

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)
на тему:

«Інтернет-магазин кондитерських виробів»

Виконав студент групи ПЗп-20-1
ГАЄВСЬКИЙ Денис Олегович

Керівник роботи:
ЧЕПЮК Ларіна Олексіївна

Рецензент:
ЧИЖМОТРЯ Олексій Володимирович

Житомир – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення
Тетяна ВАКАЛЮК
«04» вересня 2024 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ГАСВСЬКИЙ Денис Олегович**

Керівник роботи: **ЧЕПЮК Ларіна Олексіївна**

Тема роботи: **«Інтернет-магазин кондитерських виробів»**,
затверджена Наказом закладу вищої освіти від **«02» вересня 2024 р., №433/с**
Вихідні дані для роботи: **об'єктом дослідження є додаток з використанням веб-технологій, спрямований на створення вітрини кондитерських товарів, а предметом дослідження є архітектура та компонентна структура веб-сайту.**

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Локтікова Т.М.	05.09.2024р.	05.09.2024р.
2	Локтікова Т.М.	05.09.2024р.	05.09.2024р.
3	Локтікова Т.М.	05.09.2024р.	05.09.2024р.

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з веб-додатку та пояснювальної записки. Пояснювальна записка до випускної роботи містить 74 сторінок, 37 ілюстрацій та 12 таблиць.

Метою роботи є створення інтернет-магазину кондитерських виробів із використанням сучасних технологій, а також розробка власних методів для побудови гнучкої архітектури веб-додатку.

Методи, використані під час створення додатку, включають в себе проектування на веб-орієнтованих мовах програмування та розробку бази даних для зручної взаємодії користувачів і адміністраторів з додатком.

Предмет дослідження – архітектура веб-додатку та логіка його роботи. Дослідження спрямоване на вивчення технологій та методів, що дозволяють забезпечити надійний та ефективний збут товару в рамках інтернет-магазину.

Результати розробки полягають у створенні інтерактивного та привабливого сайту для справжніх любителів солоденького.

КЛЮЧОВІ СЛОВА: ВЕБ-САЙТ, ІНТЕРНЕТ-МАГАЗИН, КОНДИТЕРСЬКІ ВИРОБИ, ЧАТ, MVC, PHP.

					ІПЗ.КР.Б-121-24-ІПЗ						
Зм.	Арк.	№ докум.	Підпис	Дата							
Розроб.		Гаєвський Д.О.			Інтернет-магазин кондитерських виробів. Пояснювальна записка			Літ.	Арк.	Аркуші	
Керівник		Чепюк Л.О.								3	74
Рецензен.		Чижмотря О.В.						Житомирська політехніка, група ІПЗп-20-1			
Зав. каф.		Вакалюк Т.А.									

ABSTRACT

The final bachelor's thesis consists of a web application and an explanatory note. The explanatory note to the final thesis contains 74 pages, 37 illustrations and 12 tables.

The purpose of the work is to create an online confectionery store using modern technologies, as well as to develop own methods for building a flexible web application architecture.

The methods used to create the application include designing in web oriented programming languages and developing a database for convenient interaction of users and administrators with the application.

The subject of the research is the architecture of a web application and the logic of its operation. The research is aimed at studying technologies and methods that allow to ensure reliable and efficient sales of goods within an online store.

The results of the development are the creation of an interactive and attractive website for true sweet lovers.

KEYWORDS: WEBSITE, ONLINE STORE, CONFECTIONERY, CHAT, MVC, PHP.

ЗМІСТ

РЕФЕРАТ.....	3
ABSTRACT.....	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	6
ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ЗАСТОСУВАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ ІНТЕРНЕТ-МАГАЗИНУ.....	9
1.1. ПОСТАНОВКА ЗАДАЧІ.....	9
1.2. АНАЛІЗ АНАЛОГІВ ПРОГРАМНОГО ПРОДУКТУ.....	10
1.3. ВИБІР АРХІТЕКТУРИ ВЕБ-ДОДАТКУ.....	18
1.4. ОБҐРУНТУВАННЯ ВИБОРУ ІНСТРУМЕНТАЛЬНИХ ЗАСОБІВ ТА ВИМОГИ ДО АПАРАТНОГО ЗАБЕЗПЕЧЕННЯ.....	21
Висновки до першого розділу.....	23
РОЗДІЛ 2. ПРОЕКТУВАННЯ ІНТЕРНЕТ-МАГАЗИНУ.....	24
2.1. ВИЗНАЧЕННЯ ВАРІАНТІВ ВИКОРИСТАННЯ ТА ОБ’ЄКТНО-ОРІЄНТОВАНОЇ СТРУКТУРИ СИСТЕМИ.....	24
2.2. РОЗРОБКА БАЗИ ДАНИХ СИСТЕМИ.....	26
2.3. ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ АЛГОРИТМІВ РОБОТИ СИСТЕМИ.....	37
2.4. РЕАЛІЗАЦІЯ ВЕБ-ДОДАТКУ.....	39
Висновки до другого розділу.....	45
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ДОДАТКОМ.....	46
3.1. СТРУКТУРА ІНТЕРФЕЙСУ.....	46
3.2. ТЕСТУВАННЯ.....	52
Висновки до третього розділу.....	55
ВИСНОВКИ.....	56
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	57
ДОДАТКИ.....	58

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

IDE (Integrated Development Environment) – це набір інструментів, які дозволяють створювати програми.

MVC (Model-View-Controller) – архітектурний шаблон, який використовується під час проєктування та розробки програмного забезпечення.

СУБД (Система Управління Базами Даних) – набір взаємопов'язаних даних і програм для доступу до цих даних.

SQL (Structured Query Language) – декларативна мова програмування для взаємодії користувача з базами даних.

MySQL – вільна система керування реляційними базами даних.

API (Application Programming Interface) – прикладний програмний інтерфейс, спрямований на розширення функціоналу додатку шляхом інтеграції готових рішень.

SDK (Software Development Kit) – набір інструментів для розробки програмного забезпечення.

JSON (JavaScript Object Notation) – текстовий формат, призначений для зберігання структурованих даних.

XAMPP (акронім) – 'X' (будь-яка з чотирьох операційних систем), 'A'pache, 'M'ySQL, 'P'HP, 'P'perl.

ВСТУП

В час розвитку технологій для полегшення життя люди все більше звертають увагу на онлайн-магазини як зручний спосіб зробити потрібні покупки, від побутових товарів до автомобілів та будинків. Враховуючи те, що найпопулярніші онлайн-магазини мають власні точки видачі товарів, такий досвід може перетворитися у щось на кшталт походу в торговий центр.

Актуальність теми полягає у тому, що онлайн магазини та маркетплейси є невід'ємними елементами сучасної торгівлі та покупок. Адже це справді зручно з багатьох сторін: продавець може зберігати всі товари навіть у себе вдома, а для потенційного покупця пропонуються швидкі та зручні покупки, різноманітний асортимент товарів.

Розвиток інформаційних технологій та доступність інтернету значно змінили споживацькі звички, відкривши нові можливості для бізнесу. Інтернет-магазини дозволяють підприємствам ефективніше взаємодіяти зі споживачами, надаючи їм можливість купувати товари та послуги з будь-якої точки світу у зручний для них час. Це створює величезний потенціал для збільшення продажів, розширення аудиторії та зниження витрат на утримання фізичних магазинів.

Окрім того, досвід пандемії COVID-19 ще більше підкреслює важливість онлайн-торгівлі. Через обмеження на пересування та закриття фізичних магазинів багато компаній були змушені перейти на цифрові платформи, що прискорило розвиток інтернет-магазинів. Це дозволило підприємствам не лише вижити в умовах кризи, але й зберегти та розширити клієнтську базу.

Інтернет-магазини є також важливою частиною цифрової трансформації бізнесу. Завдяки аналітичним інструментам та сучасним технологіям, таким як штучний інтелект і автоматизація, компанії можуть збирати дані про поведінку споживачів, персоналізувати пропозиції та покращувати клієнтський досвід. Це робить інтернет-магазини не просто каналом збуту, а стратегічним інструментом для розвитку бізнесу у глобальній цифровій економіці.

Метою даної роботи є розробка веб-сайту з можливістю покупки товарів, їх оцінювання та сортування задля привернення уваги потенційних користувачів до додатку як платформи в цілому.

Основні завдання визначаються сформульованою метою, а саме:

- 1) аналіз теоретичних та практичних вимог щодо розробки веб-сайту;
- 2) визначення структури та загальної будови системи;
- 3) оцінка існуючих інтернет-магазинів з обраної теми та визначення їх особливостей;
- 4) обґрунтування та вибір інструментів реалізації;
- 5) проектування архітектури та алгоритмів функціонування;
- 6) реалізація веб-сайту.

