization([FromBody] UserViewModel userModel)
{
   AuthenticationService service = new(userModel, _userService);

   if (!service.IsUserDataValid())
   {
      return BadRequest(ModelState);
   }

   _session.SetInt32("UserId", service.GetAuthenticatedUserId());

   return Ok();
```

}

В цьому коді, контролер приймає запит з даними форми, які зберігають інформацію користувача для реєстрації. Викликається сервіс, що обробляє процес аутентифікації, встановлюється змінна сесії, і користувачу повертається відповідь.

Використовуючи AJAX [38] дані надсилаються на контролер методом POST, попередньо дані запиту підготовлюються для відправки форми.

Додаток використовує інтеграцію з PayPal на сторінці оплати (рис 2.6). Механізм здійснення оплати працює наступним чином:

– Створення кнопки PayPal. На сайті існує контейнер для кнопки, котра ініціалізується з використанням JavaScript.

– Обробка замовлення на клієнтський стороні та повернення його ідентифікатора.

– На клієнтській сторони користувач здійснює оплату Google Pay, використовуючи при цьому VISA [39] або Mastercard [40].

– Підтвердження оплати та відображення статусу.

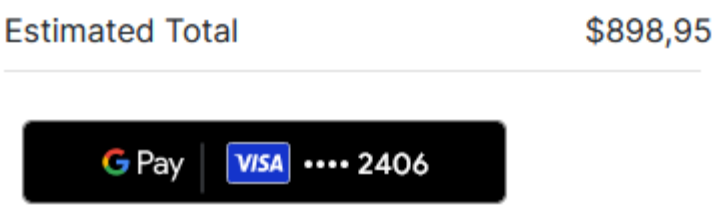

Рис. 2.6. Кнопка оплати GooglePay

Окрім цього, слід зазначити алгоритм роботи системи додавання товарів та змінення їх опису: модератор додає дані, вводячи їх в відповідні поля на сторінці редактора продуктів (рис. 2.7). Далі, є можливість додати опис до цього сервісу або продукту.

Дані, що були занесені, записуються до відповідних таблиць бази даних, та при здійсненні запиту до представлення переліку продуктів виконується обробка отриманих елементів таблиць. За дану обробку відповідають сервіси кожної з таблиць, а саме інтерфейси, функціонал котрих реалізується для здійснення маніпуляції з даними. Вигляд інтерфейсу зображено на рис 2.8. Даний підхід забезпечує гнучкість та можливість легкої заміни реалізації на іншу. В цілому, сервісний шар позволяє виконати абстракцію та ізолювати роботу с даними від контролерів.

Далі, цей сервіс реєструється в контейнері залежностей програми, в відповідній частині, що виконує конфігурацію. Таким чином, загальний алгоритм дій складає:

- визначення контракту для взаємодії з продуктами;
- реалізація методів для взаємодії з базою даних;
- реєстрація сервісів в контейнері залежностей;
- використання логіки сервісу в контролері шляхом виконання ін’єкції.

```
using WebApplication1.Models;

namespace WebApplication1.Services
{
    Ссылок: 10
    public interface IProductService
    {
        Ссылок: 2
        void CreateProduct(Product product);
        Ссылок: 1
        public Product GetProductById(int productId);

        Ссылок: 3
        public List<Product> GetAllProducts();

        Ссылок: 1
        public void UpdateProduct(Product product);
        Ссылок: 1
        public void DeleteProduct(int productId);
    }
}
```

Рис. 2.8. Сервіс продуктів

Як приклад використання даної системи, використано редактор додавання продуктів, що зображено на рис. 2.9.

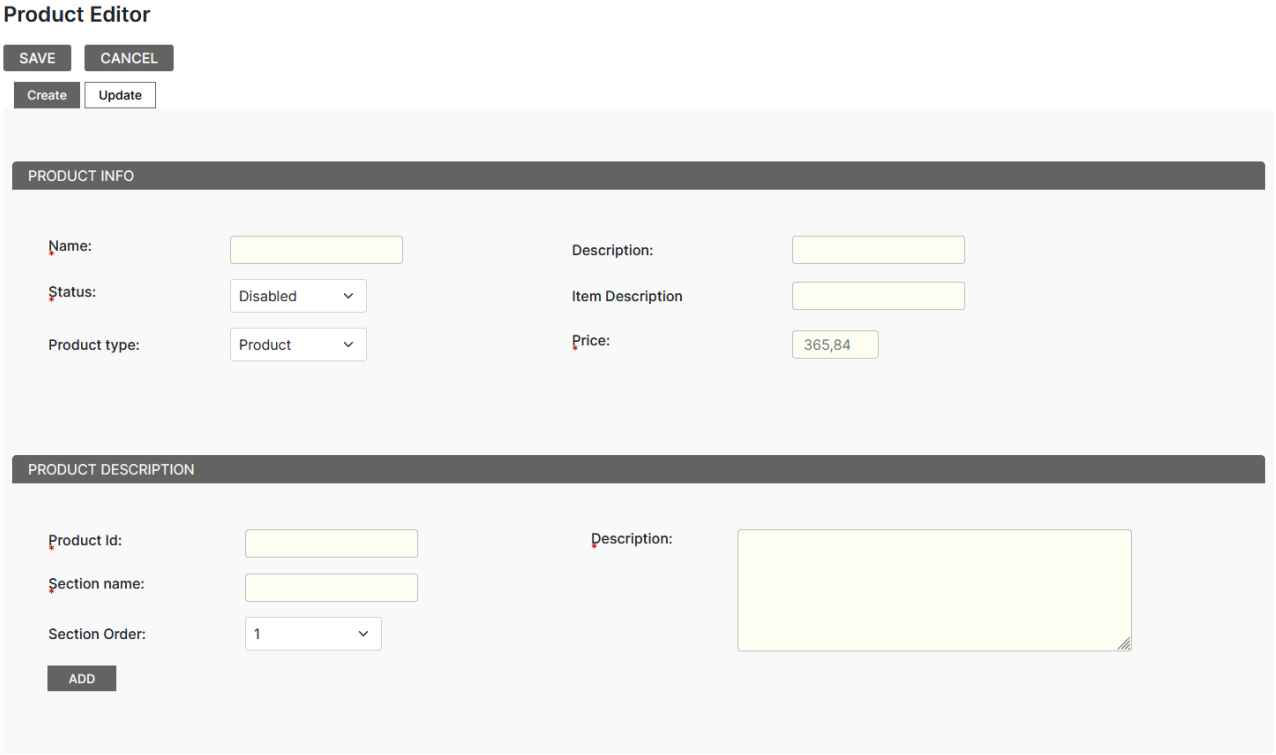

Рис. 2.10. Додавання продуктів

2.4.2. Структура бази даних

База даних, що визначена для цієї кваліфікаційної роботи, є реляційною. Було визначено основні логічні сутності та побудовано фізичні структури, а також контекст бази даних для взаємодії з таблицями в застосунку. Основними сутностями є:

– Замовлення. Дана сутність представляє список замовлень, які виконали користувачі. Ця інформація відображується на сторінці списку товарів.

– Продукти. Містить інформацію про усі наявні продукти, їх назву, відображуваний опис на сторінці каталогу та ціну продукту, а також загальний статус відображення в магазині.

– Транзакції. Зберігає усю інформацію про виконані замовлення, та користувача, що був ініціатором даної операції.

– Опис товарів. Являє собою опис загальної структури кожної окремої сторінки продукту. А саме назви розділів та опис для них.

– Ролі. Представляє список усіх доступних ролей та рівнів доступу, що може мати користувач.

Таким чином, логічна структура бази даних інтернет – магазину буде складатися та мати зв’язки, що вказані на рис. 2.11.

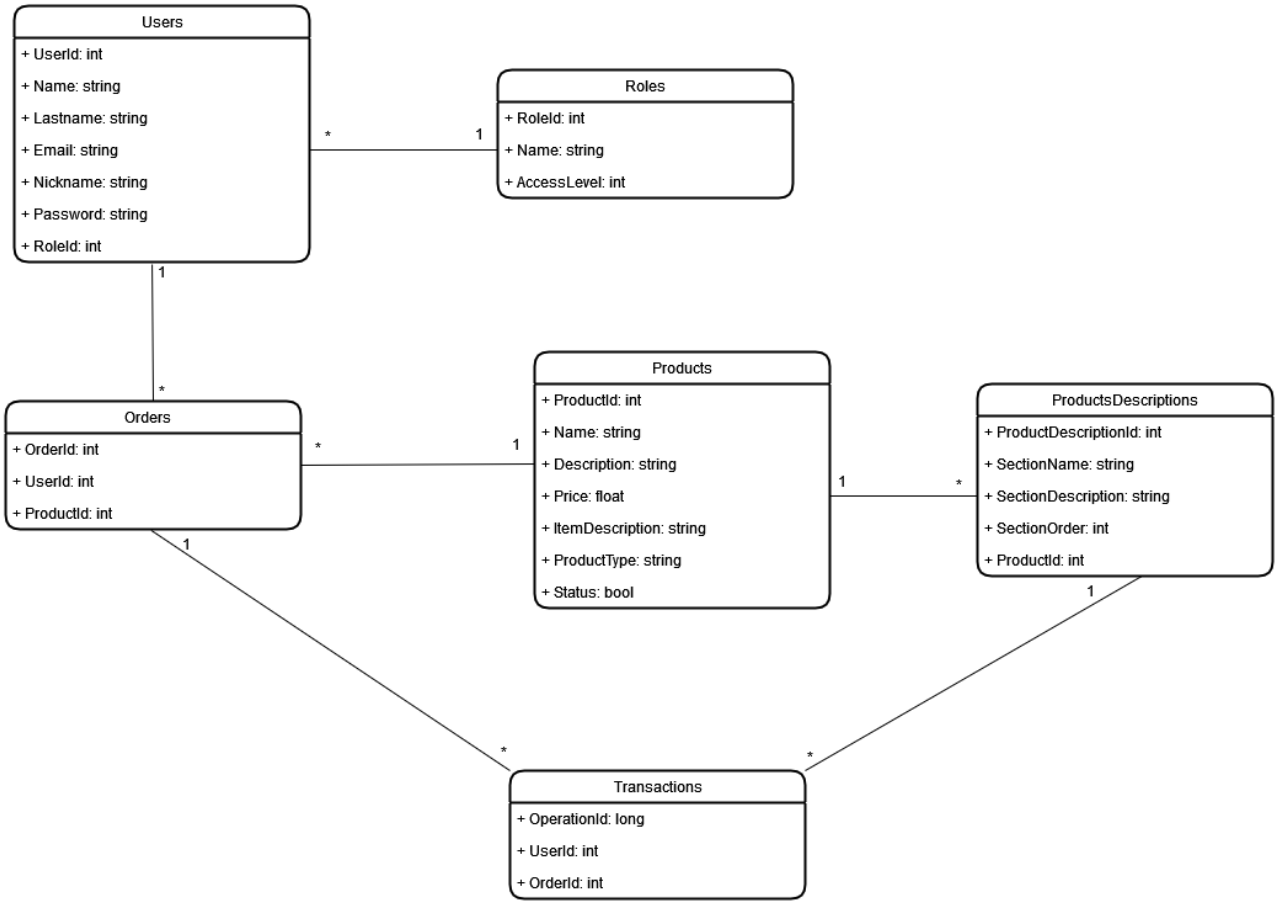

Рис. 2.11. Логічна модель даних UML

Детальний опис призначення атрибутів вказано в таблиці 2.2.

Таблиця 2.2

Атрибути і первинні ключі сутностей

СУТНІСТЬ	ПЕРВИННИЙ КЛЮЧ	АТРИБУТИ
Orders	Унікальний ключ номер замовлення	Унікальний ключ номер замовлення Унікальний ключ номер клієнта Унікальний ключ номер продукту
Products	Унікальний ключ номер продукту	Унікальний ключ номер продукту Назва продукту Опис Ціна Опис в каталозі Тип продукту Статус
ProductsDescriptions	Унікальний ключ номер опису продукту	Унікальний ключ номер опису продукту Назва секції Опис секції Порядок секції Унікальний ключ номер продукту

Продовж. табл. 2.2

Transactions	Унікальний ключ номер операції	Унікальний ключ номер операції Унікальний ключ номер користувача Унікальний ключ номер замовлення
Roles	Унікальний ключ номер ролі	Унікальний ключ номер ролі Назва ролі Рівень доступу
Users	Унікальний ключ номер користувача	Унікальний ключ номер користувача Прізвище користувача Ім’я Нікнейм Пароль Емейл Унікальний ключ номер ролі

Таким чином, опис зв’язків таблиць буде наступним:

– Orders и Products. Кожне замовлення пов’язане з одним продуктом, але один продукт може бути пов’язаний з множиною замовлень. Отже, зв’язок один – до – багатьох.

– Orders и Users. Кожне замовлення пов’язано з одним користувачем, та один користувач може бути зв’язаний з множиною замовлень. Тип даного зв’язку – багато – до – одного.

– Users та Roles. Кожен користувач має одну роль, але одна роль може бути прив’язана до багатьох користувачів. Отже, зв’язок – один – до – багатьох.

– ProductDescriptions та Products. Кожен опис продукту зв’язаний з одним продуктом, але один продукт може мати багато описів. Цей зв’язок представляю один – до – багатьох.

– Orders та Transactions та Users і Transactions представляють зв’язки один – до – багатьох, де в кожній транзакцій для кожного запису може бути вказані замовлення та користувачі, що до них відносяться.

Для детального представлення опису фізичної моделі використано таблиці (табл. 2.3 – 2.7).

Таблиця 2.3

Таблиця «Замовлення»

Orders (Замовлення)				1
№ п/п	Найменування стовбців	Примітка	Тип	Розмір
1.	OrderId	Унікальний ключ номер замовлення	integer	
2.	UserId	Унікальний ключ номер користувача	integer	
3.	ProductId	Унікальний ключ номер продукту	integer	

Таблиця 2.4

Таблиця «Користувачі»

Users (Користувачі)				2
№ п/п	Найменування стовбців	Примітка	Тип	Розмір
1.	UserId	Унікальний ключ номер користувача	integer	
2.	Name	Ім’я користувача	text	
3.	Lastname	Прізвище користувача	text	
4.	Email	Емейл	text	

Продовж. табл. 2.4

5.	Nickname	Нікнейм користувача	text	
6.	Password	Пароль	text	
7.	RoleId	Унікальний ключ номер ролі	integer	

Таблиця 2.5

Таблиця «Ролі»

Roles (Ролі)				3
№ п/п	Найменування стовбців	Примітка	Тип	Розмір
1.	RoleId	Унікальний ключ номер ролі	Integer	
2.	Name	Назва ролі	text	
3.	AccessLevel	Рівень доступу	integer	

Таблиця.2.6

Таблиця «Продукти»

Products (Продукти)				4
№ п/п	Найменування стовбців	Примітка	Тип	Розмір
1.	ProductId	Унікальний ключ номер продукту	integer	
2.	Name	Назва продукту	text	
3.	Description	Опис	text	
4.	Price	Ціна продукту	real	
5.	ItemDescription	Опис в каталозі	text	
6.	ProductType	Тип продукту	text	
7.	Status	Статус продукту	text	

Таблиця 2.7

Таблиця «Транзакції»

Transactions (Транзакції)				4
№ п/п	Найменування стовбців	Примітка	Тип	Розмір
1.	OpereationId	Унікальний ключ номер операції	integer	
2.	OrderId	Унікальний ключ номер замовлення	integer	
3.	UserId	Унікальний ключ номер користувача	integer	

Фізична структура бази даних представлена на рис. 2.12. Ця діаграма відображує структуру бази даних на фізичному рівні: колонки, та які дані будуть зберігатися в таблицях, типи їх даних, індекси, обмеження та зв’язки між ними. Це надає повне представлення о фізичній реалізації.

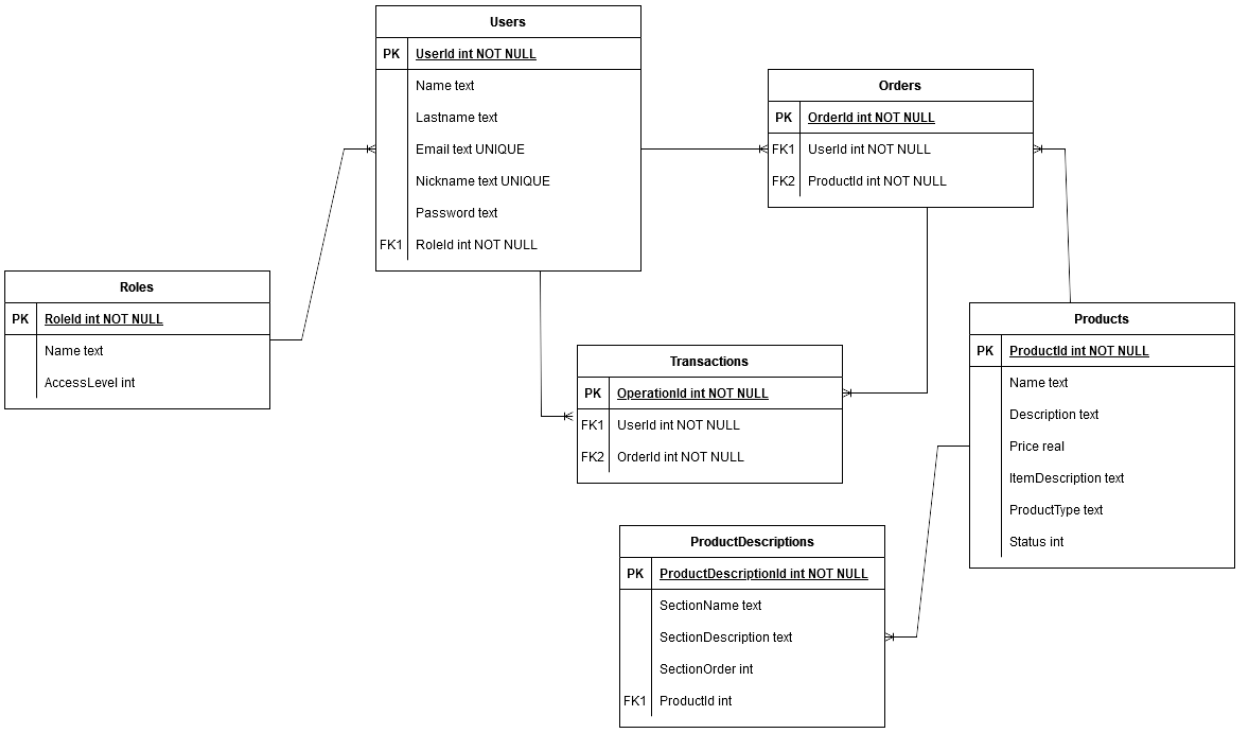

Рис. 2.12. Фізична структура бази даних

2.4.3. Файлова структура проекту

Основна структура проекту, який розроблений за архітектурним шаблоном MVC включає:

– Контролери. Містить обробники, що відпрацьовують вхідні запроси від користувачів, взаємодіють з моделлю та обирають відповідне представлення для відображення даних.

– Моделі. Зберігає файли класів моделей, що представляють дані застосунку та бізнес логіку.

– Представлення. Містить файли, які відображають дані користувача. Це файли Razor (.chtml) [41]

– Wwwroot. В поточному проекті зберігає статичні файли, такі які CSS, JavaScript, зображення, що доступні клієнту;

Загальна структура поточного проекту зображена на рис 2.13 – 2.15.

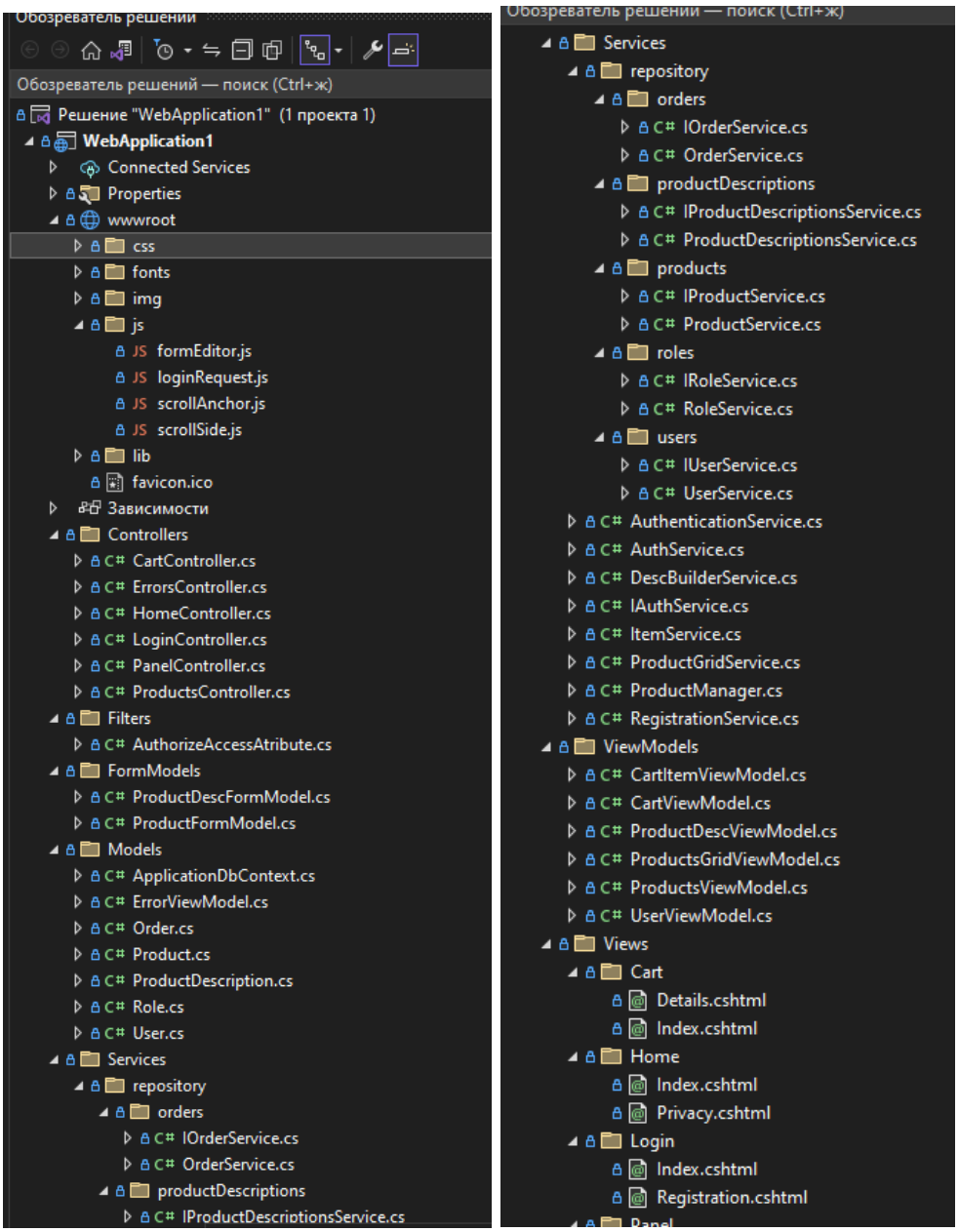

Рис. 2.13. Файлова структура проекту

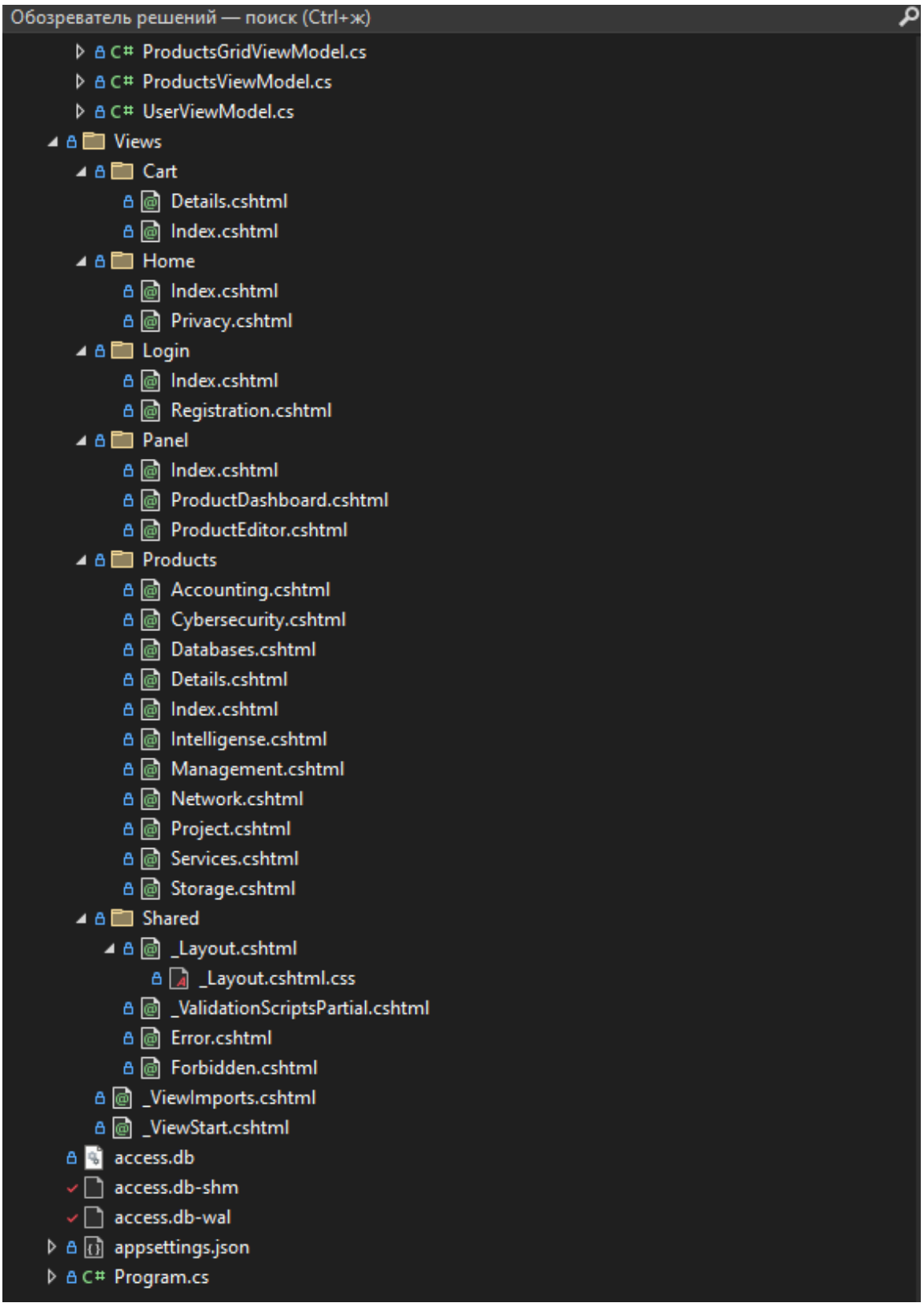

Рис. 2.14. Розгорнута файлова структура

2.5. Обґрунтування та організація вхідних та вихідних даних програми

Програмне забезпечення здійснює отримання даних, безпосередньо, використовуючи клієнтські форми. Вхідні дані підготовлені згідно отримуваної структури, що міститься в приймаючому контролері, та надсилаються за допомогою формату JSON. Також, дані передаються у вигляді строкового запиту параметрів GET.

Вхідні дані включають:

- дані про продукти;
- дані користувачів;
- інформація про замовлення;

Формат цих даних містить:

- Текстові. Назви, описи, характеристики товарів.
- Цифрові. Ціни, кількість товарів у замовленнях.

Вхідні дані кодуються за допомогою стандартів UTF-8. Для забезпечення коректного відображення різних мов і символів.

Формат вихідних даних надає інформацію про продукти та замовлення, а також профіля користувача у вигляді HTML – сторінок, стилізованих за допомогою CSS та JavaScript. Таким чином, вихідні дані наступні:

- список каталогів;
- список замовлень;
- дані користувачів;
- інформація про замовлення;
- особиста інформація профілю;

Вихідні дані формуються відповідно до кожного представлення. Для цього, використовуються класи ViewModel [42], що зображено на рис 2.16, які передаються до представлень і зберігають інформацію про кожну необхідну на сторінці сутність. Попередньо, ці дані оброблюються та заповнюються в контролері, з використанням сервісів додатку.

Серед одних вхідних даних користувача є сторінка аутентифікації та авторизації, що з’являється при входу у систему (рис 2.16).