Об'єктом дослідження є додаток з використанням веб-технологій, який буде спрямований на створення вітрини кондитерських товарів, організацію процесу їх купівлі/продажу, обробку даних про користувачів та реалізацію не менш важливих деталей, зокрема, відгуків, замовлень, адміністрування та іншого.

Предметом дослідження є архітектура веб-додатку та логіка його роботи. Дослідження спрямоване на вивчення технологій та методів, що дозволяють забезпечити надійний та ефективний збут товару в рамках інтернет-магазину.

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ЗАСТОСУВАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ ІНТЕРНЕТ- МАГАЗИНУ

1.1. Постановка задачі

У межах даної бакалаврської роботи, веб-додаток проектуватиметься у вигляді сайту, створеного з метою продажу та доставкою солодощів у будь-який куточок України. Завдання полягає в створенні простого та надійного сайту, що стабільно працюватиме не лише задля збільшення обсягу продажів, а й пришвидшення та спрощення самого процесу.

Зазвичай, такий сайт містить наступні елементи:

- 1) Реєстрація нового користувача через E-Mail.
- 2) Створення кошика товарів.
- 3) Політика та умови користування.
- 4) Новини магазину.
- 5) Соціальні функції, такі як можливість залишати відгуки для товарів та спільний чат.
- 6) Інтеграція з іншими сервісами. Зв'язок з соціальними мережами. Інтеграція з сервісами оплати (PrivatBank) для оплати товарів. Підтримка зовнішніх API для розширення функціоналу.
- 7) Безпека та конфіденційність. Шифрування даних для захисту приватності користувачів. Налаштування доступу до сторінок сайту та інших функцій.

Технічні завдання реалізації додатку наступні:

- 1) Вибір технологій:
 - визначення стеку технологій для розробки веб додатку;
 - вибір фреймворку для швидкої розробки та підтримки.
- 2) Розробка бази даних:
 - проектування та розробка структури бази даних для зберігання координат, маршрутів або просто поміток;

- вибір та налаштування системи управління базами даних.
- 3) Користувацький інтерфейс:
- розробка користувацького інтерфейсу для додатку з використання CSS, HTML та Bootstrap;
 - реалізація адаптивного дизайну для оптимального відображення на різних пристроях.
- 4) Функціонал додатку:
- створення та впровадження функцій та особливостей додатку з метою забезпечення його функціональності та ефективності.
- 5) Тестування та відлагодження:
- проведення тестування всіх функцій та компонентів додатку для виявлення та виправлення помилок.
- 6) Основні цілі на даному етапі:
- пошук та аналіз аналогів;
 - пошук та збір необхідних даних для створення додатку;
 - визначення вимог до програмного забезпечення;
 - вибір інструментів для розробки;
 - розробка бази даних і налаштування зв'язків всередині неї;
 - визначення структури додатку, кількості модулів, їх ролей;
 - проектування та кодування інтернет-магазину;
 - тестування додатку.

1.2. Аналіз аналогів програмного продукту

Дослідження та порівняння аналогів є важливим кроком в розробці будь-якого продукту або ж сервісу. Також це корисно в процесі оновлення чи вдосконалення вже існуючих проектів. Дискутуючи про аналіз в цілому, слід підкреслити його необхідність для розуміння базових принципів у галузі досліджуваного продукту та можливостей для його покращення. Важливо, що разом із проведенням аналізу аналогів часто приходять і нові ідеї. Здебільшого, вони стосуються функціональних можливостей або підходів, які можна

впровадити в продукт для зростання його конкурентоспроможності, відповідності потребам користувачів і сучасного світу загалом.

“ТПрестиж” (tprestige.com) – є веб-сайтом від однойменної компанії, що надає можливість купувати оптом різноманітні види солодоців.

Серед основних функцій:

- 1) Відображення асортименту кондитерських виробів із категоріями (торти, тістечка, печиво, цукерки тощо).
- 2) Можливість швидкого пошуку продукції за назвою, складом, ціною, популярністю або іншими параметрами.
- 3) Зручна форма оформлення замовлення.
- 4) Новини інтернет-магазину.

Переваги:

- 1) Широкий асортимент товарів.
- 2) Зручна навігація по сайту.
- 3) Стильний дизайн, чудові анімації.
- 4) Можливість вибору англійської локалізації.
- 5) Достатньо високий рівень інформативності.
- 6) Зручні канали для зв'язку (чат, телефон, електронна пошта) для
- 7) вирішення питань та консультацій.

Недоліки:

- 1) Відсутня можливість реєстрації.
- 2) Відсутня корзина товарів.
- 3) Відсутня можливість покупки в один клік.
- 4) Відсутня онлайн-оплата замовлення.
- 5) Неточність даних, стосується годин роботи бізнесів, адрес або телефонних номерів.

Загальний вигляд сайту можна побачити на рис. 1.1 та 1.2.

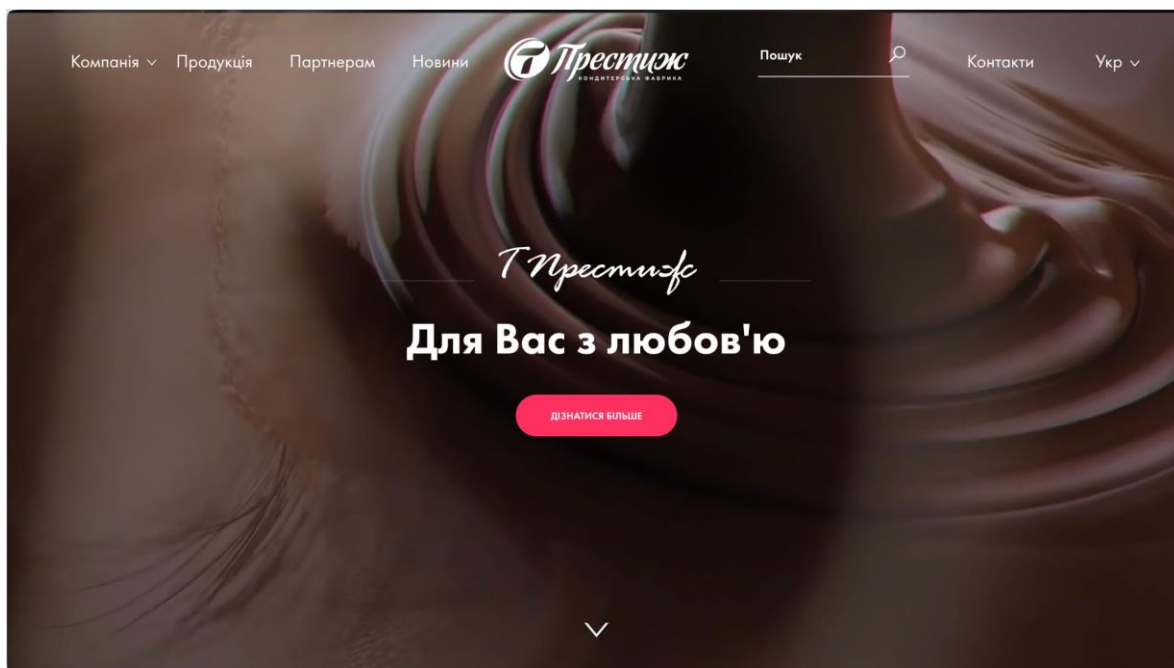


Рисунок 1.1 - Головна сторінка сайту

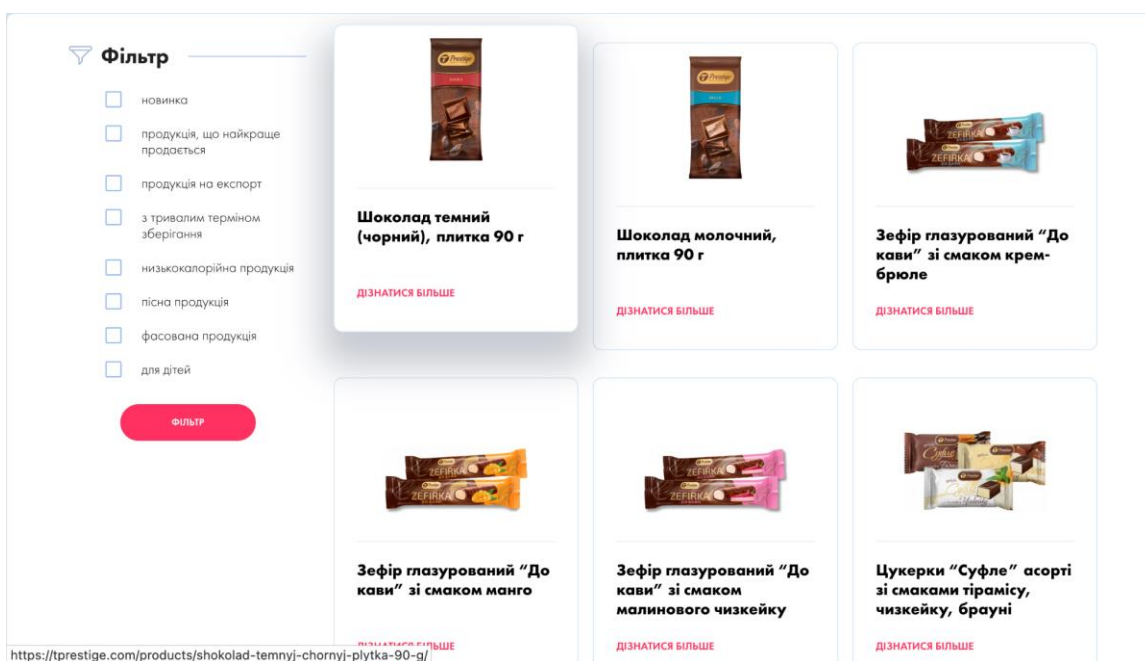


Рисунок 1.2 - Перегляд каталогу товарів

“Пригощайся” (gzpt.com.ua) – є веб-сайтом однойменної кондитерської фабрики, що надає можливість купувати різноманітні види солодошів, що виробляються на цій фабриці, а також ознайомлення зі структурою організації та пропозиціями співпраці для партнерів.