```
namespace WebApplication1.ViewModels
{
    Ссылок: 16
    public class CartItemViewModel
    {
        Ссылок: 3
        public int? productId { get; set; }
        Ссылок: 5
        public string? Name { get; set; }
        Ссылок: 9
        public int? Count { get; set; }
        Ссылок: 3
        public string? Description { get; set; }
        Ссылок: 7
        public float? Price { get; set; }
    }
}
```

Рис. 2.15. Елемент кошика. CartItemViewModel

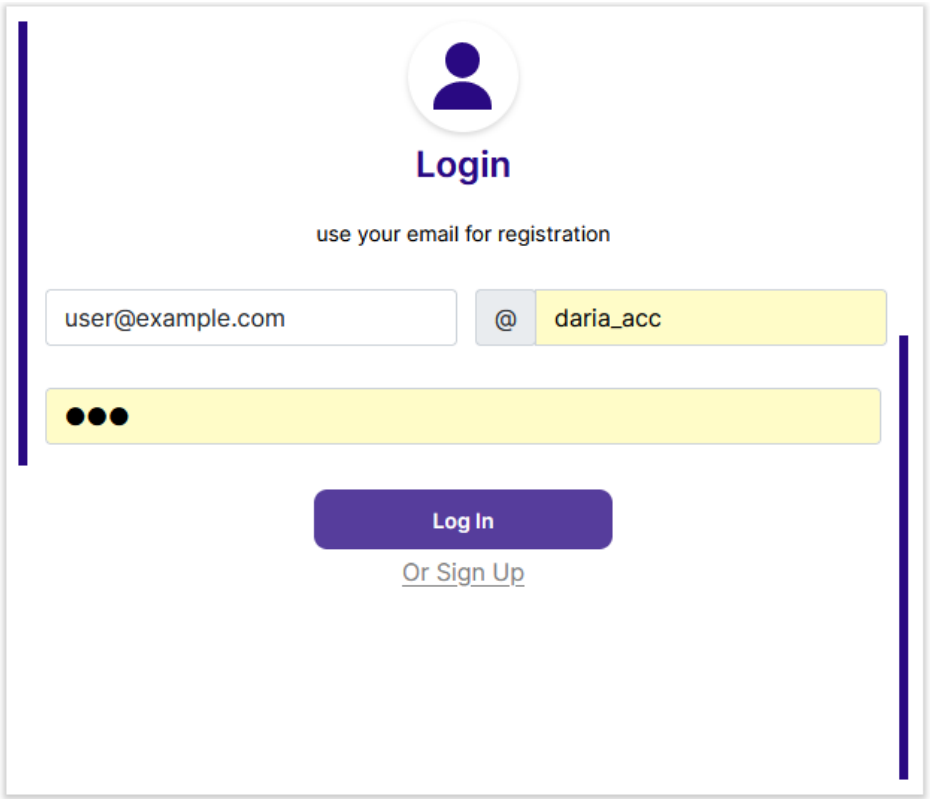

Рис. 2.16. Аутентифікація

2.6. Опис розробленої системи

2.6.1. Використані технічні засоби

Для розробки та проведення тестування додатку використовувалися технічні засоби з наступними характеристиками:

- центральний процесор Intel Core i7-1165G7 2.80GHz;
- оперативна пам’ять: 32гб;
- диск SDD: Kingston A2000 NVMe PCIe SSD 1TB;
- відеокарта: NVIDIA GeForce RTX 3060 6гб;
- операційна система: 64 – розрядна Windows 11;
- комп’ютерна миша;
- клавіатура.

2.6.2. Використані програмні засоби

Під час розробки даного продукту було використано наступні програмні засоби:

- Visual Studio;
- Visual Studio Code with OpenLive server;
- SQLite DB Browser;
- Figma;

Visual Studio [43] – це інтегроване середовище розробки (IDE) від компанії Microsoft, що надає інструменти для створення різноманітного програмного забезпечення. Ця платформа підтримує широкий спектр програмування, таких як C#, що використовується в поточному проекті, C++ та перелік інших мов, забезпечуючи розробникам редактор коду з функціями автозавершення, підсвічування синтаксису та навігації по коду.

Visual Studio (рис 2.17) є потужним інструментом для веб – розробки, та підтримує сучасні технології, такі як ASP.NET, дозволяючи створювати як клієнтські, так і серверні веб – додатки, які можна тестувати та розгортати.

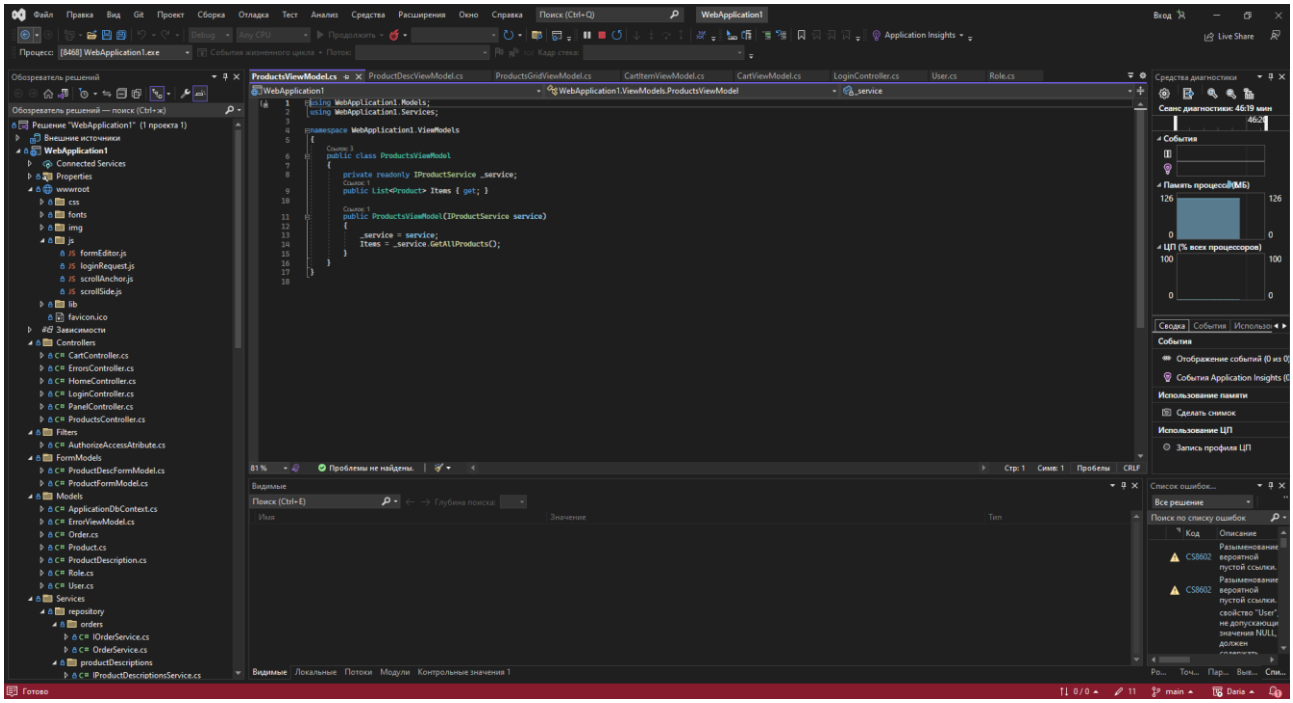

Рис. 2.17. Середовище Visual Studio

Visual Studio Code [44] – є більш легким редактором коду, що теж розроблений компанією Microsoft. Він доступний безкоштовно та відкритий для всіх основних операційних систем.

Однією з переваг VS Code (рис 2.18) є його висока продуктивність та швидкість. Редактор швидко завантажується та працює, не займаючи багато ресурсів системи, що дозволяє розробникам працювати на менш потужних комп'ютерах. Окрім цього, редактор має екосистему розширень, завдяки чому його можна доповнювати новими функціями та інструментами.

В контексті даного додатку, VS Code використовувався для написання та тестування інтерфейсу користувача. Для цього застосовувався Live Server [45] – який забезпечує автоматичне оновлення веб- сторінки при збереженні змін у коді. Це розширення створює локальний сервер, який автоматично відображає зміни в

реальному часі, дозволяючи бачити результати роботи миттєво після кожної зміни.

LiveServer підтримує типи файлів, що включають HTML, CSS, JavaScript та інші ресурси, що використовуються для створення веб – додатків.

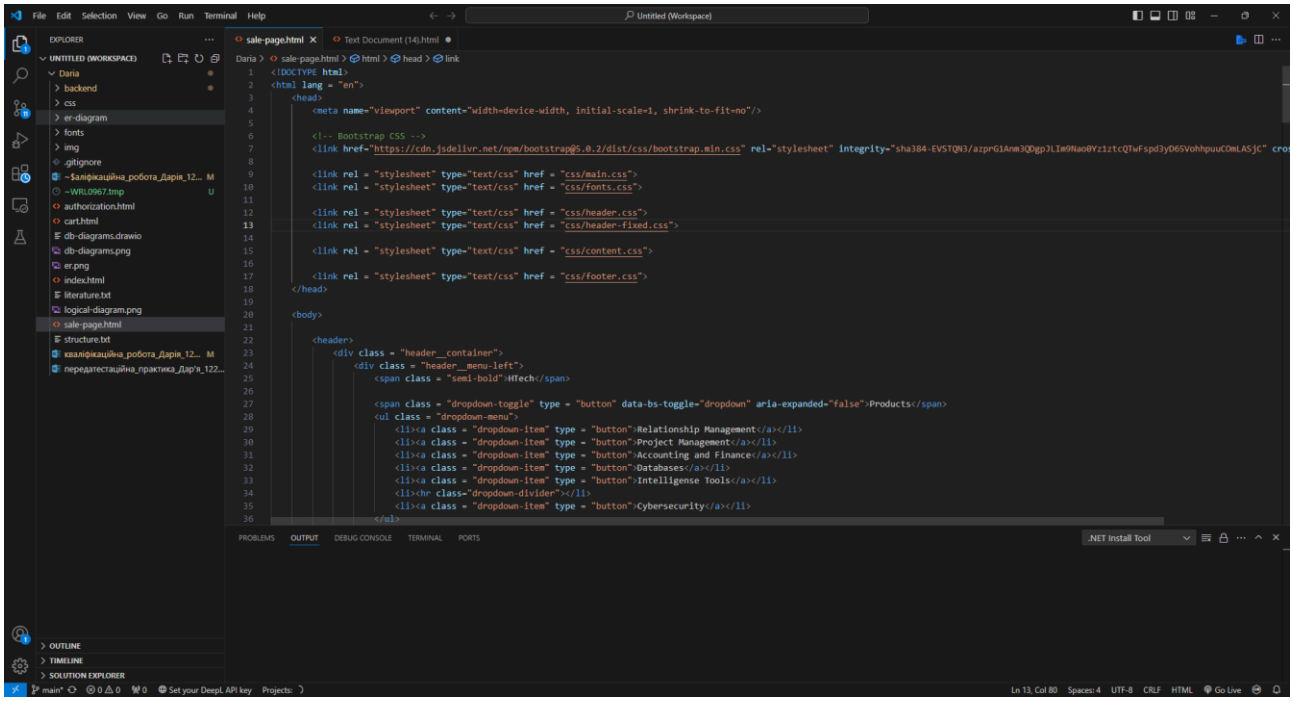

Рис. 2.18. Середовище Visual Studio Code

DB Browser for SQLite [46] – інструмент для роботи з базами даних, що пропонує графічний інтерфейс керування даними. Цей інструмент дозволяє легко створювати, редагувати і переглядати бази даних без необхідності писати SQL – запити вручну.

Під час розробки проекту, даний інструмент, інтерфейс якого зображено на рис. 2.20, використовувався з ціллю перевірки коректності створення таблиць бази даних, що виконував Entity Framework, а також додавання стартових даних для тестування застосунку.

Окрім зазначених інструментів, використовувався сервіс «Figma» [47]. Зображення робочої середи зображено на рис. 2.21. Цей веб – сервіс пропонує можливості редактора векторної графіки та засобів створення прототипів.

Особливістю є робота в реальному часі, та редагування додатку одночасно декількома користувачами. Використовуючи цей інструмент, створено десктопні прототипи макетів застосунку.

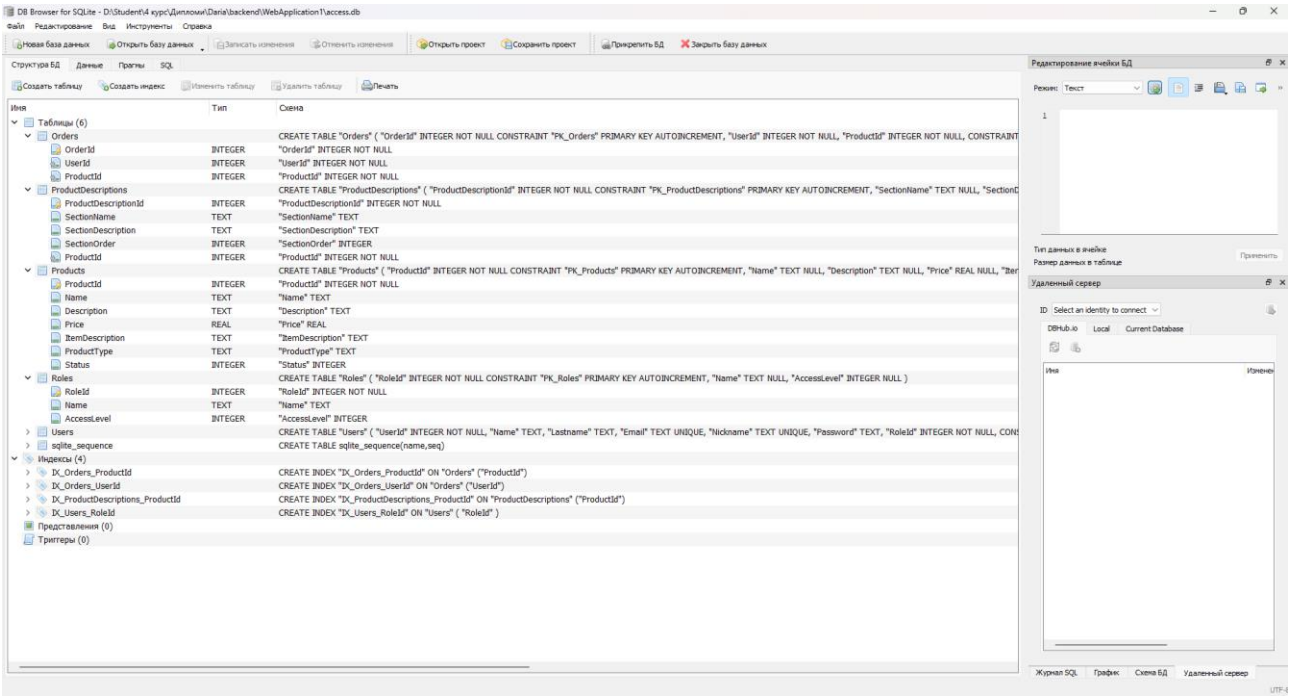

Рис 2.20. Інтерфейс керування DB Browser

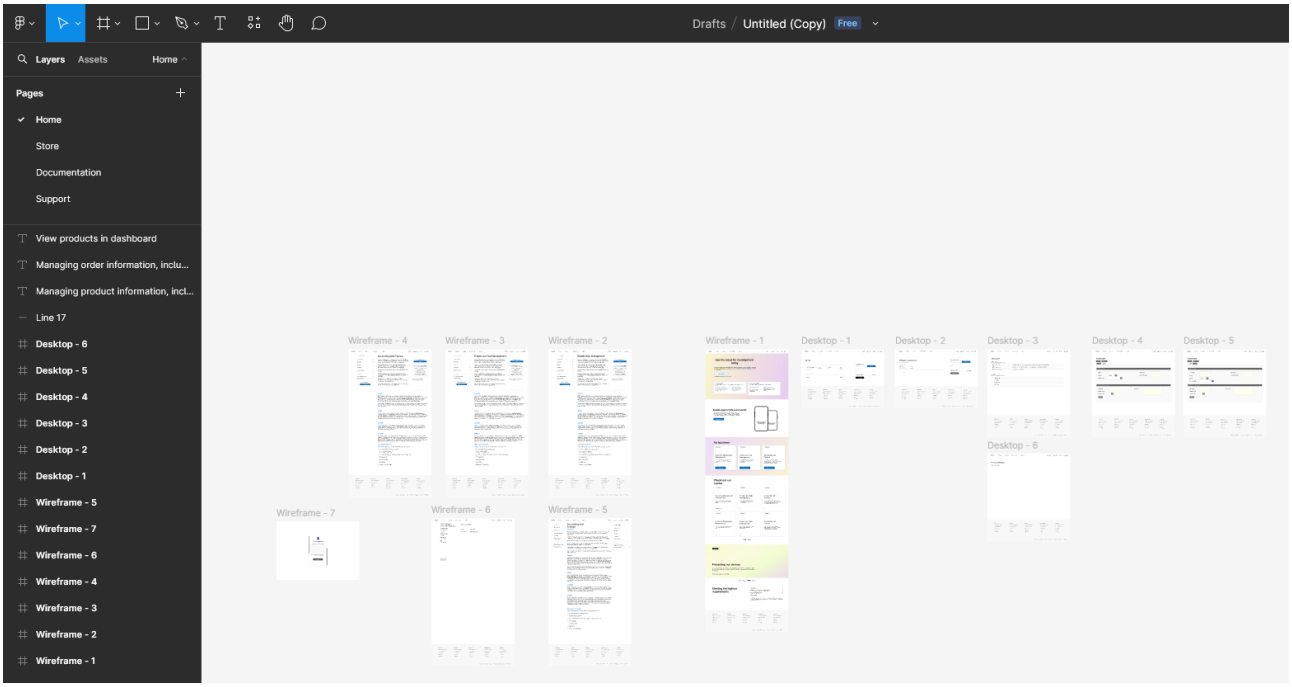

Рис 2.21. Робоче середовище Figma, прототипи сторінок проекту

2.6.3. Виклик та завантаження програми

У процесі тестування допускається запуск програмного рішення з використанням персонального комп’ютера, але для запуску в комерційному застосуванні знадобиться серверне обладнання для обробки великої кількості запитів до системи. Для виклику та завантаження необхідно мати:

– операційна система, що підтримує ASP.NET 6.0. Це може бути Windows, Linux або macOS;

– пакет .NET 6.0 SDK на комп’ютері, що забезпечить необхідні інструменти та бібліотеки для розробки та запуску додатків;

– для запуску ASP.NET додатку на локальному комп’ютері може знадобитися Web Server, такий як IIS;

– при використанні локального серверу, необхідно ввести посилання у браузері. URL для локального застосунку може бути: http://localhost:5002;

– після введення адреси, буде відкрито веб – орієнтований додаток.

2.6.4. Опис інтерфейсу користувача

Після запуску веб – додатку, та звернення до адреси сайту, можна побачити головну сторінку сайту (рис. 2.22). Ця сторінка відображує основну інформацію та елементи, які мають ціль захопити увагу відвідувачів, а також допоможуть знайти необхідні продукти та послуги.

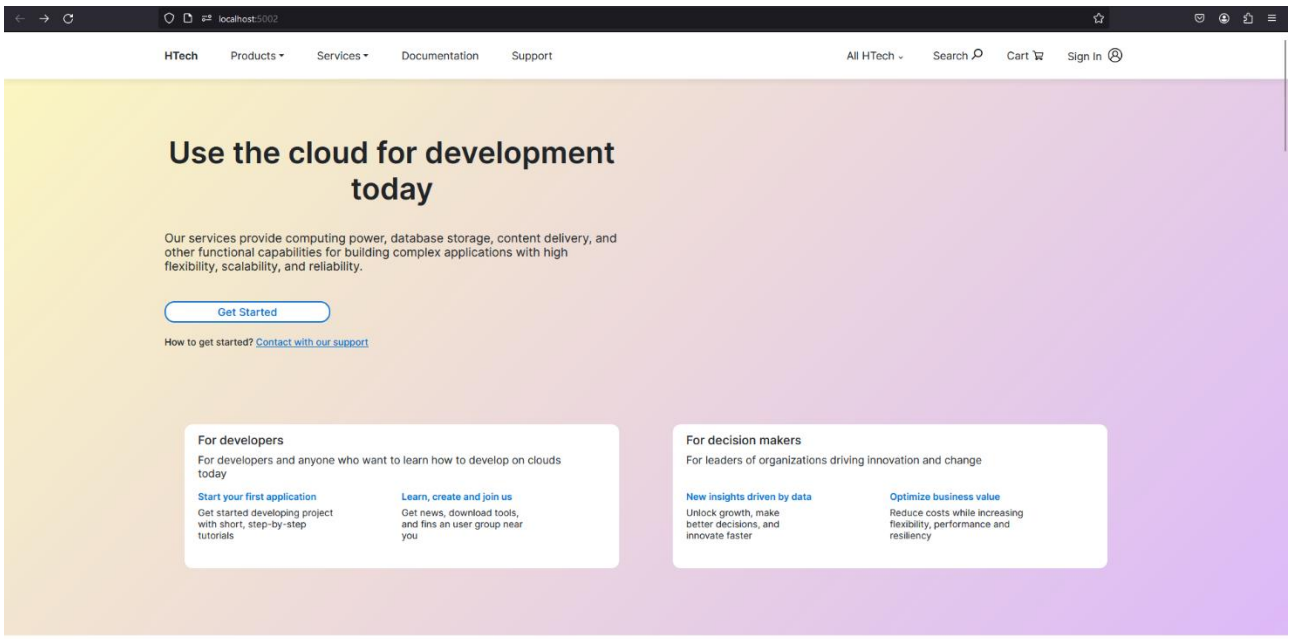

Рис. 2.22. Головна сторінка сайту

Головна сторінка складається з декількох секцій, а саме:

– шапка сайту;
– вступна інформація;
– опис застосунку;
– продукти для бізнеса;
– каталог сервісів;
– секція слайдеру;
– додаткова інформація;
– підвал сайту.

Шапка сайту містить елементи перемикання вкладок (рис. 2.23 – 2.24) та інші сторінки, такі як підтримка, кошик, пошук, авторизація.

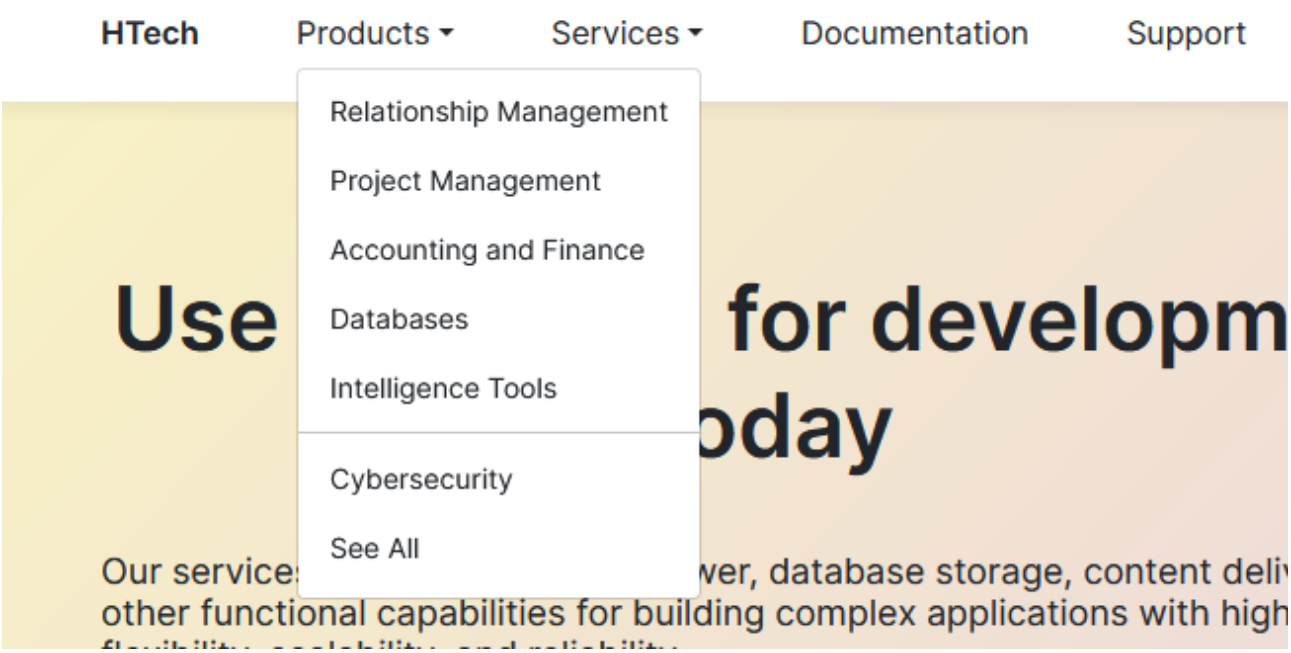

Рис 2.23. Перемикання вкладок основних товарів

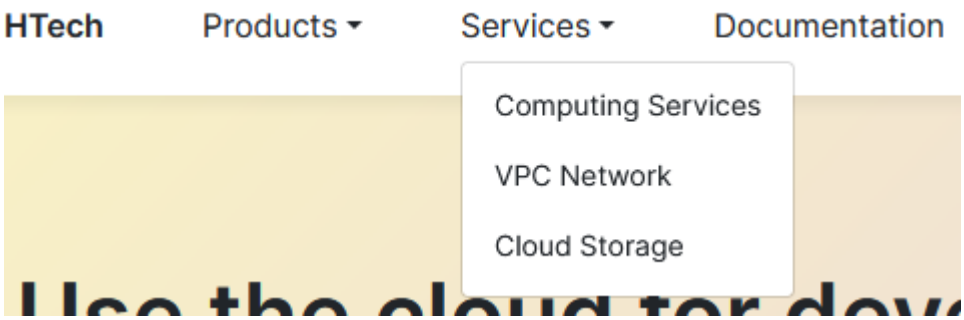

Рис 2.24. Сторінки сервісів

Нижче розташована інформативна секція (рис. 2.25), яка посилає користувача на сторінку документації.

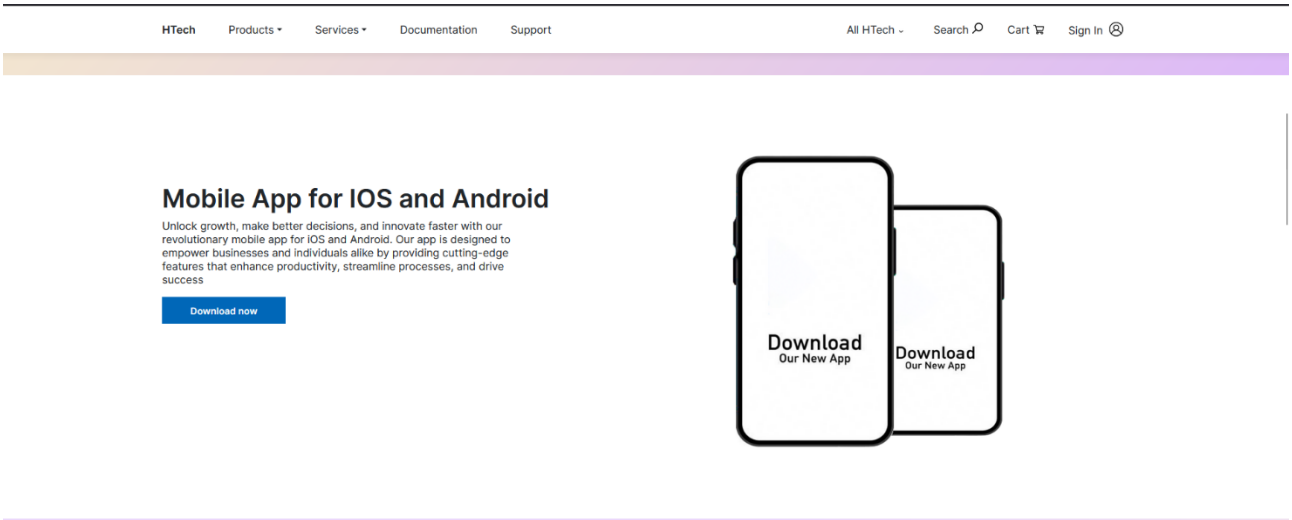

Рис 2.25. Інформативна секція

Основний зміст головної сторінки відображується в розділі сервісів та продуктів для бізнесу (рис 2.26 – 2.28). Кожен продукт має посилання на відповідну сторінку додатка.

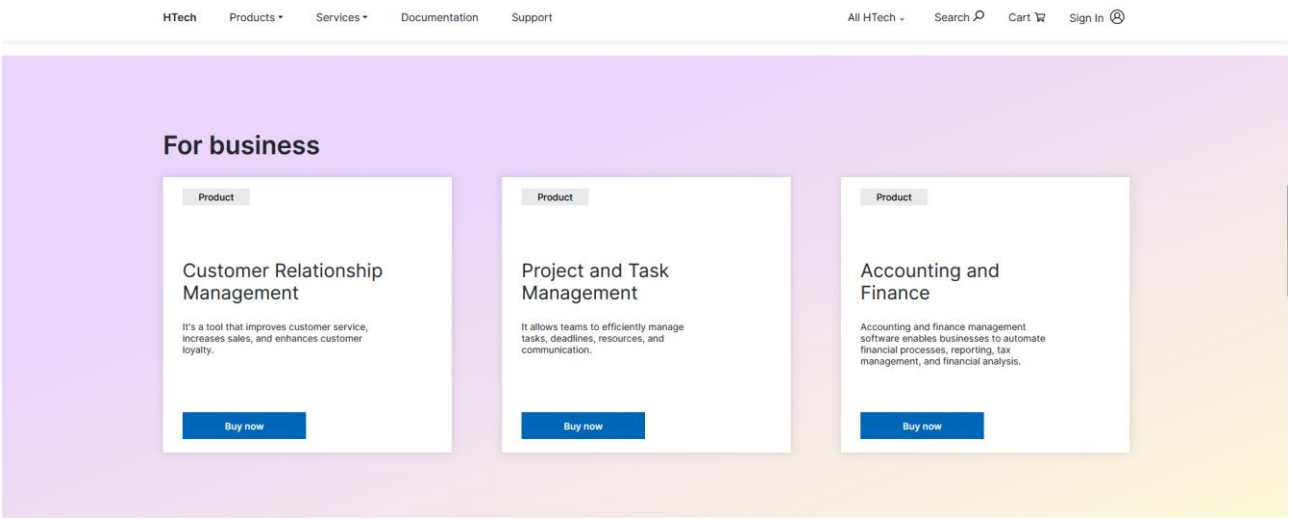

Рис 2.26. Бізнес – рішення

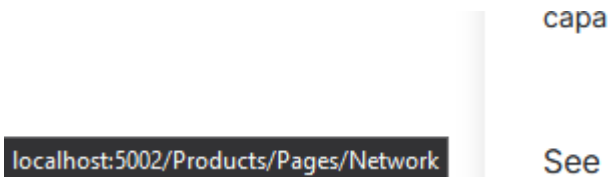

Рис. 2.27. Відображення посилання

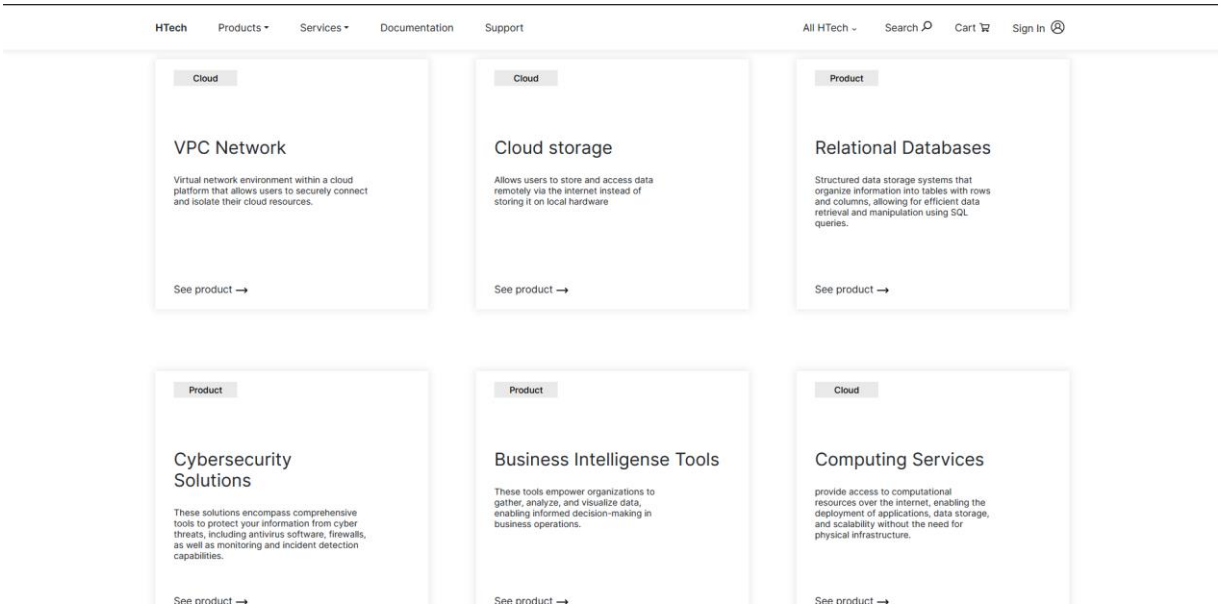

Рис. 2.28. Перелік основних товарів

Окрім цього, на сторінці реалізована секція слайдеру, який надає загальну інформацію користувачам (рис. 2.29).

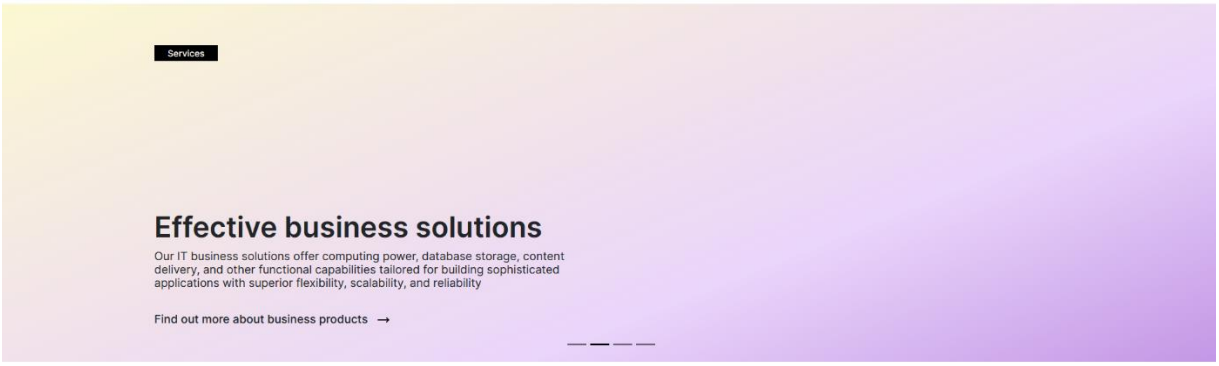

Рис. 2.29. Секція слайдер

Додаткова інформація надає дані про сервіс (рис. 2.30) , та які функціє надає магазин. Представляє собою списки, що розкриваються при натисканні.

Рис. 2.30. Секція додаткової інформації

Футер сайту, що зображений на рис. 2.31, складається з посилань на внутрішні ресурси сайту. Як і шапка сайту, він є статичним для усіх сторінок, окрім вікон реєстрації та авторизації користувача.

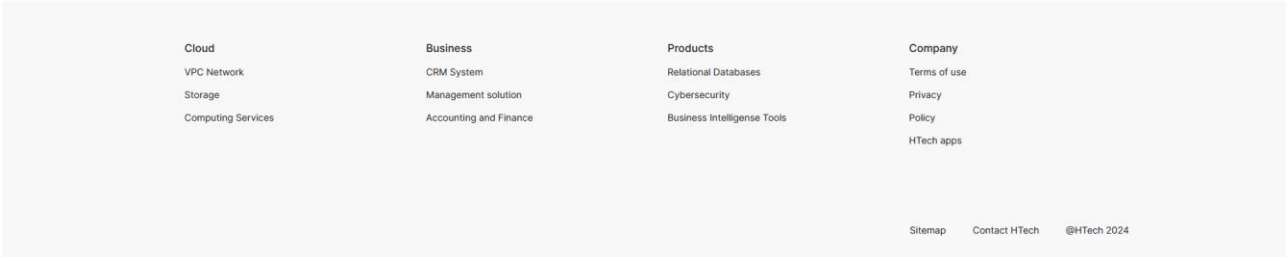

Рис. 2.31. Підвал сайту

Вікно авторизації пропонує ввести дані, якщо користувач вже зареєстрований. При введені некоректних даних з’являється інформація про неіснуючого користувача. Також, форма має механізм валідації (рис 2.32 - 2.34).

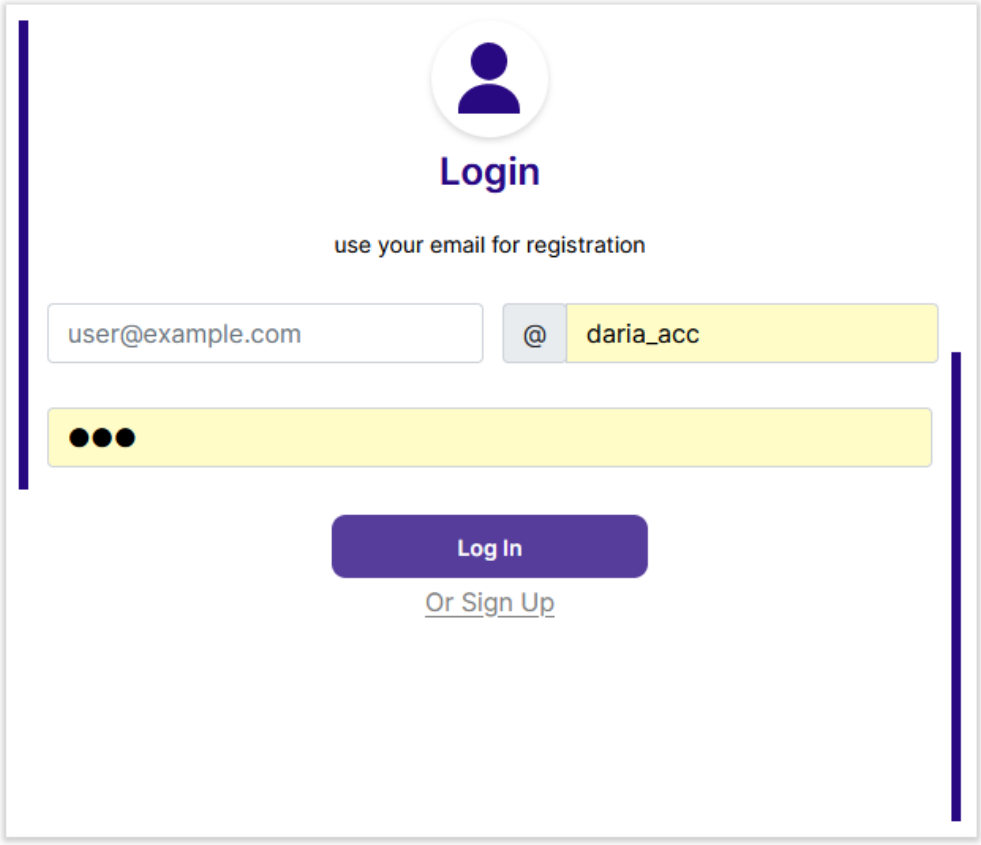

Рис. 2.32. Форма авторизації

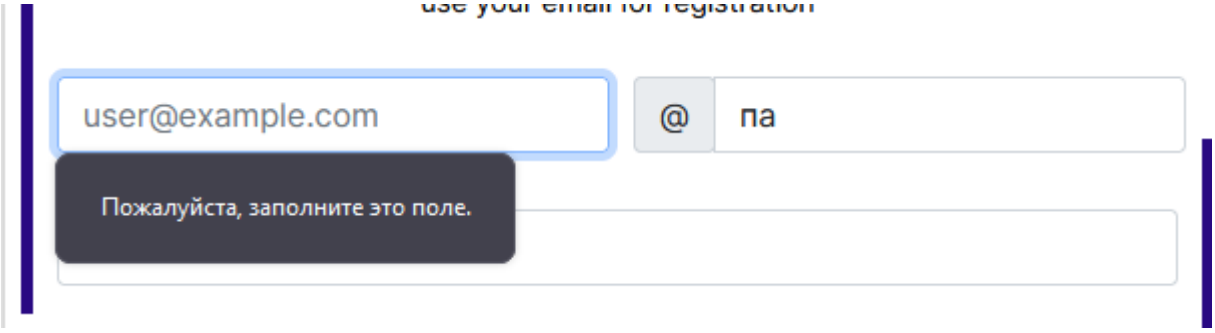

Рис. 2.33. Валідатор форми

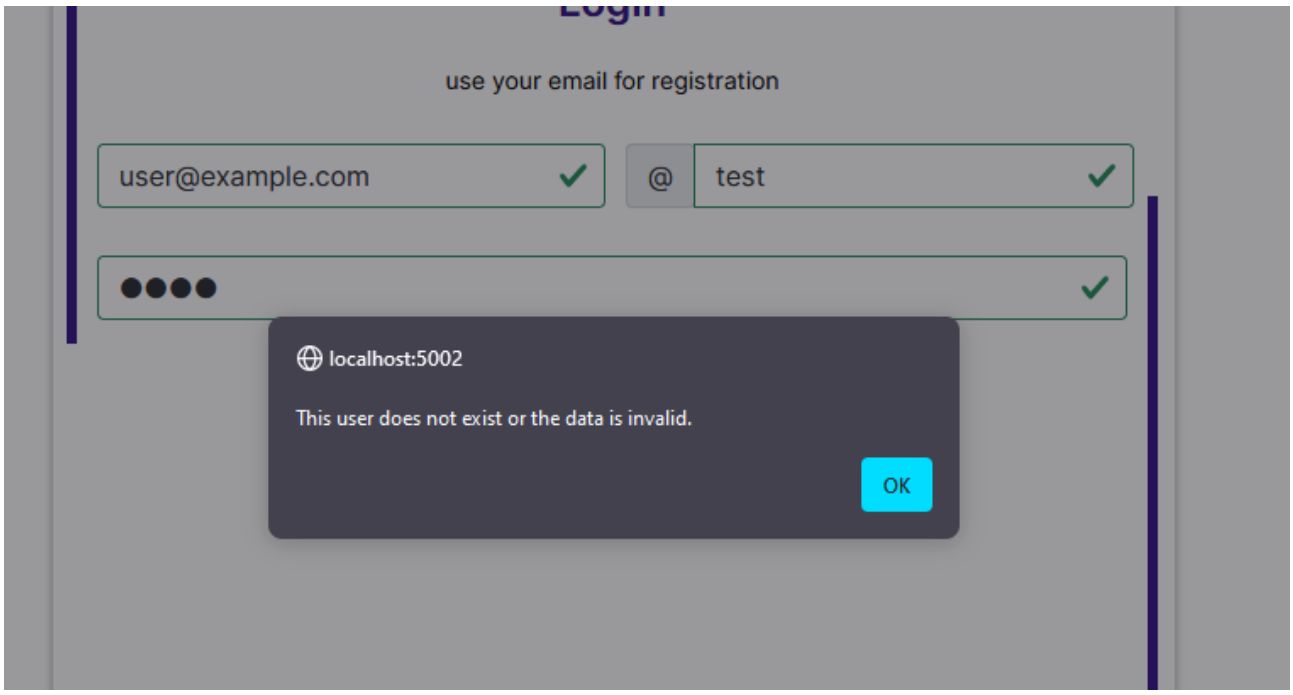

Рис. 2.34. Помилка аутентифікації при вході в систему

Окрема форма реєстрації, яка доступна за посиланням на сторінці авторизації, представляє схожу форму. На даній формі, також присутня функція валідації усіх полів, що зображено на рис 2.35. У разі вдалої реєстрації, користувач потрапляє на голову сторінку. При цому, відбуваються зміни в шапці сайту (рис 2.36).

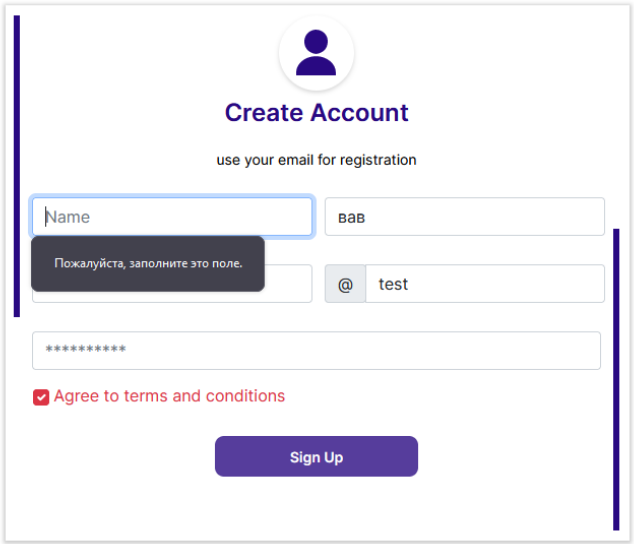

Рис. 2.35. Форма реєстрації

Рис 2.36. Шапка сайту

Після авторизації, користувач може переглянути та оформити замовлення. Ці замовлення потрапляють до кошика (рис 2.37), в якому надається детальна інформацію до кожного товару. Також, існують наступні функції, як Edit та Remove. Перша функція посилає користувача на окрему сторінку, а друга – дозволяє видалити поточний елемент кошика. На рис. 2.38 зображено декілька елементів кошика. Праворуч знаходиться бокова частина. В ній описано елементи застосування промо – коду до загальної вартості покупок та загальна інформацію про ціну.

Кнопка Google Pay працює з методами VISA та Mastercard, що виконана, як зазначалося в минулих підрозділах, як інтеграція сервісу PayPal.

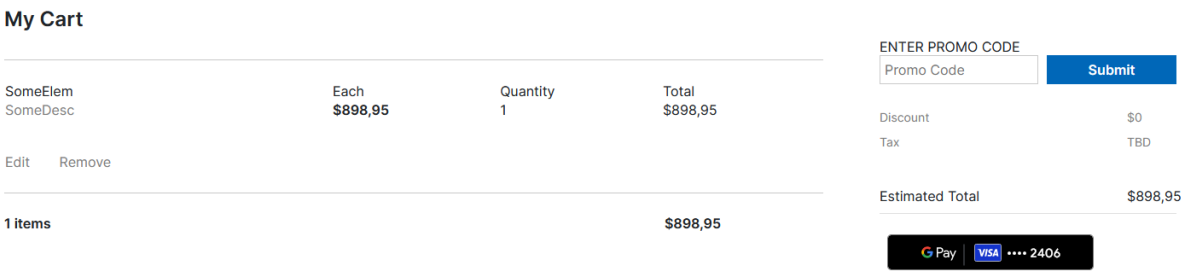

Рис 2.37. Інформація про товари

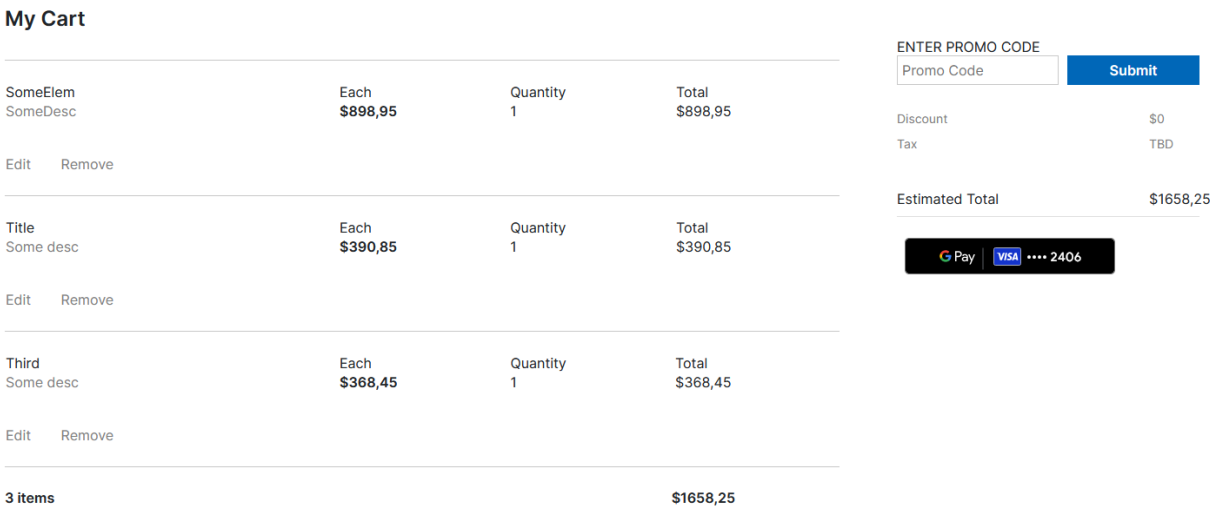

Рис 2.38. Відображення декількох елементів кошику та загальна сума.

Якщо елементів кошику не існує, сторінка буде відображувати відповідну кількість товарів – тобто нуль (рис. 2.38).

Слід зазначити, що при кожному завантажені сторінки, кнопка покупки GooglePay завантажується (рис. 2.39). Таким чином, якщо не виникає мережевих або інших помилок, наприклад, з сумісністю, тоді користувач може побачити її в завантаженому стані.

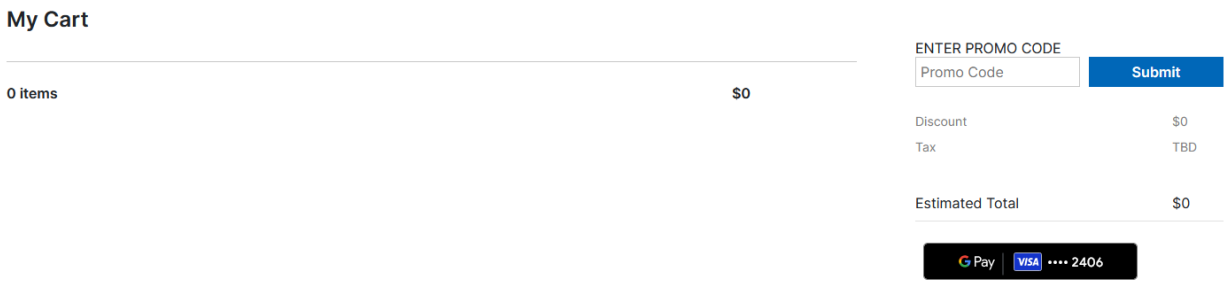

Рис. 2.38. Порожній кошик

Рис. 2.39. Етап завантаження кнопки оплати

Окрім зазначеного функціоналу кошика, користувач може переглянути та відредагувати деталі кожного окремого товару (рис .2.40). Це дозволяє змінити кількість бажаних одиниць, або переглянути інформацію тільки про загальну вартість окремого продукту або сервісу.

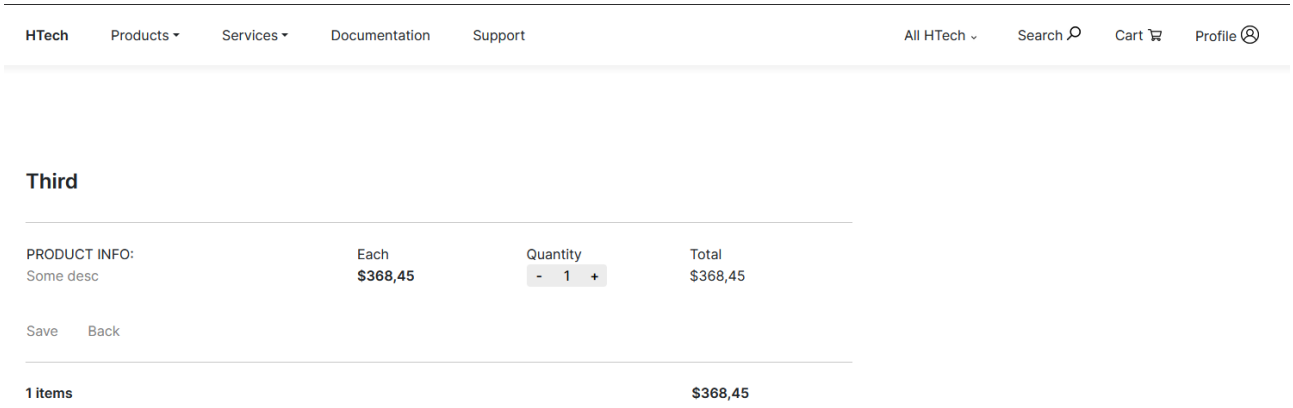

Рис. 2.40. Детальна інформація про товар

Модератор має функціонал взаємодії з продуктами та переглядання додаткової інформації. Так, маючи доступ до панелі користувача (рис 2.41 – 2.42), можливо виконувати певні функції. В випадку, якщо звичайний користувач не має відповідного рівня доступу до цієї секції, сторінка формує відповідне повідомлення (рис 2.43) .

Пункти панелі складаються інструментів керування продуктами та користувачами. Розділ керування користувачами включає менеджер ролей та

користувачів, а також загальний розділ перегляду інформації про них (рис. 2.44 – 2.46).

Панель керування продуктами містить редактор продуктів (рис 2.47) та замовлень (рис. 2.48), а також таблицю перегляду інформації про продукти (рис. 2.49).

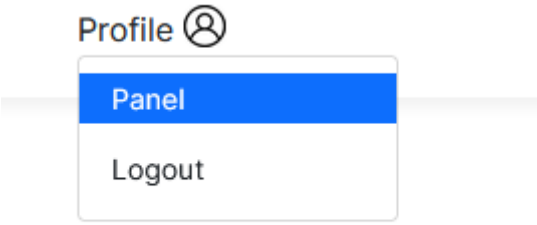

Рис. 2.41. Пункт меню – панель адміністратора

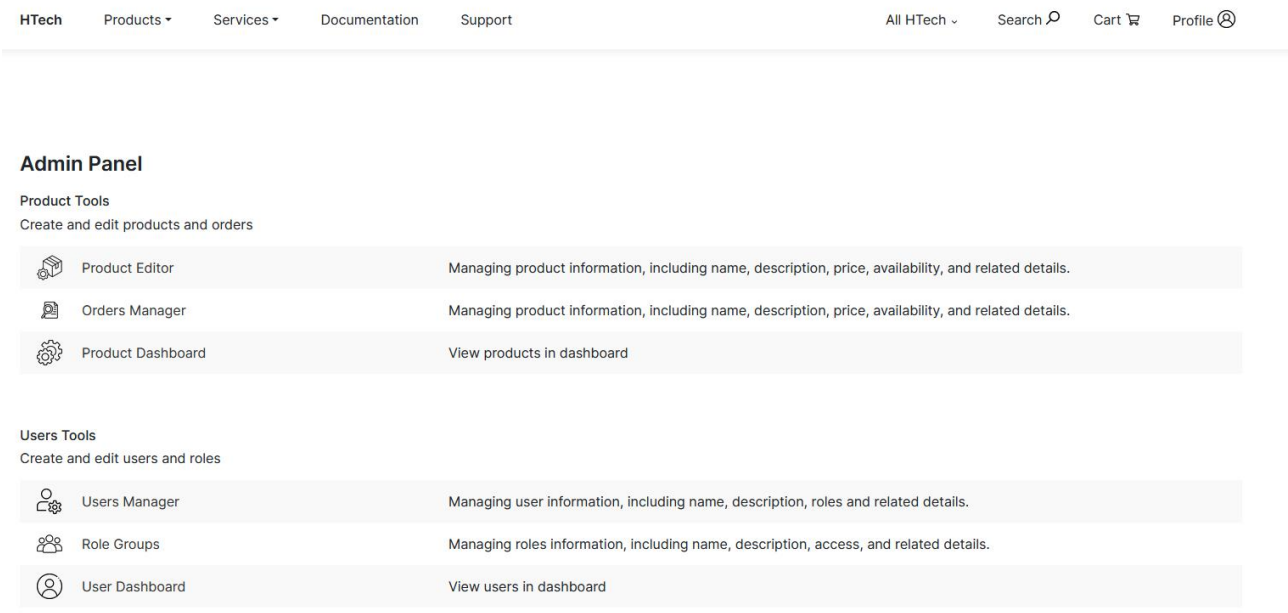

Рис. 2.42. Панель керування адміністратора

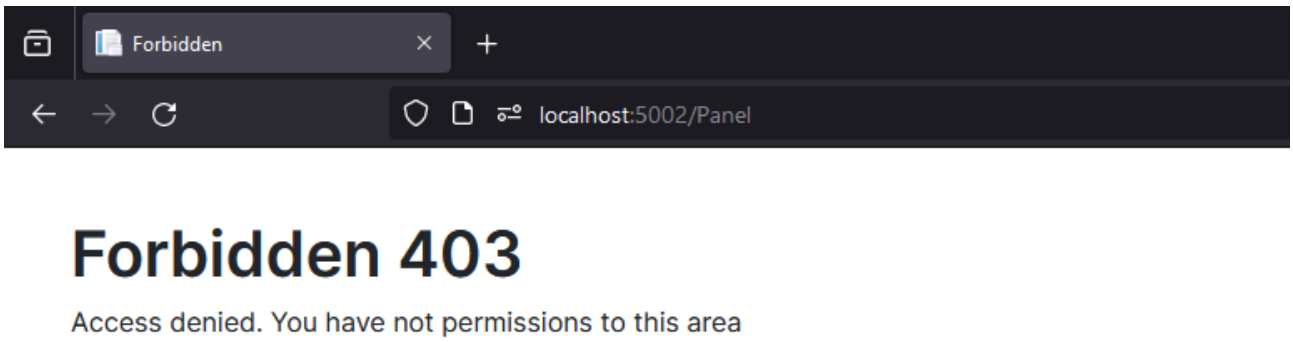

Рис. 2.43. Відсутність доступу до панелі керування

Редактор користувачів складається з єдиної секції, а саме з полей для введення даних нового користувача. Окрім цього, модератор може обрати доступну роль. Ця роле може буде як звичайного користувача, так і модератора системи.

Менеджер продуктів складається з двох секцій: додавання нових продуктів, та керування описом кожного розділу. Опис розділів представляє структуру кожної окремої сторінки, та відповідного опису до цього розділу. Таким чином, формується унікальна та змістовна сторінка кожного товару, що є елементом CMS.