Серед основних функцій:

- 1) Відображення широкого асортименту кондитерських виробів із категоріями.
- 2) Можливість швидкого пошуку продукції за назвою, складом, ціною, популярністю або іншими параметрами.
- 3) Можливість зміни представлення каталогу товарів у вигляді списку та перегляд артикулів.
- 4) Зручне оформлення замовлення.
- 5) Пропозиція товарів на основі попередніх замовлень або переглядів.
- 6) Відгуки.
- 7) Блог інтернет-магазину.

Переваги:

- 1) Зручна навігація по сайту.
- 2) Швидка реєстрація на сайті.
- 3) Широкий асортимент товарів.
- 4) Вигідні акційні пропозиції.
- 5) Є корзина для подальшого оформлення замовлення.
- 6) Є можливість оплати онлайн.
- 7) Достатньо високий рівень інформативності.
- 8) Зручні канали для зв'язку (чат, телефон, електронна пошта) для вирішення питань та консультацій.

Недоліки:

- 1) Досить примітивний дизайн.
- 2) Відсутня англійська локалізація.
- 3) Відсутня можливість покупки в один клік.
- 4) Завелика кількість тексту в деяких розділах сайту.

Загальний вигляд сайту можна побачити на рис. 1.3 – 1.5.

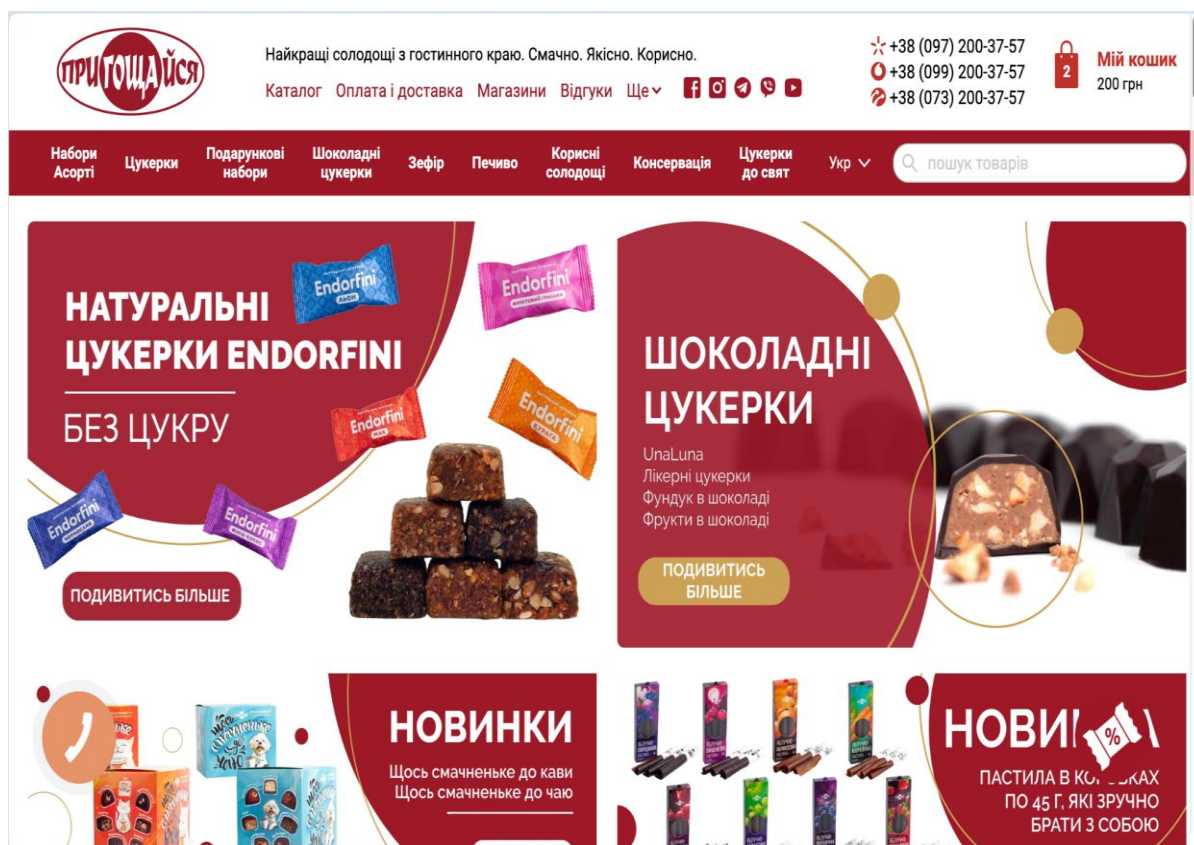


Рисунок 1.3 - Головна сторінка сайту

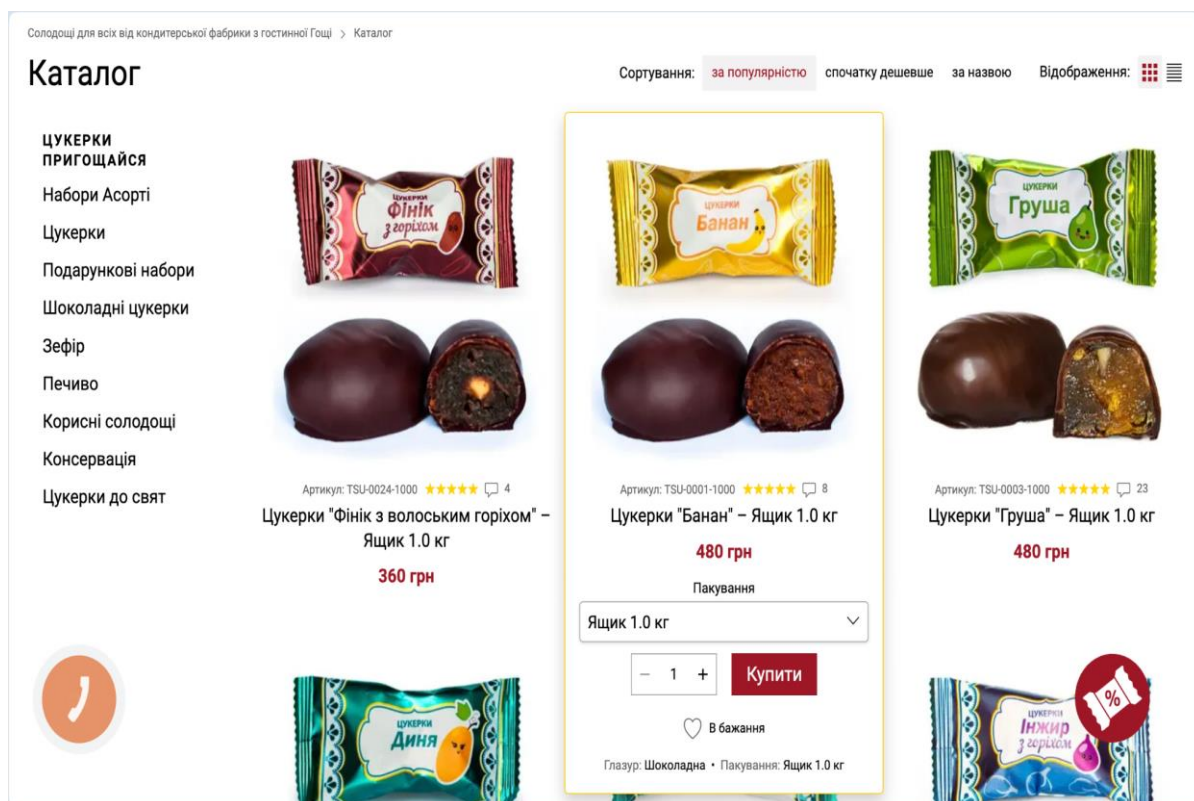


Рисунок 1.4 - Каталог товарів

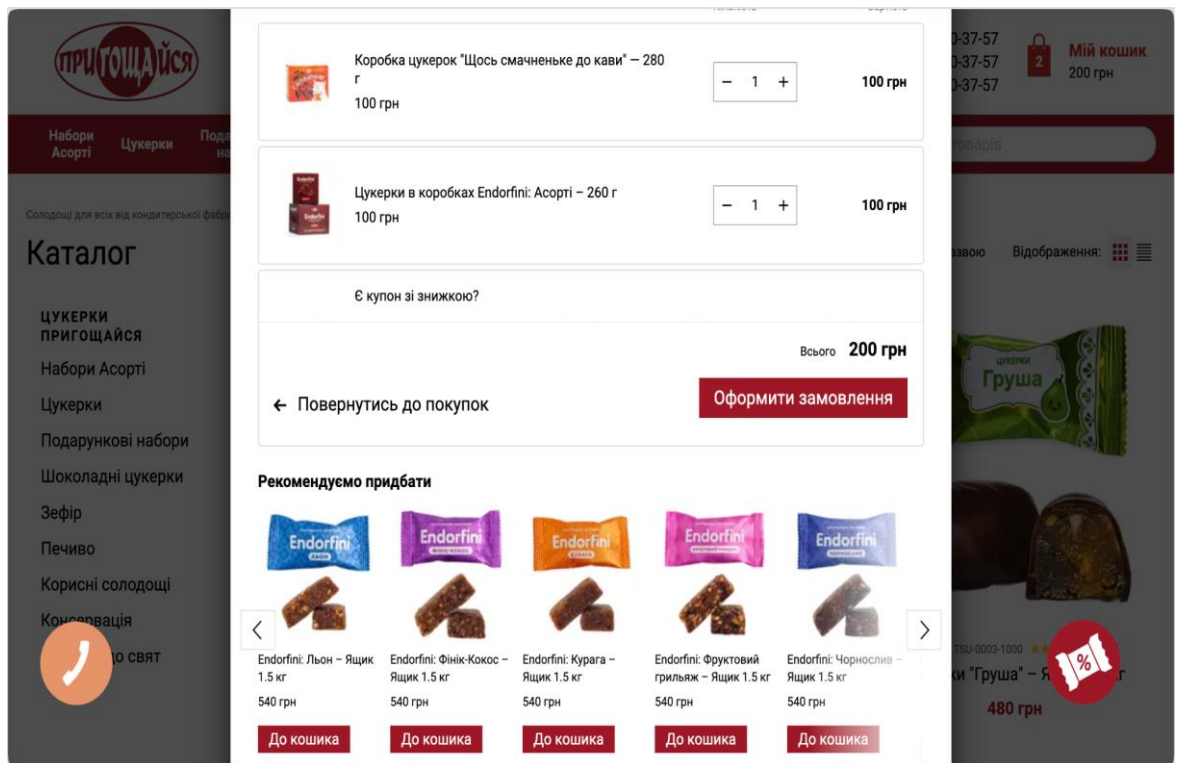


Рисунок 1.5 - Корзина користувача

“Житомирські ласощі” (zl.com.ua) – потужний веб-сайт однойменної кондитерської фабрики, що надає можливість купувати різноманітні види солодоців, що виробляються на цій фабриці, а також ознайомлення зі структурою організації, з пропозиціями співпраці для партнерів, постачальників та акціонерів.

Серед основних функцій:

- 1) Відображення широкого асортименту кондитерських виробів із категоріями.
- 2) Можливість швидкого пошуку продукції за назвою, складом, ціною, популярністю або іншими параметрами.
- 3) Можливість зміни представлення каталогу товарів у вигляді списку та перегляд артикулів.
- 4) Можливість швидкого перегляду деталей про товар.
- 5) Зручне оформлення замовлення.
- 6) Пропозиція товарів на основі попередніх замовлень або переглядів.
- 7) Новини інтернет-магазину.

Переваги:

- 1) Простий та красивий дизайн.
- 2) Зручна навігація по сайту.
- 3) Швидка реєстрація на сайті.
- 4) Широкий асортимент товарів.
- 5) Розділ “Акції”.
- 6) Є корзина для подальшого оформлення замовлення.
- 7) Є можливість оплати онлайн.
- 8) Достатньо високий рівень інформативності.
- 9) Зручні канали для зв'язку (чат, телефон, електронна пошта) для вирішення питань та консультацій.
- 10) Англійська локалізація.
- 11) Можливість покупки в один клік.

Недоліки:

- 1) Низька якість окремих зображень в деяких розділах сайту.

Загальний вид додатку можна побачити на рис 1.6 - 1.8.