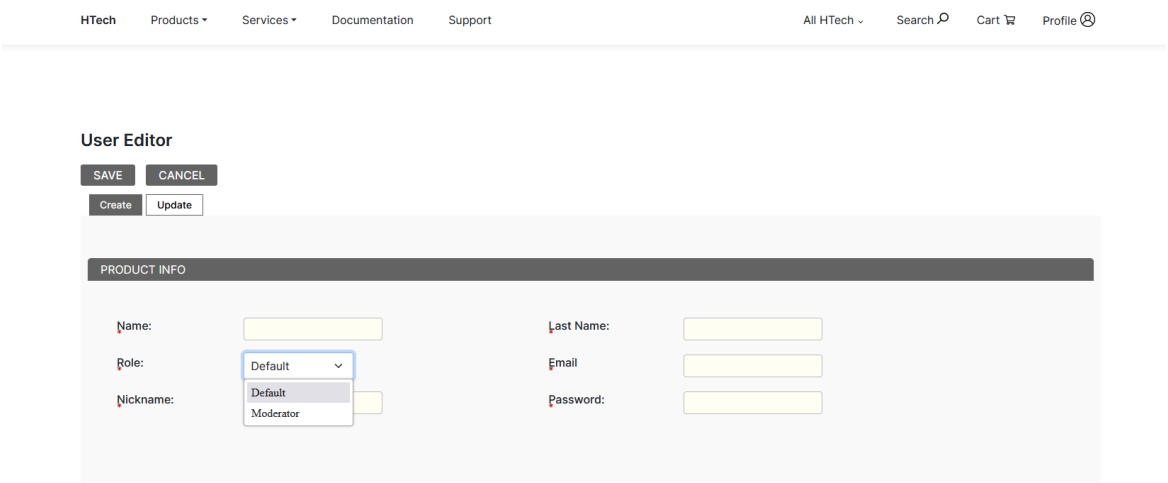

Рис. 2.44. Редактор користувачів

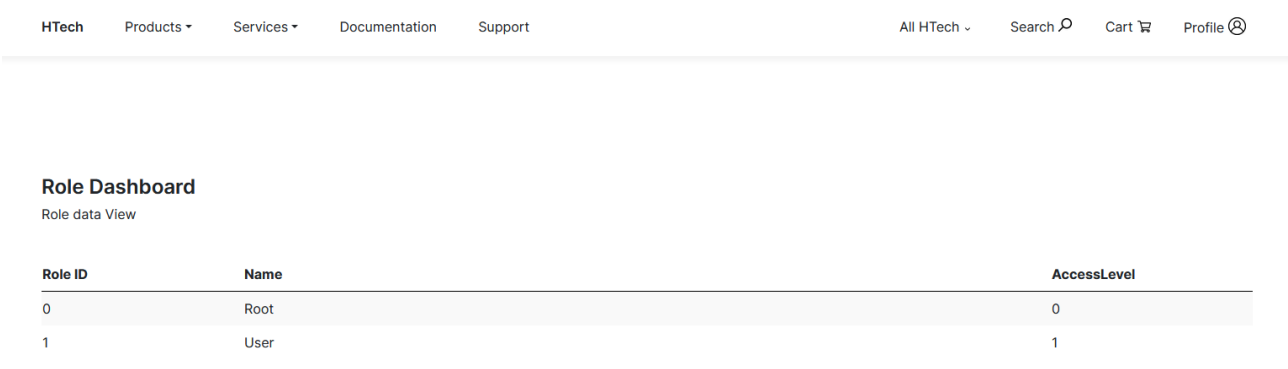

Рис. 2.45. Таблиця ролей

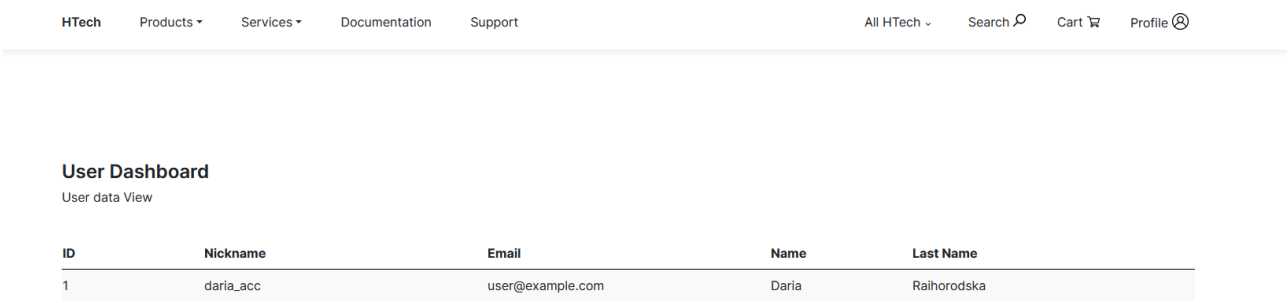

Рис. 2.46. Таблиця користувачів

Таблиця відображення продуктів складається з переліку основних продуктів, та статусу їх відображення на сайті. Якщо статус продукту – вимкнено, то товар не буде зберігатися в каталозі. Як можна побачити на рисунку таблиці продуктів, приведено список макетних даних для відображення працездатності.

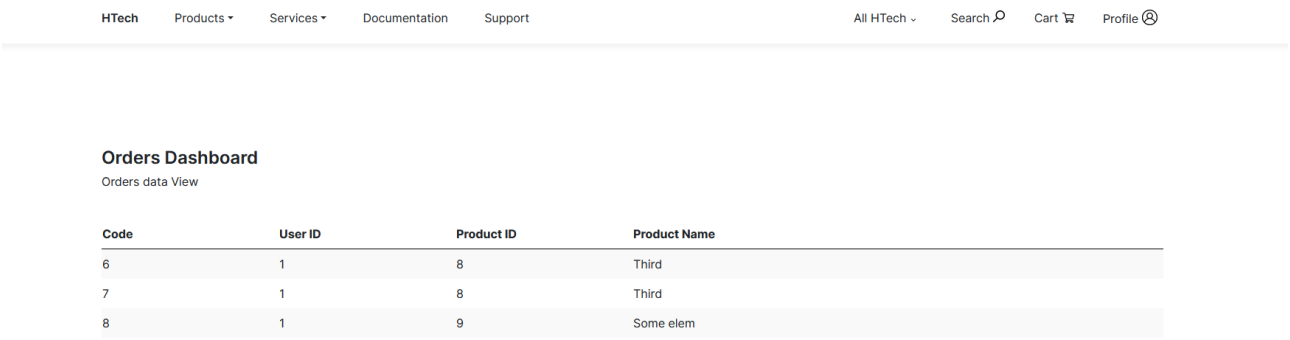

Рис. 2.47. Таблиця замовлень

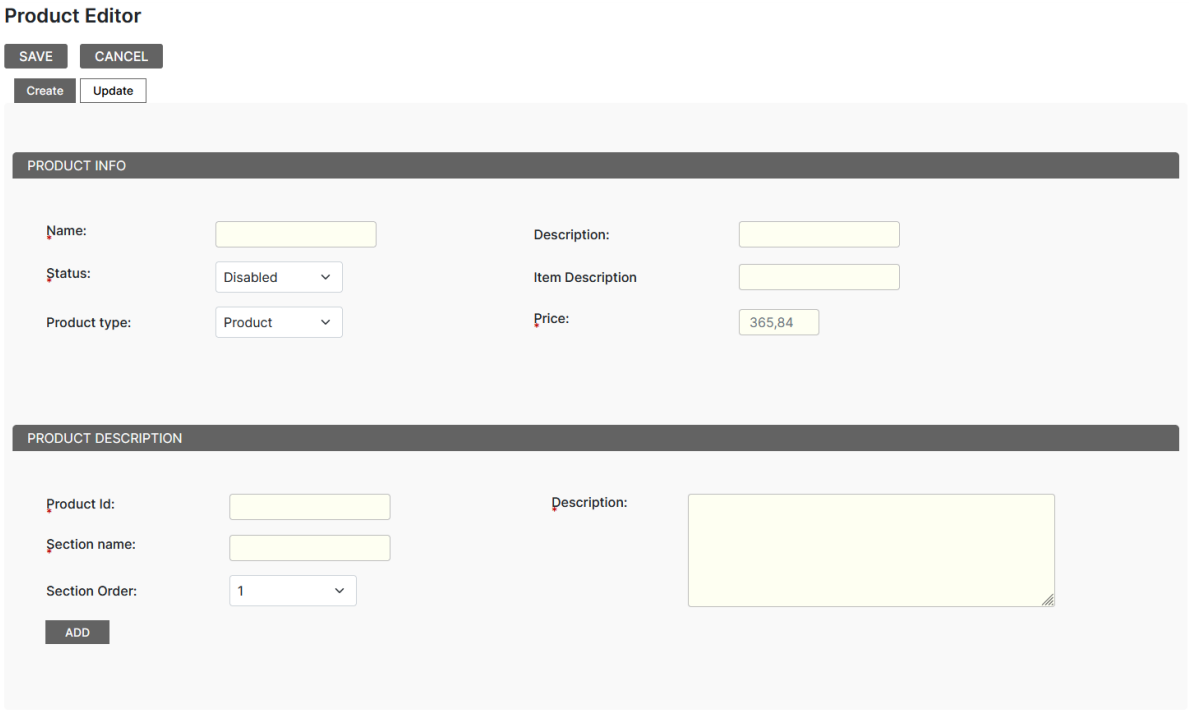

Рис. 2.48. Редактор продуктів

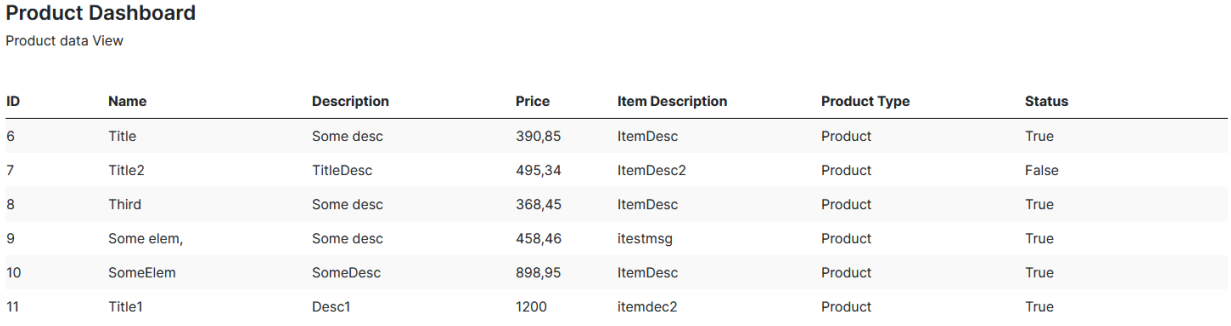

рис. 2.49. Таблиця продуктів

Кожна сторінка продукту або сервісу (рис. 2.50) надає інформацію заголовків та опису, який генерується з опису секцій продуктів, що можна відредагувати в відповідному редакторі. На сторінці наявна кнопка «придбати», з правої частини, що направляє продукт до кошика.

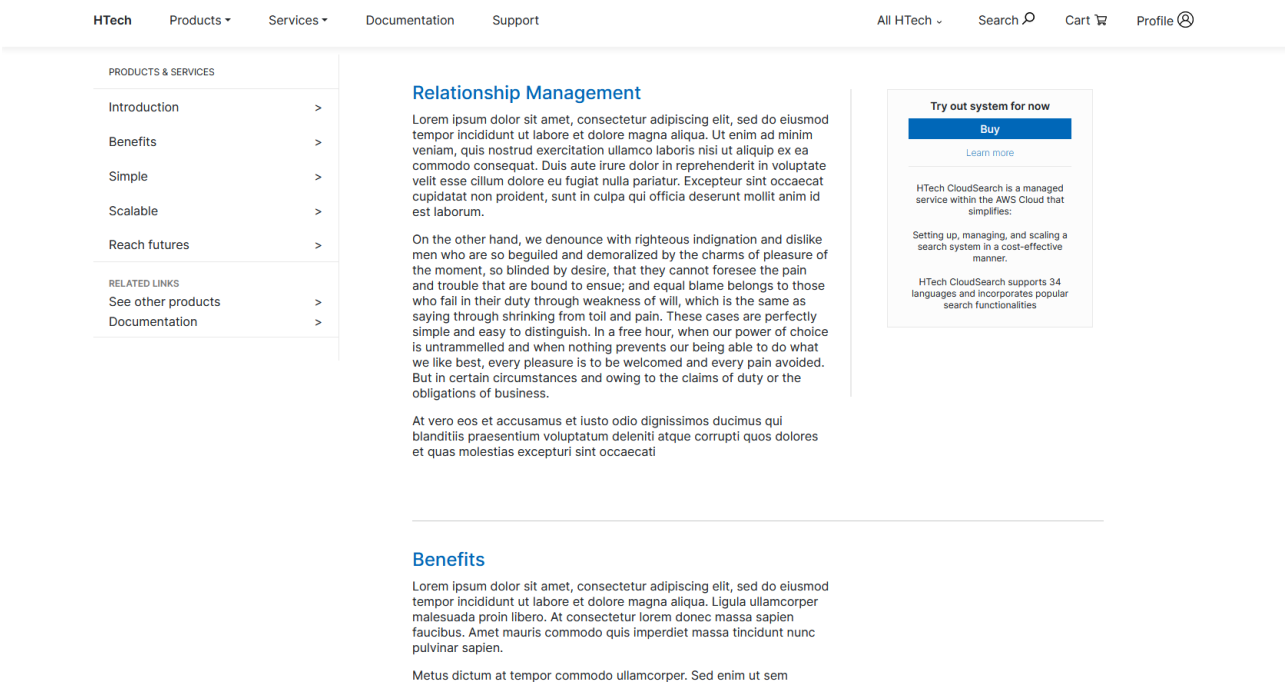

Рис. 2.50. Деталі продукту

Окрім зазначеного функціоналу, існує сторінка загального перегляду продуктів, як зображено на рис. 2.51 – 2.52. В цьому варіанті зображено макетні дані, які співпадають з даними таблиці продуктів.

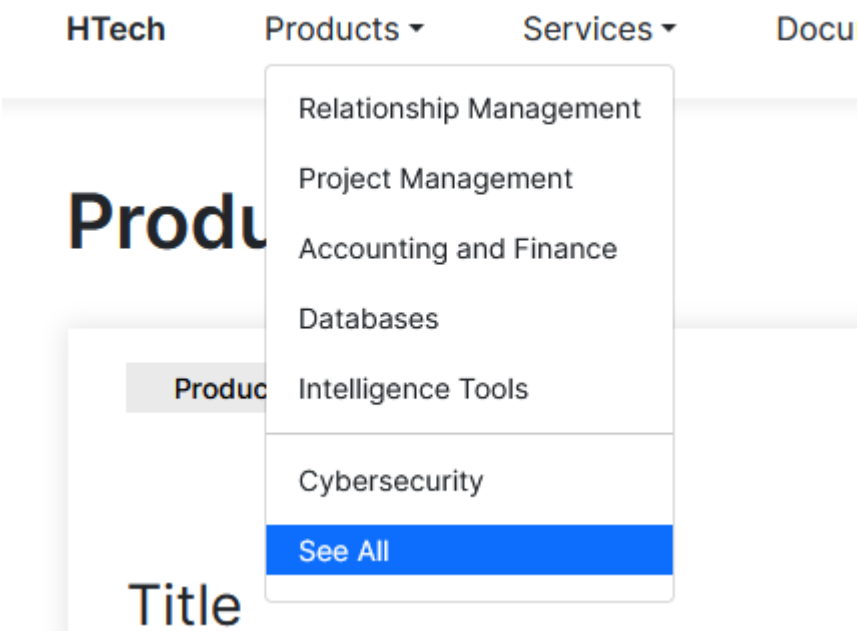

Рис. 2.51. Каталог продуктів в меню

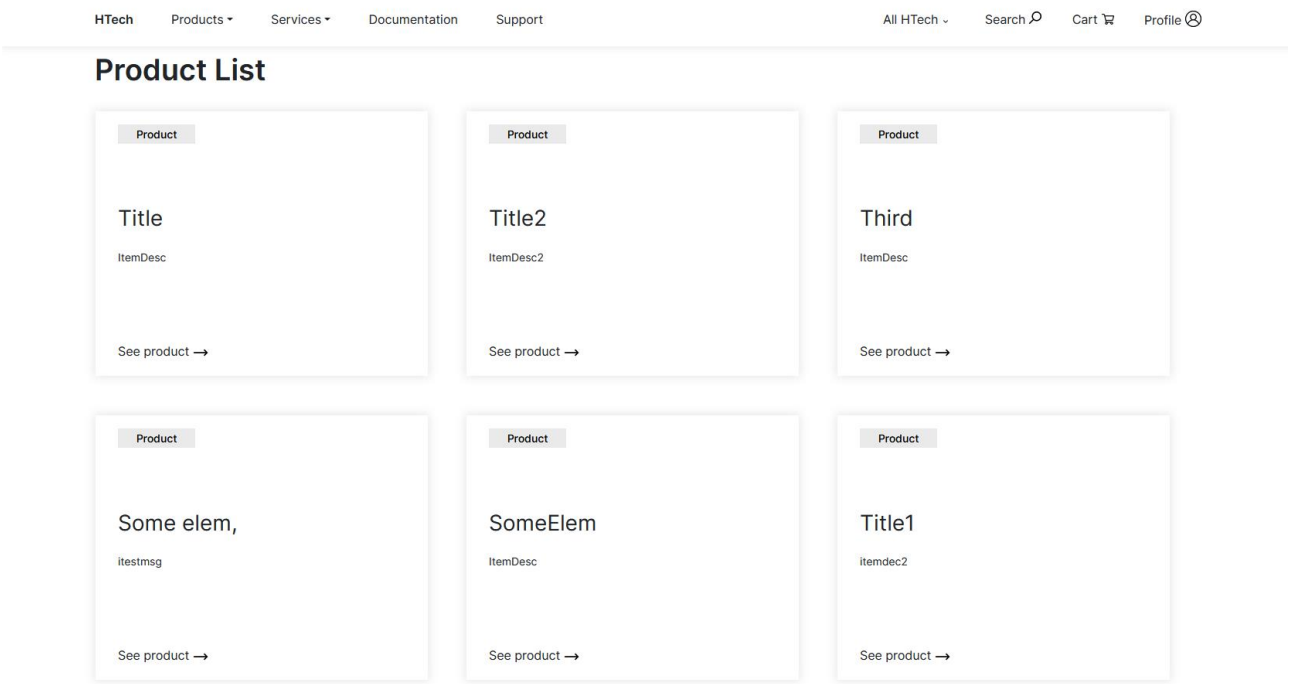

Рис. 2.52. Список елементів каталогу

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Визначення трудомісткості та вартості розробки програмного продукту

Початкові дані:

1. передбачуване число операторів програми – 1700;
2. коефіцієнт складності програми – 1,3;
3. коефіцієнт корекції програми в ході її розробки – 0,1;
4. годинна заробітна плата програміста – 421, 35 грн/год;

Згідно статистики заробітної плати програміста C#/.NET, що представлена на сайті DOU[48], середнє значення становить 2500$, що виходячи з поточного курсу валют (40,45) дорівнює 101 125 грн. Отже, виходячи з розрахунку, погодинна заробітна плата становить 421, 35 грн.

5. коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі – 1,2;
6. коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 1,2;
7. вартість машино-години ЕОМ – 0,48 грн/год.

Оцінка вартості проводилася з урахуванням часу, затраченого на написання програми впродовж 6 місяців. Ціна на електропостачання в Україні на поточний час становить 2,64 грн за 1 кВт/год, ураховуючи ПДВ. Споживання електроенергії пристроєм, на якому велась розробка (75 Вт на годину). Таким чином, витрати на електроенергію під час роботи складають 0,075*2,64=0,2 Інтернет підключення від провайдера Vega має щомісячну плату 200 грн. Вартість використання інтернету погодинно становить 200 / (30*24) = 0,28 грн/год. Отже, загальні вартість машино – години (ЕОМ) дорівнює 0,2+0,28= 0,48.

Нормування праці в процесі створення ПЗ істотно ускладнено в силу творчого характеру праці програміста. Тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_o + t_u + t_a + t_n + t_{отл} + t_д \text{, людино-годин,} \qquad (3.1)$$

де t_o - витрати праці на підготовку й опис поставленої задачі (приймається 50 людино-годин);

t_u - витрати праці на дослідження алгоритму рішення задачі;

t_a - витрати праці на розробку блок-схеми алгоритму;

t_n - витрати праці на програмування по готовій блок-схемі;

$t_{отл}$ - витрати праці на налагодження програми на ЕОМ;

$t_д$ - витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у програмному забезпеченні, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q \cdot C \cdot (1 + p), \qquad (3.2)$$

де q - передбачуване число операторів (1700);

C - коефіцієнт складності програми (1,3);

p - коефіцієнт корекції програми в ході її розробки (0,1).

Звідси умовне число операторів в програмі:

$$Q = 1{,}3 \cdot 1700 \cdot (1 + 0{,}1) = 2431$$

Витрати праці на вивчення опису задачі t_u визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q \cdot B}{(75..85) \cdot k}, \text{ людино-годин,} \qquad (3.3)$$

де B - коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі;

k - коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності. При стажі роботи від 3 до 5 років він складає 1,2.

Приймемо збільшення витрат праці внаслідок недостатнього опису завдання не більше 50% (B = 1.2). З урахуванням коефіцієнта кваліфікації k = 1,2, отримуємо витрати праці на вивчення опису завдання:

$$t_u = (2431 \cdot 1,2) / (75 \cdot 1,2) = 32,41 \text{ людино-годин}$$

Витрати праці на розробку алгоритму рішення задачі визначаються за формулою:

$$t_a = \frac{Q}{(20 \ldots 25) \cdot k}, \text{ людино-годин,} \qquad (3.4)$$

де Q – умовне число операторів програми;

k – коефіцієнт кваліфікації програміста.

Підставивши відповідні значення в формулу (3.2), отримаємо:

$$t_a = 2431 / (20 \cdot 1,2) = 101,29 \text{ людино-годин.}$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20\ldots25)\cdot k}, \text{ людино-годин,} \qquad (3.5)$$

$t_n = 2431/ (25 \cdot 1{,}2) = 81$ людино-годин.

Витрати праці на налагодження програми на ЕОМ:

- за умови автономного налагодження одного завдання:

$$t_{отл} = \frac{Q}{(4..5)\cdot k}, \text{ людино-годин,} \qquad (3.6)$$

$t_{отл} = 2431 / (5 \cdot 1{,}2) = 405{,}17$ людино-годин.

- за умови комплексного налагодження завдання:

$$t^k_{отл} = 1{,}5 \cdot t_{отл}, \text{ людино-годин,} \qquad (3.7)$$

$t^k_{отл} = 1{,}5 \cdot 405{,}17 = 607{,}76$ людино-годин.

Витрати праці на підготовку документації визначаються за формулою:

$$t_д = t_{др} + t_{до}, \text{ людино-годин,} \qquad (3.8)$$

де $t_{др}$ - трудомісткість підготовки матеріалів і рукопису:

$$t_{др} = \frac{Q}{(15..20)\cdot k}, \text{ людино-годин,} \qquad (3.9)$$

$t_{до}$ - трудомісткість редагування, печатки й оформлення документації:

$$t_{до} = 0{,}75 \cdot t_{др}, \text{ людино-годин,} \qquad (3.10)$$

Підставляючи відповідні значення, отримаємо:

$$t_{др} = 2431 / (18 \cdot 1{,}2) = 112{,}55 \text{ людино-годин.}$$

$$t_{до} = 0{,}75 \cdot 112{,}55 = 84{,}41 \text{ людино-годин.}$$

$$t_{д} = 112{,}55 + 84{,}41 = 196{,}96 \text{ людино-годин.}$$

Повертаючись до формули (3.1), отримаємо повну оцінку трудомісткості розробки програмного забезпечення:

$$t = 50 + 32{,}41 + 101{,}29 + 81 + 405{,}17 + 196{,}96 = 866{,}83 \text{ людино-годин.}$$

3.2. Розрахунок витрат на створення програми

Витрати на створення ПЗ $К_{ПО}$ включають витрати на заробітну плату виконавця програми $З_{ЗП}$ і витрат машинного часу, необхідного на налагодження програми на ЕОМ:

$$К_{ПО} = З_{ЗП} + З_{МВ}, \text{ грн,} \qquad (3.11)$$

Заробітна плата виконавців визначається за формулою:

$$З_{ЗП} = t \cdot С_{ПР}, \text{ грн,} \qquad (3.12)$$

де: t - загальна трудомісткість, людино-годин;

$С_{ПР}$ - середня годинна заробітна плата програміста, грн/година

З урахуванням того, що середня годинна зарплата програміста становить 421,35 грн / год, отримуємо:

$$З_{ЗП} = 866{,}83 \cdot 421{,}35 = 365\ 238{,}82 \text{ грн.}$$

Вартість машинного часу, необхідного для налагодження програми на ЕОМ, визначається за формулою:

$$З_{мв} = t_{отл} \cdot C_{мч}, \text{ грн,} \qquad (3.13)$$

де $t_{отл}$ - трудомісткість налагодження програми на ЕОМ, год;

$C_{мч}$ - вартість машино-години ЕОМ, грн/год (0,48).

Підставивши в формулу (3.3) відповідні значення, визначимо вартість необхідного для налагодження машинного часу:

$$З_{мв} = 405{,}17 \cdot 0{,}48 = 194{,}48 \text{ грн.}$$

Звідси витрати на створення програмного продукту:

$$К_{ПО} = 194{,}48 + 365\ 238{,}82 = 365\ 433{,}3 \text{ грн.}$$

Очікуваний період створення ПЗ:

$$T = \frac{t}{B_k \cdot F_p}, \text{ міс.,} \qquad (3.14)$$

де B_k - число виконавців (дорівнює 1);

F_p - місячний фонд робочого часу (при 40 годинному робочому тижні F_p=176 годин);

t – загальна трудомісткість, людино годин()

Звідси витрати на створення програмного продукту:

$$T = 866{,}83 / 1 \cdot 176 \approx 4{,}92 \text{ міс.}$$

Висновок: для розробки веб-орієнтованого додатку для інтернет-магазинун необхідно 866,83 людино-годин. Очікується, що процес створення програмного забезпечення триватиме приблизно 4,92 місяця, а загальні витрати на його розробку становитимуть 365 433,3грн.

ВИСНОВКИ

В кваліфікаційнії роботі було розроблено веб – орієнтований додаток – інтернет магазин для діяльності IT - компанії.

Провівши аналіз існуючих рішень в даній галузі, зроблено висновок, що в Україні відсутні місцеві глобальні аналоги, які мають конкуретні компанії за кордоном. Основною відмінністю яких є практичність в застосуванні та зручний в керуванні інтерфейс для взаємодії з компонентами системи і придбання товарів.

Практичне значення розробленого додатку полягає в покращенню взаємодії між клієнтами та компанією, а також створення місцевих сервісів що стануть глобальним аналогом, порівнюючи з існуючими рішеннями. Розроблена система надає можливості для користувачів:

– обирати сервіси або продукти та здійснювати покупки;

– керувати обраними продуктами і здійснювати оплату використовуючи міжнародну електронну платіжну систему;

– здійснювати модераторами перегляд та редагування користувачів та товарів, сторінок товарів.

В застосунку реалізована система аутентифікації та авторизації користувачів, що здійснює підтвердження особи і наявності прав доступу до ресурсів магазину.

Система розроблена згідно архітектурного шаблону MVC, з використанням інструментів фреймворку ASP.NET. Бізнес – логіка реалізована з застосуванням слою моделі та сервісів як елементів архітектури додатку, і використовують SQLite в якості бази даних для збереження інформації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Amazon AWS / URL: https://aws.amazon.com/. Дата звернення: 01.05.2024.

2. Microsoft Store / URL: https://www.microsoft.com/uk-ua/. Дата звернення: 01.05.2024.

3. Gandhi, V. A., & Kumbharana, C. K. 2021. Comparative study of Amazon EC2 and Microsoft Azure cloud architecture. *International Journal of Advanced Networking Applications (IJANA)*, 117. ISSN: 0975-0290.

4. Choudhary A, Verma PK, Rai P. A walkthrough of Amazon Elastic Compute Cloud (Amazon EC2): A Review. Int J Res Appl Sci Eng Technol. 2021 Nov;9(XI):93. ISSN: 2321-9653. IC Value: 45.98; SJ Impact Factor: 7.429.

5. GigaCloud / URL: https://gigacloud.ua/. Дата звернення 01.05.2024.

6. DeNovo / URL: https://denovo.ua/. Дата звернення 01.05.2024.

7. Блог компанії De Novo. Хмарний ринок України. / URL: https://denovo.ua/blog/ukrainian-cloud-market-2020. Дата звернення 01.05.2024.

8. VMware / URL: https://www.vmware.com/. Дата звернення 01.05.2024.

9. SAP / URL: https://www.sap.com/. Дата звернення: 05.05.2024

10. IBM. Information Security / URL: https://www.ibm.com/topics/information-security. Дата звернення: 25.05.2024.

11. Cisco. What Is Infromation Security? / URL: What Is Information Security (InfoSec)? - Cisco. Дата звернення: 23.05.2024.

12. ISO/IEC 27001:2022(en). Information security, cybersecurity and privacy protection — Information security management systems — Requirements. International Organization for Standardization, 2022.