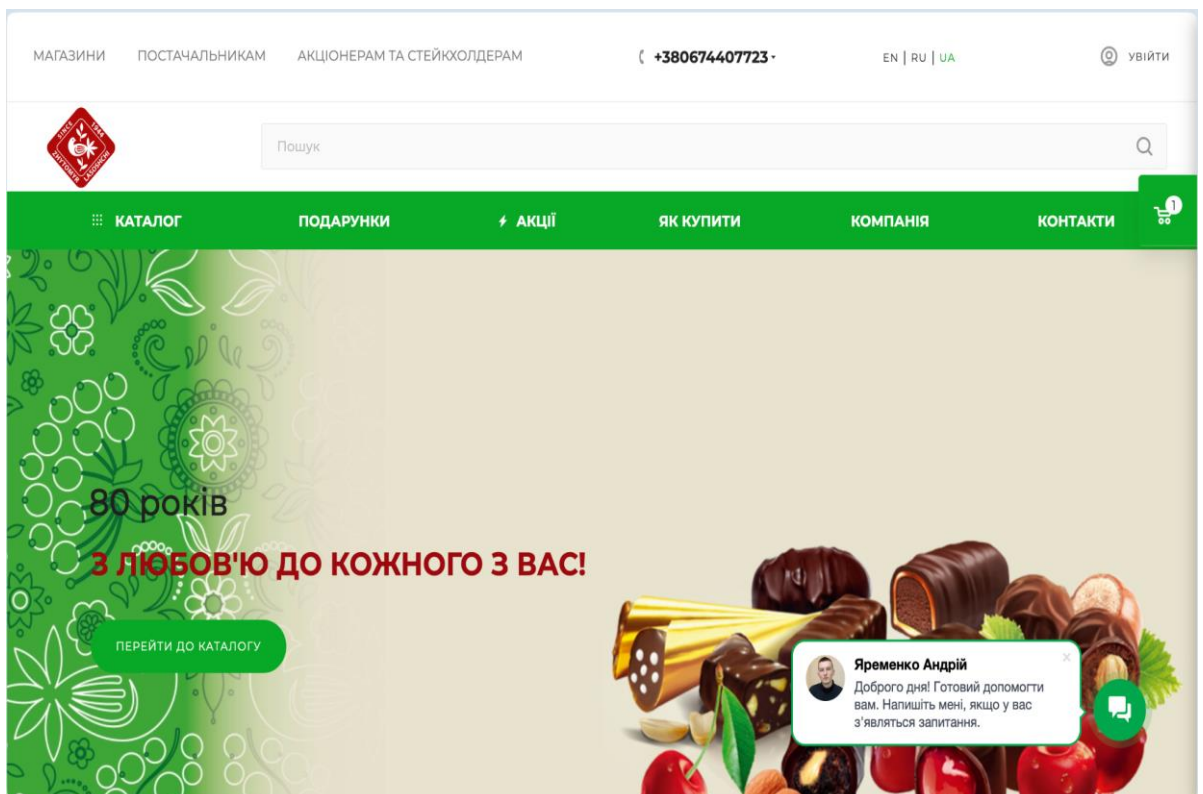


Рисунок 1.6 - Головна сторінка сайту

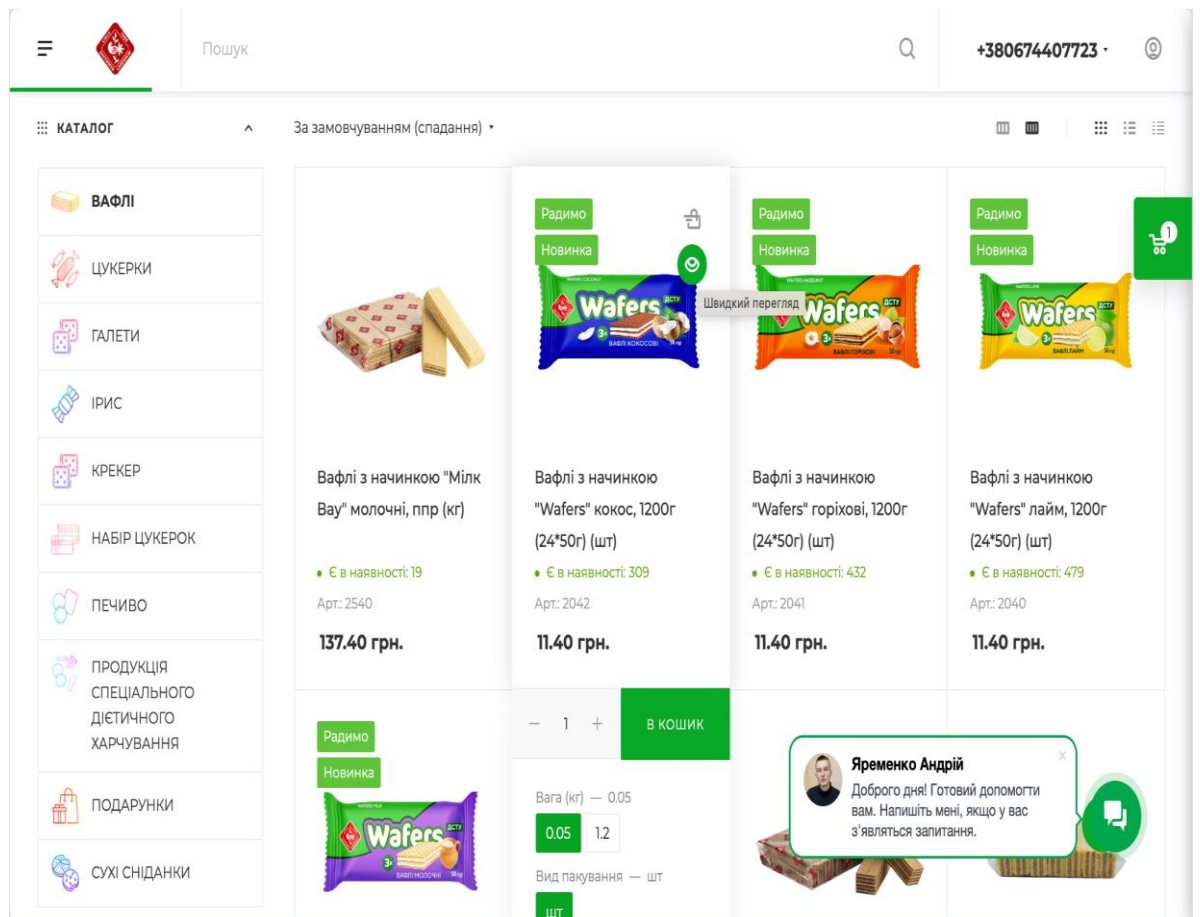


Рисунок 1.7 - Каталог товарів

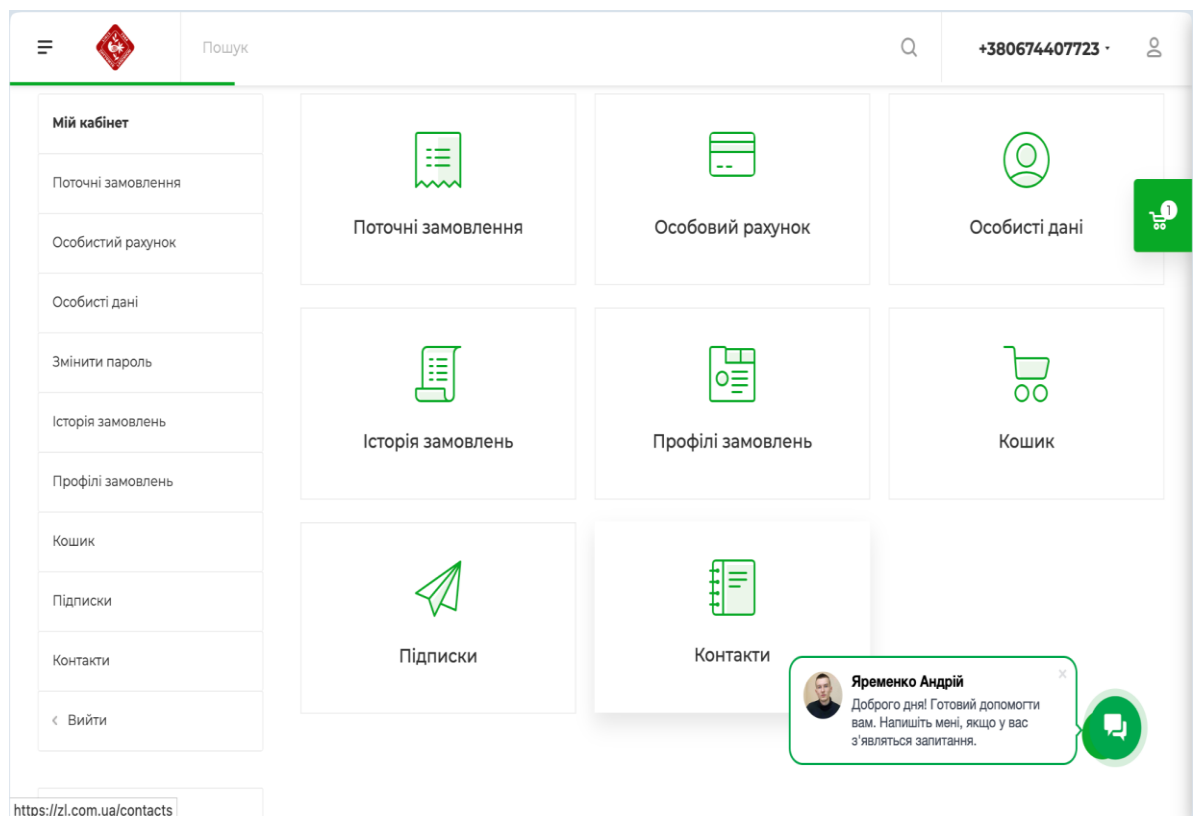


Рисунок 1.8 - Особистий кабінет користувача

Виходячи з наявних переваг та недоліків, складемо порівняльну таблицю вищезгаданих аналогів (табл. 1.1).

Таблиця 1.1

Порівняння функціоналу інтернет-магазинів

Характеристика	ТПрестиж	Пригощайся	Житомирські ласощі
Особистий кабінет	—	+	+
Корзина товарів	—	+	+
Фільтрація каталогу та пошук	+	+	+
Блог магазину	+	+	+
Онлайн оплата	—	+	+
Чат користувачів	—	—	—
Відгуки	—	+	+

Всі представленні інтернет-магазини мають свої особливості та переваги. “Пригощайся” та “Житомирські ласощі” чудово підходять людям, які просто люблять та купують солодощі, чи обирають їх у якості подарунку. На відміну від попередніх двох прикладів, “ТПрестиж” призначений, скоріше, для вирішення питань оптової торгівлі та партнерських замовлень.

Є ще багато інших цікавих аналогів нашого майбутнього інтернет-магазину. Але для виявлення нових ідей, функціональних можливостей або підходів, які можна впровадити для збільшення конкурентоспроможності та відповідності потребам наразі цілком достатньо проаналізованих щойно аналогів.

1.3. Вибір архітектури веб-додатку.

Проектування будь-якого програмного забезпечення, незалежно від типу та специфікації, завжди починається з побудови базової архітектури. Архітектурних рішень в області веб-розробки існує дуже багато. Ось неповний список “лідерів” [4]:

- 1) Client-Server Architecture.
- 2) Layered (багатошарова архітектура).

- 3) Microservices (мікросервісна архітектура).
- 4) Peer-to-Peer (P2P).