13. Hatzivasilis G, Papaefstathiou I, Manifavas C. Password Hashing Competition - Survey and Benchmark. Dept. of Electronic & Computer Engineering, Technical University of Crete, Chania, Crete, Greece; Dept. of Informatics Engineering, Technological Educational Institute of Crete, Heraklion, Crete, Greece.

14. Sriramya P, Karthika RA. Department of Computer Science and Engineering, Saveetha School of Engineering, Saveetha University, Chennai, India. ARPN Journal of Engineering and Applied Sciences. VOL. 10, NO. 13, JULY 2020. ISSN 1819-6608.

15. Miller PM. TCP/IP The ultimate Protocol Guide: Volume 2 - Applications, access and data Security. Boca Raton, FL: BrownWalker Press; 2019. ISBN-10: 1-59942-493-2 (paper), ISBN-10: 1-59942-494-0 (ebook).

16. luwatosin HS. Client-Server Model. OSR J Comput Eng. 2020 Feb;16(1):67-71. e-ISSN: 2278-0661, p-ISSN: 2278-8727.

17. Rahim R, Aryza S, Wibowo P, Prasnowo MA, Djanggih H, Karinda K, Nasrudin N, Agustina I. Prototype File Transfer Protocol Application for LAN and Wi-Fi Communication. Int J Eng Technol. 2020; 7(2.13):345-347

18. Freeman, A. 2020. *Pro ASP.NET Core 3: Develop Cloud-Ready Web Applications Using MVC, Blazor, and Razor Pages*. Apress Media, LLC. ISBN-13 (pbk): 978-1-4842-5439-4. ISBN-13 (electronic): 978-1-4842-5440-0.

19. Wiley, John. 2021. Professional ASP.NET MVC 4. John Wiley & Sons, Inc. ISBN: 978-1-118-34846-8.

20. Freeman, A. 2019. Pro ASP.NET Core MVC (6th ed.). Apress. ISBN: 978-1-4842-0397-2.

21. Gaffney, K. P., Prammer, M., Brasfield, L., Hipp, D. R., Kennedy, D., & Patel, J. M. 2022. SQLite*:* past, present, and future. Proceedings of the VLDB Endowment, 15(12), 3535-3547. ISSN: 2150-8097. Award ID: 1835446. NSF-PAR ID: 10390276.

22. Lerman, J. 2019. Programming Entity Framework, Second Edition. O'Reilly Media, Inc. ISBN: 978-0-596-80726-9.

23. Smith, J. 2019. Entity Framework Core in Action. Manning Publications. ISBN: 9781617294563.

24. De Barros Lima, A.M.E.; Rodrigues, C.A.Q.; Haikal, D.S.A.; Silveira, M.F.; Mendes, D.C.; Oliviera, P.M.; Andrade, A.F.; de Feritas, C.V.; Pordeus, I.A. Desenvolvimento de um programa de computador para levantamentos

epidemiológicos sobre condições de saúde bucal. Unimontes Científica 2020. O'Reilly Media, 1st edition, November 18, 2019. ISBN-13: 978-1449312947.

25. Шreland, Christopher; Bowers, David; Newton, Michael; Waugh, Kevin. "Understanding Object-Relational Mapping: A Framework Based Approach." 2020. Department of Computing, The Open University, Milton Keynes, UK.

26. Zmaranda, Doina; Pop-Fele, Lucian-Laurentiu; Győrödi, Cornelia; Győrödi, Robert; Pecherle, George. "Performance Comparison of CRUD Methods using .NET Object Relational Mappers: A Case Study." International Journal of Advanced Computer Science and Applications (IJACSA), vol. 11, no. 1, 2020. Department of Computer Science and Information Technology, University of Oradea, Oradea, Romania. Faculty of Electrical Engineering and Information Technology, University of Oradea, Oradea, Romania.

27. Dapper ORM / URL: https://www.learndapper.com/. Дата звернення: 03.05.2024.

28. Hibernate ORM / URL: https://hibernate.org/orm/. Дата звернення: 03.05.2024.

29. C# Documentation / URL: https://learn.microsoft.com/en-us/dotnet/csharp/. Дата звернення: 03.05.2024.

30. Bootstrap Documentation / URL: https://getbootstrap.com/docs/. Дата звернення: 03.05.2024.

31. Smith, J. 2020. Mastering Bootstrap 4: A Comprehensive Guide to Building Responsive Web Pages. O'Reilly Media.

32. PayPal. / URL: https://www.paypal.com/ua/home. Дата звернення: 03.05.2024.

33. Платіжна система PayPal. / URL: https://uk.wikipedia.org/wiki/PayPal. Дата звернення 03.05.2024.

34. Журнал Forbes. Запуск PayPal в Україні / URL: https://forbes.ua/inside/paypal-golosno-zayaviv-pro-zapusk-v-ukraini-shcho-tse-oznachae-ta-chi-vsi-mozhut-skoristatisya-poyasnyue-forbes-18032022-4807. Дата звернення: 03.05.2024.

35. Google Pay / URL: https://pay.google.com/intl/uk_ua/about/. Дата звернення: 03.05.2024.

36. BEM Documentation / URL: https://en.bem.info/methodology/. Дата звернення: 03.05.2024.

37. BEM. Naming convention / URL: Naming convention / Methodology / BEM. Дата звернення: 25.05.2024.

38. JQuery Documentation / URL: https://api.jquery.com/jQuery.ajax/. Дата звернення: 03.05.2024.

39. VISA Home Page / URL: https://www.visa.com.ua/uk_UA. Дата звернення: 03.05.2024.

40. Mastercard Home Pahe / URL: https://www.mastercard.ua/uk-ua.html. Дата звернення: 03.05.2024.

41. Microsoft Documentation. Razor. / URL: https://learn.microsoft.com/ru-ru/aspnet/core/razor-pages/?view=aspnetcore-8.0&tabs=visual-studio. Дата звернення: 03.05.2024.

42. Android Developer Documentation. / URL: https://developer.android.com/topic/libraries/architecture/viewmodel. Дата звернення: 03.05.2024.

43. Microsoft Visual Studio / URL: https://visualstudio.microsoft.com. Дата звернення: 03.05.2024.

44. Visual Studio Code / URL: https://code.visualstudio.com/. Дата звернення: 03.05.2024.

45. Live Server Extension / URL: https://marketplace.visualstudio.com/items?itemName=ritwickdey.LiveServer . Дата звернення: 03.05.2024.

46. DB Browser for SQLite / URL: https://sqlitebrowser.org/. Дата звернення 03.05.2024.

47. Figma Documentation / URL: https://help.figma.com/hc/en-us. Дата звернення: 03.05.2024.

48. .NET DOU / URL: https://jobs.dou.ua/salaries. Дата звернення: 03.05.2024.

.

ДОДАТОК А

КОД ПРОГРАМИ

CartController.cs