- 5) MVC (Model-View-Controller).
- 6) MVVM (Model-View-ViewModel).
- 7) MVP (Model-View-Presenter).

MVC, MVVM та MVP добре підходять розробнику, коли необхідно створити веб-додаток зі складною структурою, де важливі чітке розділення бізнес-логіки, інтерфейсу та обробки запитів. Це класичний вибір для проектів, які потребують легкого масштабування, модульності та зручної підтримки. Розглянемо їх ключові характеристики, щоб визначити найбільш зручний і доцільний (табл. 1.2).

Таблиця 1.2

Порівняння архітектурних патернів

Особливості	MVC (Model-View-Controller)	MVVM (Model-View-ViewModel)	MVP (Model-View-Presenter)
Призначення	Класична архітектура для веб-додатків зі складною логікою; чітко розділяє представлення, логіку і управління запитами.	Використовується в більшості десктопних або мобільних додатків (наприклад, Xamarin, WPF), де вкрай важливим є механізм двостороннього зв'язку між View і ViewModel.	Призначений для чіткого розділення інтерфейсу користувача та бізнес-логіки; широко використовується для мобільних додатків.
Компоненти	Model – логіка і дані; View – інтерфейс; Controller – управління запитами.	Model – логіка; View – інтерфейс; ViewModel – пов'язує View і Model, а також підтримує двосторонній зв'язок з View.	Model – дані; View – інтерфейс; Presenter – логіка, яка керує взаємодією між View і Model.
Взаємодія	View пасивно отримує дані через Controller. Користувач взаємодіє з Controller, що оновлює Model і View.	View взаємодіє з ViewModel через двосторонній зв'язок, що оновлює UI автоматично при зміні стану Model.	View передає дані Presenter, який обробляє їх і оновлює Model і View; Presenter є посередником між View та Model.

Двосторонній зв'язок	Немає вбудованої підтримки двостороннього зв'язку; потрібна ручна синхронізація між View і Model.	Є вбудований двосторонній зв'язок між View та ViewModel (зазвичай через data-binding).	Немає двостороннього зв'язку; Presenter активно передає дані з Model у View і навпаки.
Тестування	Легке тестування Model і Controller; View важче тестувати через прив'язку до UI.	Дуже зручне тестування завдяки ViewModel, який ізольований від UI і може бути протестований окремо.	Presenter легко тестується, оскільки він відокремлений від View і безпосередньо не взаємодіє з UI.
Залежності	Controller часто містить код як для логіки, так і для обробки запитів, що може призводити до змішування відповідальностей.	ViewModel має чітку залежність від View і Model; ця структура спрощує двосторонній зв'язок, але може ускладнювати саму ViewModel.	Presenter має чітко визначену роль посередника, але може стати надмірно громіздким, якщо додати забагато логіки.

У підсумку до завершеного порівняльного аналізу відомих класичних архітектурних підходів вдалося визначити їх особливості. Досить добре помітно не лише спільні риси, а й відмінності патернів. Враховуючи всі їх переваги та недоліки, виявлені в ході аналізу, а також, спираючись на власний досвід використання архітектурних патернів у веб-розробці, було обрано саме MVC в якості найбільш зручного та доцільного серед решти інших.

MVC суттєво знизить вірогідність виникнення критичних помилок при роботі з великим обсягом даних, складною бізнес-логікою та інтеграцією різних сервісів, забезпечуючи стабільну і гнучку архітектуру. При цьому, працюватимуть зрозумілі та прості механізми взаємодії компонентів додатку (рис. 1.9).

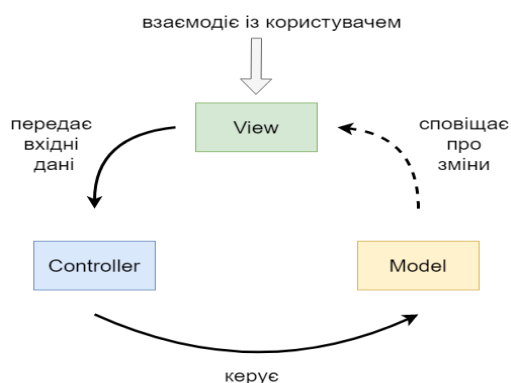


Рисунок 1.9 - Схема роботи MVC

1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Для створення сайту з архітектурою MVC вибір мови програмування залежить від декількох факторів: вимог проекту, наявного досвіду розробника, підтримуваних технологій, та екосистеми фреймворків. На даному етапі вже чітко визначено всі попередні умови. Отже, далі коротко проаналізуємо найпопулярніші інструменти розробника, котрі за довгий час перетворилися з універсальних мов програмування на стандарти веб-розробки.

Python:

Фреймворк: Django, Flask.

Переваги: мова проста у використанні та має великий набір бібліотек. Django надає зручну інтеграцію для реалізації MVC (хоча в Django офіційно використовується патерн MTV - Model-Template-View, він близький до MVC). Підтримка ORM спрощує роботу з базами даних.

Недоліки: Висока популярність для невеликих та середніх проектів, але може бути менш ефективною для масштабних систем порівняно з іншими рішеннями.

JavaScript (Node.js):

Фреймворк: Express, NestJS

Переваги: Node.js дозволяє використовувати JavaScript як на клієнтській, так і на серверній стороні. Express — це легкий фреймворк для побудови MVC-додатків, а NestJS — структурованіший і ближчий до традиційного MVC. JavaScript має великий вибір бібліотек і модулів.

Недоліки: Асинхронність може викликати труднощі в управлінні кодом, якщо проект дуже великий.

PHP [1]:

Фреймворк: Laravel, Symfony

Переваги: PHP давно використовується для веб-розробки і має багато фреймворків для MVC. Laravel дуже популярний завдяки своїй простоті й

зручним інструментам для побудови сайтів та API. Symfony підходить для більш складних проектів з вимогою до масштабованості.

Недоліки: PHP критикується за продуктивність у порівнянні з іншими мовами, хоча сучасні фреймворки значно підвищили ефективність.

C#:

Фреймворк: ASP.NET Core

Переваги: C# разом з ASP.NET Core є чудовим вибором для веб-додатків з архітектурою MVC, особливо якщо додаток працює на Windows-серверах або інтегрується з продуктами Microsoft. ASP.NET Core надає багато інструментів для високопродуктивних, масштабованих веб-додатків.

Недоліки: Більш складний початок для розробників без досвіду з технологіями Microsoft.

Беручи до уваги вимоги до підтримуваних технологій проекту і екосистеми фреймворків, можна стверджувати, що чудовим вибором є мова PHP. Вона добре підходить для реалізації веб-сайту з використанням MVC паттернів, фреймворків або будь-яких зовнішніх механізмів (API). А з урахуванням мого наявного досвіду, PHP слугуватиме надійним інструментом для вирішення поставлених задач.

PHP спеціально створена для розробки веб-застосунків. Вона легко інтегрується з HTML і забезпечує широкі можливості для взаємодії з базами даних. Використовується також для обробки запитів на стороні сервера, що дозволяє динамічно генерувати вміст веб-сторінок.

Вибір спеціалізованих інструментів веб-розробки (IDE) теж є дуже важливим фактором. Існує широкий асортимент таких інструментів. Однак, зупинятися на їх порівнянні не будемо, оскільки свою перевагу я давно віддаю наступним рішенням:

PhpStorm — це IDE від компанії JetBrains, орієнтований на веб-додатки та інші види програм, які можна створити за допомогою мови PHP з використанням HTML, JavaScript і CSS. Рішення PhpStorm розгортає та синхронізує проекти через FTP. Середовище PhpStorm пропонує функції автоматичного завершення

мовних конструкцій PHP в коді, перевірки коду, різних алгоритмів рефакторингу та швидкої навігації по коду.

DataGrip — це IDE від компанії JetBrains для роботи з базами даних MySQL, PostgreSQL, Oracle, SQL Server, Sybase, DB2, SQLite, HyperSQL, Apache Derby та H2. Рішення JetBrains DataGrip включає потужний текстовий редактор з мультиструмперами, забезпечує синтаксичне підсвічування коду, підтримує інтеграцію з системами контролю версій Git, Subversion тощо.

XAMPP — безкоштовна кросплатформенна збірка веб-сервера з відкритим початковим кодом, що містить HTTP-сервер Apache, базу даних MariaDB, MySQL й інтерпретатори скриптів для мов програмування PHP та Perl, а також додаткові бібліотеки, що дозволяють запустити повноцінний веб-сервер.

Висновки до першого розділу

Перед проектуванням сайту проведено аналіз існуючих аналогів для отримання ідей та вдосконалення проекту. В процесі дослідження аналогічних і схожих за концепцією продуктів були проаналізовані різні платформи з різними принципами роботи. Не змінилась головна мета проекту – створити простий і зручний інтернет-магазин. Основні його функції мають включати навігацію, перегляд та замовлення товарів, чат та перегляд блогу.

Крім цього, були сформовані основні вимоги для власної платформи. На основі відповідних вимог та критеріїв був сформований набір основних технологій і засобів реалізації для розробки веб-додатку. Було обрано напрямок створення БД на основі СУБД MySQL з використанням phpMyAdmin та DataGrip, а також додатку, написаного мовою PHP з використанням платформи PhpStorm як одного з найкращих IDE для створення сайтів на базі розмітки PHP HTML; обрано патерн проектування MVC, а також бібліотеку стилів Bootstrap.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ІНТЕРНЕТ-МАГАЗИНУ

2.1. Визначення варіантів використання та об'єктно-орієнтованої структури системи

Опис основних вимог до програмного продукту – важливий етап проектування. В нього входять визначення та деталізація функціональних і нефункціональних вимог до програмного забезпечення, оцінка часу реакції системи на дії користувачів, специфікація можливих сценаріїв використання, а також розробка загальної схеми компонентів системи з описом їхньої структури, методів взаємодії. Цей процес дозволяє чітко зрозуміти потреби користувачів та технічні характеристики, необхідні для створення ефективного та зручного веб-додатку.