```
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using WebApplication1.Models;
using WebApplication1.Services;
using WebApplication1.ViewModels;

namespace WebApplication1.Controllers
{
   public class CartController : Controller
   {
      private readonly IOrderService _orderService;
      private readonly ISession _session;

      public CartController(IHttpContextAccessor accessor, IOrderService orderService) {
         _session = accessor.HttpContext!.Session;
         _orderService = orderService;
      }
      public ActionResult Index()
      {

         if (_session.GetInt32("UserId") != null)
         {
            ItemService service = new (_orderService.GetOrdersByUserId((int)_session.GetInt32("UserId")!), _orderService);

            CartViewModel viewModel = new CartViewModel(service.MakeItemsList());

            return View(viewModel);
         }
         else
            return RedirectToAction("Index", "Login");
      }

      public IActionResult Details(int productId)
      {
         if (_session.GetInt32("UserId") != null)
         {
            ItemService service = new (_orderService.GetOrdersByUserId((int)_session.GetInt32("UserId")!), _orderService);

            return View(service.MakeItemDetails(productId));
         }
         else
            return RedirectToAction("Index", "Login");
      }

      public IActionResult Delete(int productId)
      {
         if (_session.GetInt32("UserId") != null)
         {
            List<Order> _orders = _orderService.GetOrdersByUserId((int)_session.GetInt32("UserId")!);
            _orders.RemoveAll(p => p.ProductId != productId);

            foreach (Order _order in _orders)
            {
               _orderService.DeleteOrder(_order.OrderId);
            }

            return RedirectToAction("Index");
         }
         else
```

```
            return RedirectToAction("Index", "Login");
        }

    }
}
```

HomeController.cs

```
using Microsoft.AspNetCore.Mvc;
using System.Diagnostics;
using WebApplication1.Models;
using WebApplication1.Services;
using WebApplication1.ViewModels;

namespace WebApplication1.Controllers
{
    public class HomeController : Controller
    {
        private readonly IUserService _userService;

        public HomeController(IUserService userService)
        {
            _userService = userService;
        }

        public IActionResult Index()
        {
            return View();
        }

        public IActionResult Privacy()
        {
            return View();
        }

        [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]
        public IActionResult Error()
        {
            return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });
        }
    }
}
```

ApplicationDbContext.cs

```
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using static Microsoft.EntityFrameworkCore.DbLoggerCategory.Database;

namespace WebApplication1.Models
{
    public class ApplicationDbContext : DbContext
    {
        public DbSet<User> Users { get; set; } = null!;
        public DbSet<Product> Products { get; set; } = null!;
        public DbSet<Order> Orders { get; set; } = null!;
        public DbSet<Role> Roles { get; set; } = null!;
        public DbSet<ProductDescription> ProductDescriptions { get; set; } = null!;

        public ApplicationDbContext() => Database.EnsureCreated();

        public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) : base(options)
        {
            Database.EnsureCreated();
        }
```

```
        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            optionsBuilder.UseSqlite("Data Source=access.db");
        }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            base.OnModelCreating(modelBuilder);

            modelBuilder.Entity<Order>()
                .HasOne(o => o.User)
                .WithMany(u => u.Orders)
                .HasForeignKey(o => o.UserId);

            modelBuilder.Entity<Order>()
                .HasOne(o => o.Product)
                .WithMany(p => p.Orders)
                .HasForeignKey(o => o.ProductId);

            modelBuilder.Entity<User>()
                .HasOne(o => o.Role)
                .WithMany(p => p.Users)
                .HasForeignKey(o => o.RoleId);

            modelBuilder.Entity<ProductDescription>()
                .HasOne(o => o.Product)
                .WithMany(p => p.ProductDescriptions)
                .HasForeignKey(o => o.ProductId);

        }
    }
}
```

PanelController.cs

```
using WebApplication1.Services;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using WebApplication1.Filters;
using WebApplication1.FormModels;
using WebApplication1.ViewModels;
using WebApplication1.Models;

namespace WebApplication1.Controllers
{
    [AuthorizeAccess("Root")]
    public class PanelController : Controller
    {
        private readonly ISession _session;

        private readonly IProductService _productService;
        private readonly IProductDescriptionsService _productDescriptionsService;

        private readonly IOrderService _orderService;
        private readonly IUserService _userService;

        private readonly IRoleService _roleService;
```

```
        private readonly int? _userId;
        public PanelController(IHttpContextAccessor accessor, IProductService productService,
IProductDescriptionsService descriptionsService, IOrderService orderService, IUserService
userService, IRoleService roleService) {
            _session = accessor.HttpContext!.Session;
            _productService = productService;
            _productDescriptionsService = descriptionsService;

            _orderService = orderService;
            _userService = userService;

            _roleService = roleService;

            _userId = _session.GetInt32("UserId");
        }

        public IActionResult Index()
        {
            return View();
        }

        public IActionResult ProductEditor() {
            return View();
        }

        public IActionResult UserEditor()
        {
            return View();
        }

        public IActionResult RoleDashboard()
        {
            return View(_roleService.GetAllRoles());
        }

        public IActionResult UserDashboard()
        {
            return View(_userService.GetAllUsers());
        }

        public IActionResult ProductDashboard()
        {
            ProductsViewModel productsModel = new (_productService);
            return View(productsModel);
        }

        public IActionResult OrderDashboard()
        {
            OrdersViewModel ordersView = new OrdersViewModel();

            foreach (Order order in _orderService.GetAllOrders())
```

```
            {
                string prodName = _productService.GetProductNameById(order.ProductId);
                ordersView.Orders.Add(new KeyValuePair<string, Order> (prodName, order) );
            }

            return View(ordersView);
        }

        [HttpPost]
        public IActionResult AddProduct(ProductFormModel productModel)
        {
            if(!ModelState.IsValid)
            {
                return BadRequest(ModelState);
            }

            ProductManager manager = new ProductManager(productModel, _productService);

            if (!manager.CreateProduct())
            {
                return BadRequest("Something went wrong while adding a product. Try again.");
            }


            return RedirectToAction("ProductEditor");
        }

        [HttpPost]
        public IActionResult AddProductDesc(ProductDescFormModel descModel)
        {
            if(!ModelState.IsValid)
            {
                return BadRequest(ModelState);
            }

            ProductManager manager = new ProductManager(descModel,
_productDescriptionsService);

            if (!manager.CreateProductDesc())
            {
                return BadRequest("Something went wrong while adding description. Try again.");
            }


            return RedirectToAction("ProductEditor");
        }
    }
}
```

ProductsController.cs

```
using Microsoft.AspNetCore.Http;
```

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.ViewEngines;
using Microsoft.AspNetCore.Mvc.ViewFeatures;
using System.Diagnostics;
using System.Security.Cryptography.Xml;
using WebApplication1.Models;
using WebApplication1.Services;
using WebApplication1.ViewModels;

namespace WebApplication1.Controllers
{
   public class ProductsController : Controller
   {
      private readonly IOrderService _orderService;
      private readonly IProductService _productService;
      private readonly IProductDescriptionsService _productDescriptionsService;

      private readonly ICompositeViewEngine _viewEngine;
      private readonly ISession _session;

      public ProductsController(IHttpContextAccessor accessor, IOrderService orderService, ICompositeViewEngine
viewEngine, IProductService productService, IProductDescriptionsService productDescriptionsService)
      {
         _session = accessor.HttpContext!.Session;
         _orderService = orderService;
         _viewEngine = viewEngine;
         _productService = productService;
         _productDescriptionsService = productDescriptionsService;
      }

      public IActionResult Index()
      {
         ProductGridService gridService = new ProductGridService(_productService.GetAllProducts());

         return View(gridService.CreateGridView());
      }

      [HttpGet("Products/Pages/{page}")]
      public IActionResult ShowPage(string page)
      {
         bool isViewExist = _viewEngine.FindView(ControllerContext, page, false).Success;

         if (string.IsNullOrEmpty(page) || !isViewExist)
         {
            return NotFound();
         }

         return View(page);
      }

      public IActionResult AddToCart(int productId)
      {
         if (_session.GetInt32("UserId") != null)
         {
            int userId = (int)_session.GetInt32("UserId")!;

            _orderService.CreateOrder(new Order { UserId = userId, ProductId = productId });

         }else return BadRequest();

         return RedirectToAction("Index", "Cart");
      }

      public IActionResult Details(int productId)
      {
```

```
            DescBuilderService descBuilder = new DescBuilderService(productId,
_productDescriptionsService.GetDescriptionsByProductId(productId));

            return View(descBuilder.MakeDescViewModel());
        }

        public IActionResult Logout()
        {
            _session.Clear();
            return RedirectToAction("Index", "Home");
        }

        [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]
        public IActionResult Error()
        {
            return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });
        }
    }
}
```

_Layout.cshtml

```
@using Microsoft.AspNetCore.Http
@using WebApplication1.Services

@inject IHttpContextAccessor Accessor
@inject IAuthService AuthService

@{
    ISession _session = Accessor.HttpContext!.Session;

    int? _userId = _session.GetInt32("UserId");
}

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewData["Title"]</title>

    <!-- Bootstrap CSS -->
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-
EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC" crossorigin="anonymous">

    <link rel="stylesheet" type="text/css" href="~/css/main.css">
    <link rel="stylesheet" type="text/css" href="~/css/fonts.css">

    <link rel="stylesheet" type="text/css" href="~/css/header.css">
    <link rel="stylesheet" type="text/css" href="~/css/header-fixed.css">

    <link rel = "stylesheet" type="text/css" href = "~/css/footer.css">

    @RenderSection("Styles", true)
</head>
<body>
    @if (ViewBag.excludeHeader != null && !ViewBag.excludeHeader){
        <header>
            <div class="header__container">
                <div class="header__menu-left">
                    <span class="semi-bold"><a href = "@Url.ActionLink("Index", "Home")">HTech</a></span>

                    <span class="dropdown-toggle" type="button" data-bs-toggle="dropdown" aria-
expanded="false">Products</span>
                    <ul class="dropdown-menu">
```

```
                <li><a class="dropdown-item" type="button" href="@Url.Action("ShowPage", "Products", new { page =
"Management" })">Relationship Management</a></li>
                <li><a class="dropdown-item" type="button" href="@Url.Action("ShowPage", "Products", new { page =
"Project" })">Project Management</a></li>
                <li><a class="dropdown-item" type="button" href="@Url.Action("ShowPage", "Products", new { page =
"Accounting" })">Accounting and Finance</a></li>
                <li><a class="dropdown-item" type="button" href="@Url.Action("ShowPage", "Products", new { page =
"Databases" })">Databases</a></li>
                <li><a class="dropdown-item" type="button" href="@Url.Action("ShowPage", "Products", new { page =
"Intelligense" })">Intelligence Tools</a></li>
                <li><hr class="dropdown-divider"></li>
                <li><a class="dropdown-item" type="button" href="@Url.Action("ShowPage", "Products", new { page =
"Cybersecurity" })">Cybersecurity</a></li>
                <li><a class="dropdown-item" type="button" href="@Url.Action("Index", "Products")">See All</a></li>
              </ul>

              <span class="dropdown-toggle" type="button" data-bs-toggle="dropdown" aria-
expanded="false">Services</span>
              <ul class="dropdown-menu">
                <li><a class="dropdown-item" type="button" href="@Url.Action("ShowPage", "Products", new { page =
"Services" })">Computing Services</a></li>
                <li><a class="dropdown-item" type="button" href="@Url.Action("ShowPage", "Products", new { page =
"Storage" })">VPC Network</a></li>
                <li><a class="dropdown-item" type="button" href="@Url.Action("ShowPage", "Products", new { page =
"Network" })">Cloud Storage</a></li>
              </ul>
              <span>Documentation</span>
              <span>Support</span>
            </div>
            <div class="header__menu-right">
              <span>All HTech </span>
              <span><a>Search</a></span>
              <span><a href="@Url.ActionLink("Index", "Cart")">Cart</a></span>

              @if (_userId == null)
              {
                <span>
                  <a href="@Url.ActionLink("Index", "Login")">Sign In</a>
                </span>
              }
              else
              {
                <span type="button" data-bs-toggle="dropdown" aria-expanded="false">Profile</span>
                <ul class="dropdown-menu">

                  @if(AuthService.IsUserHasAccess((int)_userId, "Root"))
                  {
                    <li><a class="dropdown-item" type="button" href="@Url.Action("Index", "Panel")">Panel</a></li>
                  }

                  <li><a class="dropdown-item" type="button" href="@Url.Action("Logout", "Products")">Logout</a></li>
                </ul>
              }

            </div>
          </div>
        </header>
        }
        <main role="main" class="pb-3">
          @RenderBody()
        </main>

      @if (ViewBag.excludeFooter != null && !ViewBag.excludeFooter)
      {
        <footer class="d-flex">
```

```
        <div class="footer__container">
         <div class="d-flex">
            <div class="col-3">
              <ul>
                <li class="bold f-16">Cloud</li>
                <li>VPC Network</li>
                <li>Storage</li>
                <li>Computing Services</li>
              </ul>
            </div>
            <div class="col-3">
              <ul>
                <li class="bold f-16">Business</li>
                <li>CRM System</li>
                <li>Management solution</li>
                <li>Accounting and Finance</li>
              </ul>
            </div>
            <div class="col-3">
              <ul>
                <li class="bold f-16">Products</li>
                <li>Relational Databases</li>
                <li>Cybersecurity</li>
                <li>Business Intelligense Tools</li>
              </ul>
            </div>
            <div class="col-3">
              <ul>
                <li class="bold f-16">Company</li>
                <li>Terms of use</li>
                <li>Privacy</li>
                <li>Policy</li>
                <li>HTech apps</li>
              </ul>
            </div>
          </div>
          <div class="row">
            <div class="d-flex col-12 justify-content-end">
              <ul class="d-flex">
                <li>Sitemap</li>
                <li>Contact HTech</li>
                <li>@@HTech 2024</li>
              </ul>
            </div>
          </div>
        </div>
      </footer>
    }
    <!-- jQuery -->
    <script src="https://code.jquery.com/jquery-3.7.1.min.js" integrity="sha256-
/JqT3SQfawRcv/BIHPThkBvs0OEvtFFmqPF/lYI/Cxo=" crossorigin="anonymous"></script>

    <!-- Bootstrap JS -->
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js" integrity="sha384-
MrcW6ZMFYlzcLA8Nl+NtUVF0sA7MsXsP1UyJoMp4YLEuNSfAP+JcXn/tWtIaxVXM"
crossorigin="anonymous"></script>

    @RenderSection("Scripts", true)

    <script src = "~/js/scrollAnchor.js"> </script>
</body>
</html>
```