Функціональні вимоги:

- 1) Відсутність потреби в автентифікації.
- 2) Пошук та фільтрація в розділі “Товари”.
- 3) Можливість додавання товарів у кошик.
- 4) Можливість залишити відгук про товар.
- 5) Можливість спілкуватись в “Чаті” з іншими користувачами.
- 6) Можливість перегляду історії замовлень.
- 7) Можливість перегляду кількості товарів у замовленні.
- 8) Можливість зміни даних користувача.

Нефункціональні вимоги:

- 1) Мова інтерфейсу користувача має бути українською.
- 2) Авторизація та автентифікація повинні виконуватися за допомогою системи, яка забезпечує надійне зберігання клієнтських даних і паролів, із використанням шифрування та захисту від несанкціонованого доступу.
- 3) Продуктивність системи повинна забезпечувати час відгуку на дії користувача не більше 3 секунд у 95% випадків.

- 4) Надійність система повинна задовольняти час безвідмовної роботи не менше 99.9% протягом місяця.
- 5) Безпека має полягати в реалізації засобів захисту від атак типу SQL-ін'єкцій, XSS (міжсайтових скриптів) та CSRF (підробки міжсайтових запитів).
- 6) Рівень масштабованості системи повинен бути достатнім для обслуговування збільшеної кількості користувачів без зниження продуктивності.
- 7) Доступність системи має задовольняти користувачів з обмеженими можливостями, відповідно до стандартів WCAG 2.1.

Діаграма варіантів використання (Use Case Diagram) — це тип діаграми в UML (Unified Modeling Language), яка показує взаємодію між користувачами (акторами) та системою для виконання конкретних функцій (варіантів використання). Вона допомагає описати функціональні вимоги до системи і зобразити, як різні користувачі взаємодіють з нею. На рис. 2.1 наведено діаграму варіантів використання інтернет-магазину.

Нижче, в табл. 2.1 знаходиться перелік та опис основних варіантів використання.

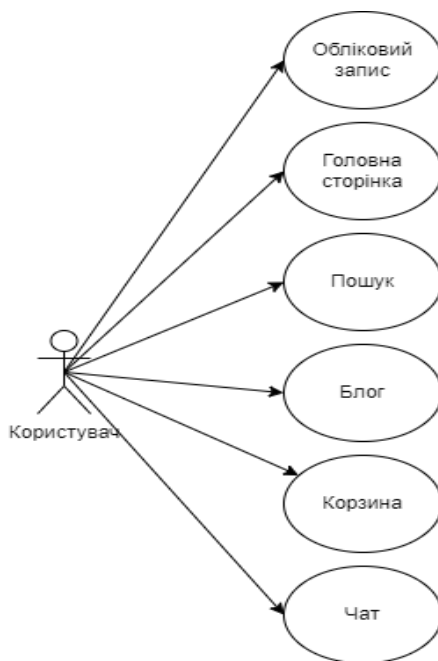


Рисунок 2.1 - Варіанти використання інтернет-магазину

Таблиця 2.1

Найменування	Опис
Авторизація	Користувач може зареєструватися або залогінитись на сайті.
Пошук	Користувач може знайти бажаний товар за коротким словом введеним в полі пошуку.
Блог	Користувач може переглянути останні новини пов'язані з сайтом.
Кондитерські вироби	Користувач може ознайомитись з переліком всіх товарів, наявних у магазині.
Чат	Користувач може перейти в глобальний чат, де всі можуть спілкуватись.
Обліковий запис	Користувач може переглянути свій обліковий запис, і відредагувати свої дані.
Замовлення	Після створення та оплати замовлення, користувач має можливість переглянути всі свої замовлення та переглянути що замовляв користувач, а також може відслідковувати статуси замовлення.
Корзина	Після того, як користувач додав якісь товари у корзину, користувач може відредагувати кількість обраних товарів у корзині, і перейти до створення замовлення.

2.2. Розробка бази даних системи

Після обґрунтування всіх ідей у визначенні функціоналу та потреби задачі, було створено алгоритм роботи додатку. При запуску будуть отримуватись дані, які далі оброблятимуться моделями та повертатимуться у вигляді представлень.

На початку користувача зустрічає головна сторінка, яка пропонує користувачу перейти в каталог товарів, подивитись новини, знайти товар за

приблизним ім'ям або за категорією та зареєструватись або увійти в обліковий запис. Схема навігації наведена на рис. 2.2.

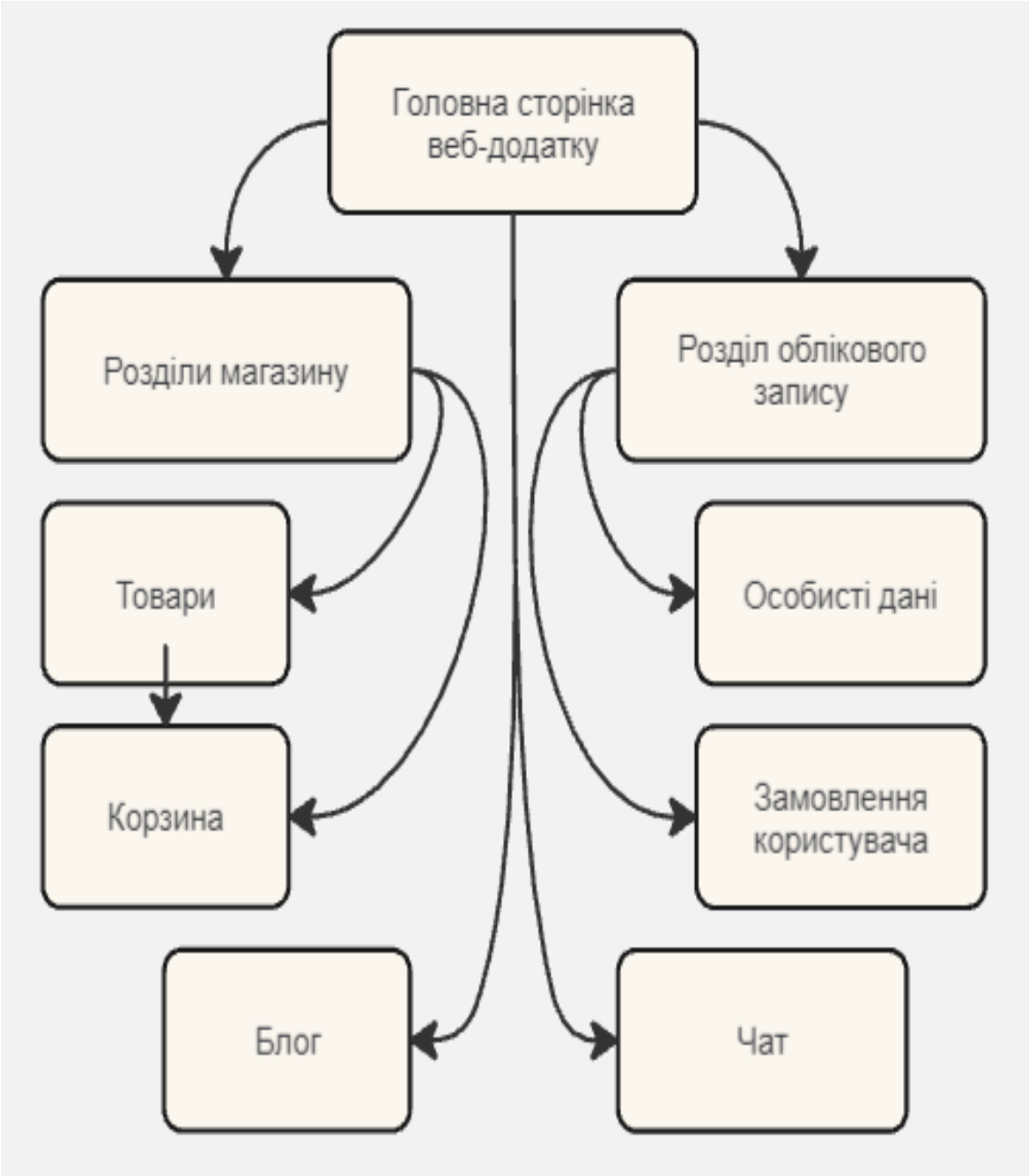


Рисунок 2.2 - Схема навігації по додатку

Після авторизації користувач зможе замовляти товари, сплачувати свої замовлення, переглядати та редагувати дані облікового запису і писати повідомлення в загальний чат.

Все вище описане буде повертатися у вигляді змінних, які будуть перевірятись і оброблятись відповідними методами ядра системи. Зрештою, оброблені дані записуватимуться в БД. Детальна схема роботи цього алгоритму зображена на рис. 2.3 та 2.4.