ProductManager.cs

```
using WebApplication1.FormModels;
using WebApplication1.Models;

namespace WebApplication1.Services
{
   public class ProductManager
   {
      private readonly ProductFormModel? _productModel;
      private readonly ProductDescFormModel? _productDescModel;

      private readonly IProductService? _productService;
      private readonly IProductDescriptionsService? _productDescriptionsService;

      public ProductManager(ProductFormModel productModel, IProductService productService) {
         _productModel = productModel;
         _productService = productService;
      }

      public ProductManager(ProductDescFormModel productDescFormModel, IProductDescriptionsService
productDescriptionsService)
      {
         _productDescriptionsService = productDescriptionsService;
         _productDescModel = productDescFormModel;
      }

      private bool isFieldsValid()
      {
         if(_productModel == null) return false;

         if (_productModel.Name == null ||
            _productModel.Status == null ||
            _productModel.Price == null ||
            _productModel.ProductType == null) {

            return false;
         }

         if(!float.TryParse(_productModel.Price, out _))
            return false;

         return true;
      }

      private bool isDescFieldsValid()
      {
         if (_productDescModel == null) return false;

         if (_productDescModel.SectionDesc == null ||
            _productDescModel.SectionDesc == null ||
            _productDescModel.ProductId == null ||
            _productDescModel.SectionOrder == null)
         {
            return false;
         }

         if(!int.TryParse(_productDescModel.ProductId, out _))
         {
            return false;
         }

         if (!int.TryParse(_productDescModel.SectionOrder, out _))
         {
            return false;
         }
```

```
    return true;
}

public bool CreateProduct()
{
    Product? product = buildProduct();

    if (product == null) {
        return false;
    }

    try
    {
        _productService!.CreateProduct(product);
    }
    catch
    {
        return false;
    }

    return true;
}

public bool CreateProductDesc()
{
    ProductDescription? desc = buildDesc();

    if (desc == null)
    {
        return false;
    }

    try
    {
        _productDescriptionsService.CreateProductDescription(desc);
    }
    catch
    {
        return false;
    }

    return true;
}

private Product? buildProduct()
{
    if (!isFieldsValid()) return null;

    Product product = new Product {
        Name = _productModel!.Name,
        Description = _productModel.Description,
        Price = float.Parse(_productModel.Price!),
        ItemDescription = _productModel.ItemDescription,
        ProductType = _productModel.ProductType,
        Status = _productModel.Status == "Enabled"
    };

    return product;
}

private ProductDescription? buildDesc()
{
    if (!isDescFieldsValid()) return null;

    ProductDescription desc = new ProductDescription()
```

```
        {
            ProductId = int.Parse(_productDescModel!.ProductId!),
            SectionName= _productDescModel!.SectionName,
            SectionDescription = _productDescModel.SectionDesc,
            SectionOrder = int.Parse(_productDescModel!.SectionOrder!)
        };


        return desc;
    }
  }
}
```

UserService.cs

```
using WebApplication1.Models;

namespace WebApplication1.Services
{
   public class UserService : IUserService
   {
      private readonly ApplicationDbContext _dbContext;

      public UserService(ApplicationDbContext dbContext)
      {
         _dbContext= dbContext;
      }

      public void CreateUser(User user)
      {
         _dbContext.Users.Add(user);
         _dbContext.SaveChanges();
      }

      public User GetUserById(int userId)
      {
         return _dbContext.Users.Find(userId);
      }

      public List<User> GetAllUsers()
      {
         return _dbContext.Users.ToList();
      }

      public User GetUserByEmail(string email)
      {
         return _dbContext.Users.FirstOrDefault(u => u.Email == email);
      }

      public Role GetUserRoleById(int userId)
      {
         User user = _dbContext.Users.Find(userId);

         Role role = _dbContext.Roles.Find(user.RoleId);
```

```
            return role;
        }

        public void UpdateUser(User user)
        {
            var existingUser = _dbContext.Users.Find(user.UserId);

            if (existingUser != null)
            {
                _dbContext.Users.Update(user);
                _dbContext.SaveChanges();
            }
        }

        public void DeleteUser(int userId)
        {
            var user = _dbContext.Users.Find(userId);

            if (user != null)
            {
                _dbContext.Users.Remove(user);
                _dbContext.SaveChanges();
            }
        }
    }
}
```

AuthorizeAccessAttribute.cs

```
using Microsoft.AspNetCore.Mvc.Filters;
using Microsoft.AspNetCore.Mvc;
using WebApplication1.Models;
using WebApplication1.Services;
using WebApplication1.ViewModels;

namespace WebApplication1.Filters
{
    public class AuthorizeAccessAttribute : ActionFilterAttribute
    {
        private readonly string _role;

        private ISession? _session;
        private IAuthService? _authService;

        public AuthorizeAccessAttribute(string role)
        {
            _role = role;
        }

        private void setServices(ActionExecutingContext context)
        {
            _authService = context.HttpContext.RequestServices.GetRequiredService<IAuthService>();
            _session = context.HttpContext.Session;

            if (_authService == null || _session == null)
                throw new Exception("Services cannot be nullable");
        }
```

```
        public override void OnActionExecuting(ActionExecutingContext context)
        {
            setServices(context);

            int? userId = _session!.GetInt32("UserId");
            if (userId == null || !_authService!.IsUserHasAccess(userId.Value, _role))
            {
                context.Result = new ViewResult { ViewName = "Forbidden" };
            }
        }
    }
}
```

ProductGridService.cs

```
using WebApplication1.FormModels;
using WebApplication1.Models;
using WebApplication1.ViewModels;

namespace WebApplication1.Services
{
    public class ProductGridService
    {
        private readonly List<Product> _products;

        private const int _gridLength = 3;
        private int _gridHeight;
        private int _gridItemsCount;

        public ProductGridService(List<Product> products)
        {
            _products = products;
            _gridItemsCount = _products.Count;


            _gridHeight = (int)Math.Ceiling((double) ( ( (double)_gridItemsCount) / ( (double)_gridLength) ) );
        }

        private Product[,] buildGrid()
        {
            Product[,] productGrid = new Product[_gridHeight, _gridLength];

            IEnumerator<Product> iterator = _products.GetEnumerator();
            iterator.MoveNext();

            for (int i = 0, count = 0; i < _gridHeight; i++)
            {
                for (int j = 0; j < _gridLength && count < _gridItemsCount; j++, iterator.MoveNext())
                {
                    productGrid[i, j] = iterator.Current;
                    count++;
                }
            }

            return productGrid;
        }

        public ProductsGridViewModel CreateGridView()
        {
            ProductsGridViewModel gridViewModel = new ProductsGridViewModel(_gridLength, _gridHeight,
_gridItemsCount, buildGrid());

            return gridViewModel;
        }
    }
```

```
}
```

ItemService.cs

```
using WebApplication1.Models;
using WebApplication1.Services;
using WebApplication1.ViewModels;

namespace WebApplication1.Services
{
   public class ItemService
   {
     private readonly List<Order> _orders;
     private readonly IOrderService _orderService;


     public ItemService(List<Order> orders, IOrderService orderService) {
       _orders = orders;
       _orderService = orderService;
     }

      private CartItemViewModel makeItem(int productId, int orderId)
      {
         int count = _orders.Count(o => o.ProductId == productId);

         Product product = _orderService.GetProductByOrderId(orderId);

         CartItemViewModel item = new CartItemViewModel
         {
            productId= productId,
            Count = count,
            Description = product.Description,
            Name = product.Name,
            Price = product.Price
         };

         return item;
      }

      public CartItemViewModel MakeItemDetails(int productId)
      {

         int orderId = _orders.Find(o => o.ProductId == productId).OrderId;

         return makeItem(productId, orderId);
      }

     public List<CartItemViewModel> MakeItemsList()
     {
         List<CartItemViewModel> items = new List<CartItemViewModel>();

         foreach (var item in _orders)
```

```
            {
                CartItemViewModel cartItem = makeItem(item.ProductId, item.OrderId);

                if (cartItem != null && items.Count(i => i.Name == cartItem.Name) == 0)
                {
                    items.Add(cartItem);
                }
            }

            return items;
        }

    }
}
```

DescBuilderService.cs

```
using WebApplication1.Models;
using WebApplication1.Services;
using WebApplication1.ViewModels;

namespace WebApplication1.Services
{
    public class DescBuilderService
    {
        private readonly List<ProductDescription> _productDesc;
        private readonly int _productId;
        public DescBuilderService(int productId, List<ProductDescription> productDesc) {

            _productDesc = productDesc;
            _productId = productId;
        }

        private List<KeyValuePair<string, string>> buildProductDesc()
        {
            List<ProductDescription> sortedItems = _productDesc.OrderBy(d => d.SectionOrder).ToList();

            List<KeyValuePair<string, string>> sections = new List<KeyValuePair<string, string>>();

            foreach(ProductDescription desc in sortedItems)
            {
                sections.Add(new KeyValuePair<string, string>(desc.SectionName, desc.SectionDescription));
            }

            return sections;
        }

        public ProductDescViewModel MakeDescViewModel()
        {
            var desc = buildProductDesc();

            KeyValuePair<string, string> title = desc.First();
            desc.Remove(title);

            return new ProductDescViewModel(_productId, desc, title);
        }
    }
}
```

CartDetails.chtml

```
@using WebApplication1.Models
@using WebApplication1.ViewModels

@model CartItemViewModel

@{
    ViewData["Title"] = "Products";

    ViewBag.excludeHeader = false;
    ViewBag.excludeFooter = false;
}

@section Styles{
    <link rel="stylesheet" type="text/css" href="~/css/cart.css">
    <link rel="stylesheet" type="text/css" href="~/css/cart-details.css">
}


<div class="cart">
    <div class="cart__container">
        <h2>@Model.Name</h2>


        <div class="d-flex">
            <div class="col-8">
                <div class="cart-divider col-12"></div>
                    <table>

                        <tr>
                            <td class="table__desc">PRODUCT INFO:</td>
                            <td>Each</td>
                            <td>Quantity</td>
                            <td>Total</td>
                        </tr>
                        <tr>
                            <td class="color-grey">
                                <div class="table__column_desc ">@Model.Description</div>
                            </td>
                            <td class="semi-bold">$@Model.Price</td>
                            <td>
                                <button class ="d-flex">
                                    <div class="col-3 bold">-</div>
                                    <div class="col-6">@Model.Count</div>
                                    <div class="col-3 bold">+</div>
                                </button>
                            </td>
                            <td>$@(Model.Price * Model.Count)</td>
                        </tr>
                        <tr class="color-grey">
                            <td class="table__control">
                                <span class="pointer">Save</span>
                                <span class="pointer">Back</span>
                            </td>
                        </tr>
                    </table>

                    <div class="cart-divider col-12"></div>
                <div class="d-flex">
                    <div class="col-1"><span class="semi-bold">@Model.Count items</span></div>
                    <div class="col-10 d-flex justify-content-end"><span class="semi-bold">$@(Model.Count *
Model.Price)</span></div>
                </div>
            </div>
        </div>
```

```
    </div>
</div>


@section Scripts{

}
```

ProductEditor.chtml

```
@{
    ViewData["Title"] = "Editor";

    ViewBag.excludeHeader = false;
    ViewBag.excludeFooter = false;

}

@section Styles{
    <link rel="stylesheet" type="text/css" href="~/css/productEditor.css">
}

<div class="editor">
    <div class="editor__container">
        <h2>Product Editor</h2>
        <div class="editor__ctrl">
            <div class="editor__ctrl-btn d-flex">
                <button id="save" type="submit" class="bold f-16">SAVE</button>
                <button class="bold f-16">CANCEL</button>
            </div>
        </div>
        <div class="editor__subpanel">
            <div class="editor__sub-ctrl">
                <div class="d-flex">
                    <button class="f-14 bold">Create</button>
                    <button class="f-14 bold">Update</button>
                </div>
            </div>
            <div class="editor__main-panel">
                <form id="prodInfo" method="POST" action="@Url.ActionLink("AddProduct", "Panel")">
                    <div>
                        <div class="editor__prod-info d-flex">
                            <span class="f-16 align-self-center">PRODUCT INFO</span>
                        </div>

                        <div class ="editor__info-data">

                            <table class="col-12">
                                <tr>
                                    <td><span class="req">Name:</span></td>
                                    <td><span><input name="name" class="form-control field" required/></span></td>
                                    <td><span>Description:</span></td>
                                    <td><input name="description" class="form-control field" required/></td>
                                </tr>

                                <tr>
                                    <td><span class="req">Status:</span></td>
                                    <td>
                                        <select name="status" class="form-select select-size" aria-label="Default select" required>
                                            <option selected>Disabled</option>
                                            <option>Enabled</option>
                                        </select>
                                    </td>
                                    <td><span>Item Description</span></td>
                                    <td><input name="ItemDescription" class="form-control field" /></td>
```

```
            </tr>

            <tr>
              <td><span>Product type:</span></td>
              <td>
                <select name="ProductType" class="form-select select-size" aria-label="Default select">
                  <option selected>Product</option>
                  <option>Cloud</option>
                  <option>Service</option>
                </select>
              </td>
              <td><span class="req">Price:</span></td>
              <td><input name="price" class="form-control field field_s" placeholder="365,84" required></td>
            </tr>
          </table>
        </div>
      </div>
</form>

<form id="prodDesc" method="POST" action="@Url.ActionLink("AddProductDesc", "Panel")">
    <div class="sub-gap">
        <div class="editor__prod-info d-flex">
            <span class="f-16 align-self-center">PRODUCT DESCRIPTION</span>
        </div>

        <div class="editor__info-data">

            <div class="d-flex">
                <table class="col-6">
                    <tr>
                        <td><span class="req">Product Id:</span></td>
                        <td><span><input name="ProductId" class="form-control field" required /></span></td>
                    </tr>

                    <tr>
                        <td><span class="req">Section name:</span></td>
                        <td>
                            <input name="SectionName" class="form-control field" required />
                        </td>

                    </tr>

                    <tr>
                        <td><span>Section Order:</span></td>
                        <td>
                            <select name="SectionOrder" class="form-select select-size" aria-label="Default select" required>
                                @for(int i = 1; i < 11; i++){
                                    <option>@i</option>
                                }
                            </select>
                        </td>
                    </tr>
                </table>

                <table class="col-6">
                    <tr>
                        <td class="d-flex"><span class="req align-self-start">Description:</span></td>
                        <td><textarea name="SectionDesc" class="form-control field area" required></textarea></td>
                    </tr>
                </table>
            </div>

            <button class="add-btn">ADD</button>
        </div>
    </div>
```

```
            </form>
          </div>
        </div>
    </div>
</div>

@section Scripts{
    <script src="~/js/formEditor.js"></script>
}
```

CartIndex.chtml

```
@using WebApplication1.Models
@using WebApplication1.ViewModels


@Model CartViewModel


@{
    ViewData["Title"] = "Products";

    ViewBag.excludeHeader = false;
    ViewBag.excludeFooter = false;

}

@section Styles{
    <link rel="stylesheet" type="text/css" href="~/css/cart.css">
}


<div class="cart">
    <div class="cart__container">
        <h2>My Cart</h2>


        <div class="d-flex">
            <div class="col-8">
                <div class="cart-divider col-12"></div>
                @foreach(CartItemViewModel cartItem in Model.Items){
                    <table>

                        <tr>
                            <td class="table__desc">@cartItem.Name</td>
                            <td>Each</td>
                            <td>Quantity</td>
                            <td>Total</td>
                        </tr>
                        <tr>
                            <td class="color-grey">
                                <div class="table__column_desc ">@cartItem.Description</div>
                            </td>
                            <td class="semi-bold">$@cartItem.Price</td>
                            <td>@cartItem.Count</td>
                            <td>$@(cartItem.Count * cartItem.Price)</td>
                        </tr>
                        <tr class="color-grey">
                            <td class="table__control">
                                <span class="pointer"><a href="@Url.ActionLink("Details", "Cart", new { ProductId =
cartItem.productId } )">Edit</a></span>
                                    <span class="pointer"><a href="@Url.ActionLink("Delete", "Cart", new { ProductId =
cartItem.productId } )">Remove</a></span>
                            </td>
```

```
                </tr>
            </table>

            <div class="cart-divider col-12"></div>
        }
        <div class="d-flex">
            <div class="col-1"><span class="semi-bold">@Model.getTotalCount() items</span></div>
            <div class="col-10 d-flex justify-content-end"><span class="semi-
bold">$@Model.getTotalPrice()</span></div>
        </div>
    </div>
    <div class="col-4 d-flex justify-content-center">
        <div class="promo-container align-items-start">
            <span>ENTER PROMO CODE</span>
            <div class="d-flex">
                <input placeholder="Promo Code">
                <button class="semi-bold">Submit</button>
            </div>

            <div class="promo__discount d-flex">
                <div class="col-10 color-grey f-14 align-items-end">Discount</div>
                <div class="col-1 color-grey f-14 align-items-end">$0</div>
            </div>

            <div class="promo__discount promo__tax d-flex">
                <div class="col-10 color-grey f-14 align-items-end">Tax</div>
                <div class="col-1 color-grey f-14 align-items-end">TBD</div>
            </div>

            <div class="estimated">
                <div class="estimated__container d-flex">
                    <div class="col-10 align-items-end">Estimated Total</div>
                    <div class="col-2 align-items-end">$@Model.getTotalPrice()</div>
                </div>
            </div>

            <div id = "btn-cont" style="margin-top: 25px;"></div>
            <!--<button class="checkout-btn semi-bold">Checkout</button> -->
        </div>
    </div>
</div>

</div>


@section Scripts{
    <script src="https://www.paypal.com/sdk/js?client-id=AZq6CewFGGS5-
Yb5fp67NhHNEPZ4UwsWGJD_hPHcrjDjtMb&currency=USD&buyer-country=US&merchant-
id=6954N&components=googlepay"></script>
    <script async src="https://pay.google.com/gp/p/js/pay.js" onload="onGooglePayLoaded()"></script>


<script>
    let paymentsClient;

    function getGooglePaymentsClient() {
        if (!paymentsClient) {
            paymentsClient = new google.payments.api.PaymentsClient({ environment: 'TEST' });
        }
        return paymentsClient;
    }

    const baseCardPaymentMethod = {
        type: 'CARD',
```

```
        parameters: {
            allowedAuthMethods: ['PAN_ONLY', 'CRYPTOGRAM_3DS'],
            allowedCardNetworks: ['MASTERCARD', 'VISA']
        }
    };

    const isReadyToPayRequest = Object.assign(
        {},
        {
            apiVersion: 2,
            apiVersionMinor: 0,
            allowedPaymentMethods: [baseCardPaymentMethod]
        }
    );

    function onGooglePayLoaded() {
        const paymentsClient = getGooglePaymentsClient();

        paymentsClient.isReadyToPay(isReadyToPayRequest)
            .then(function (response) {
                if (response.result) {
                    addGooglePayButton();
                }
            })
            .catch(function (err) {
                console.error(err);
            });
    }

    function addGooglePayButton() {
        const paymentsClient = getGooglePaymentsClient();
        const button = paymentsClient.createButton({
            onClick: onGooglePaymentButtonClicked,
            allowedPaymentMethods: [baseCardPaymentMethod]
        });
        document.getElementById('btn-cont').appendChild(button);
    }

    function onGooglePaymentButtonClicked() {
        const paymentDataRequest = Object.assign({}, isReadyToPayRequest, {
            transactionInfo: {
                totalPriceStatus: 'FINAL',
                totalPrice: '1.00',
                currencyCode: 'USD'
            }
        });

        paymentsClient.loadPaymentData(paymentDataRequest)
            .then(function (paymentData) {
                // Handle the response from Google Pay
                console.log(paymentData);
            })
            .catch(function (err) {
                console.error(err);
            });
    }
</script>
```

ДОДАТОК Б

ВІДГУК

Керівника економічного розділу

на кваліфікаційну роботу бакалавра на тему:

« Розробка веб-додатку для он-лайн магазину для ІТ-компанії на ASP.NET »

Студентки групи 122-20-1 Райгородської Дар'ї Олегівни

Керівник економічного розділу

доц. каф. ПЕП та ПУ, к.е.н **Л.В. Касьяненко**

ДОДАТОК В

ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ

Ім’я файла	Опис
Пояснювальні документи	
Райгородська.doc	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Райгородська.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF
Програма	
Райгородська.zip	Архів. Містить коди програми і откомпільовану програму
Презентація	
Райгородська.ppt	Презентація кваліфікаційної роботи