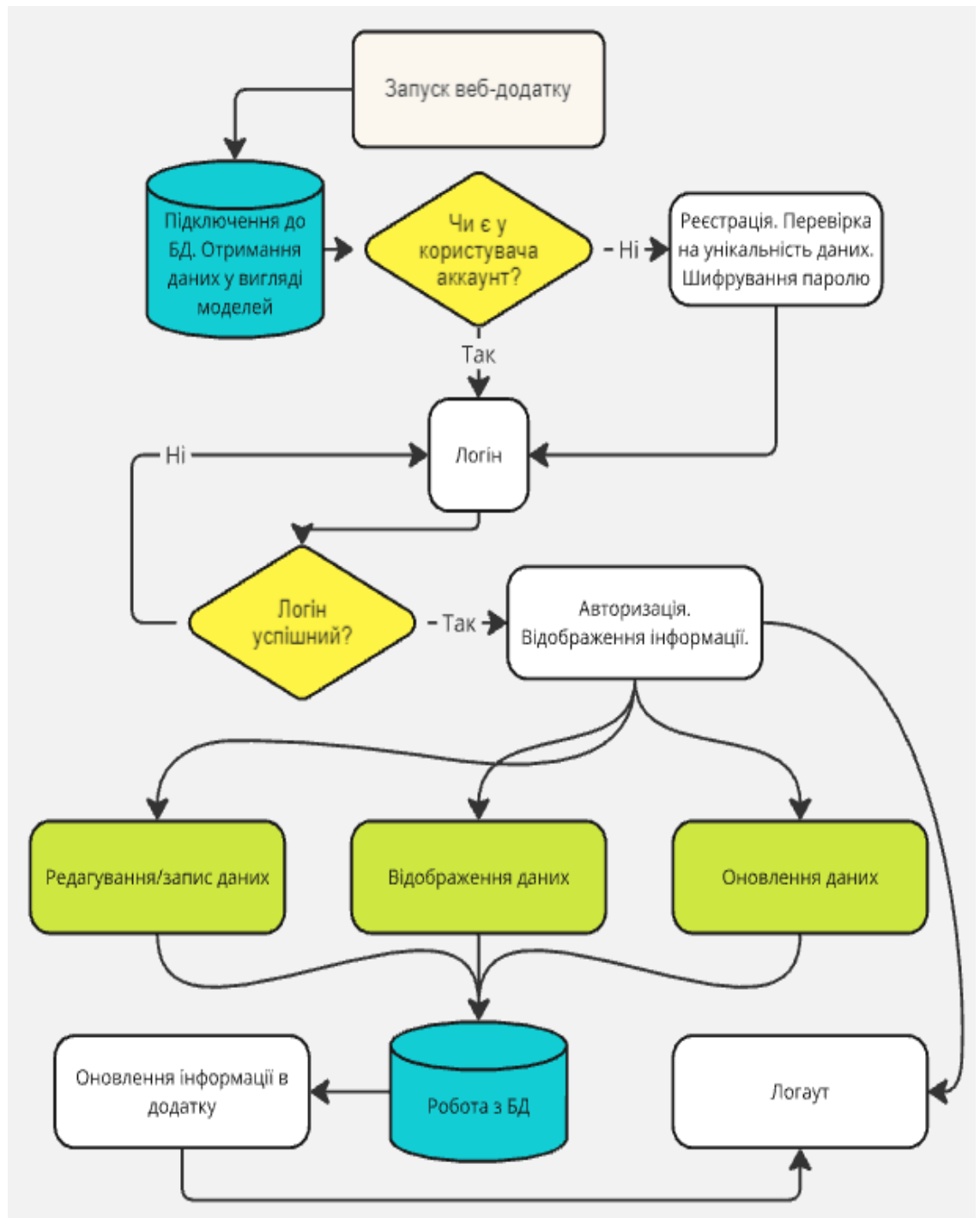


Рисунок 2.3 - Алгоритм роботи клієнтського додатку

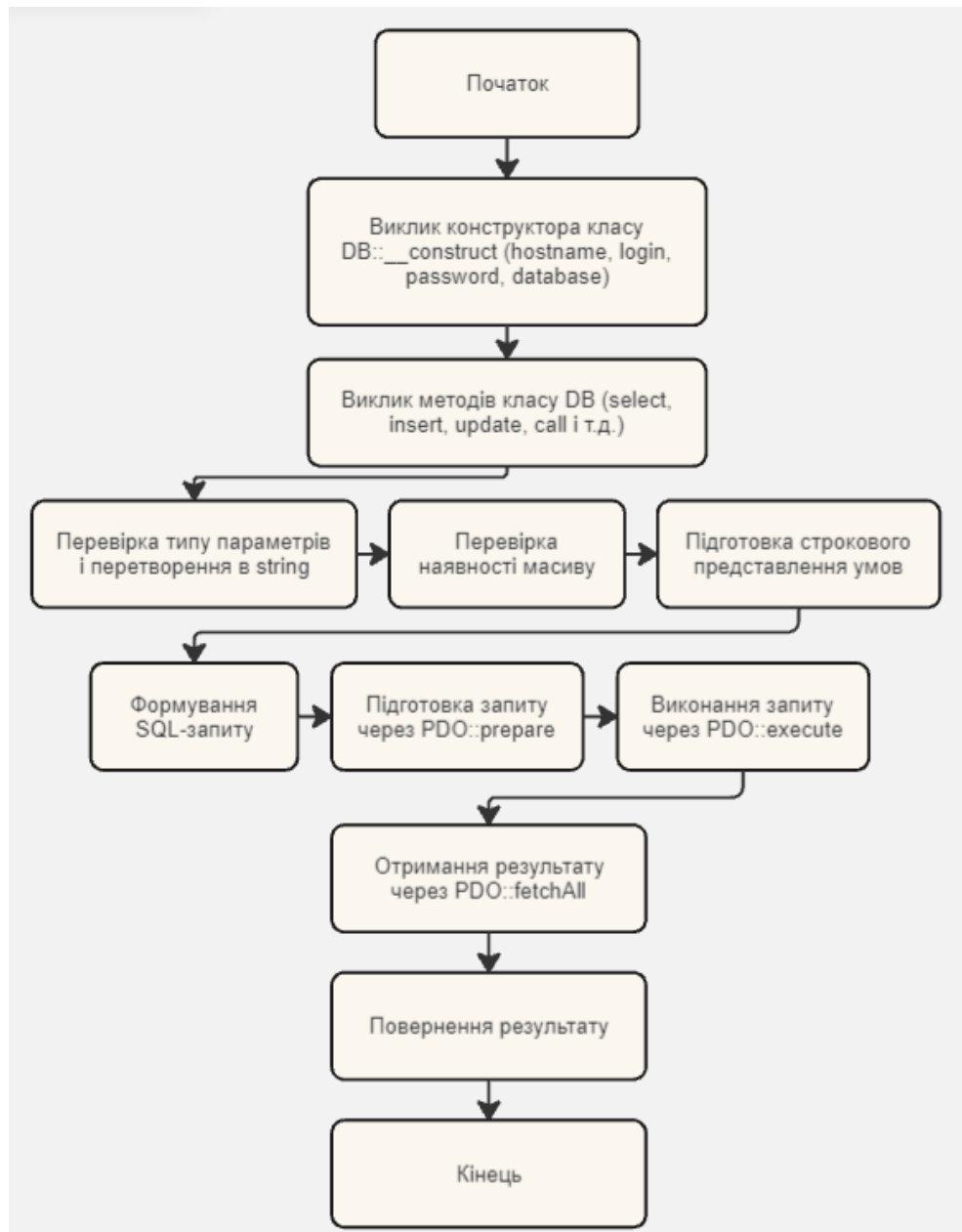


Рисунок 2.4 - Алгоритм взаємодії з БД методами Core/DB.php

Інструменти для розробки основних об'єктів і зв'язків БД повною мірою надають DataGrip та phpMyAdmin. Якщо розробнику потрібно створити або змінити таблицю, додати або змінити рядок, то графічний інтерфейс стане в нагоді. Такі зміни супроводжуються генерацією відповідного скрипта: можна без використання SQL-кодингу вносити зміни в БД, користуючись можливостями phpMyAdmin, або створити SQL-запит в IDE, тобто працювати безпосередньо з кодом та виконувати його й, таким чином, крок за кроком створювати та вдосконалювати БД.

Оскільки наша задача полягає в створенні максимально зручного та безпечного додатку, обсяг даних і кількість таблиць, необхідних для збереження даних потрібно мінімізувати. В таблицях 2.2 – 2.10 наведено назви та короткий опис полів створених SQL-таблиць.

Таблиця 2.2

users – містить облікові дані користувачів

Назва	Тип	Опис
id	int(11)	Унікальний ідентифікатор
login	varchar(255)	Е-mail користувача
password	varchar(255)	Хешований пароль користувача
access	int(11)	Рівень доступу
lastname	varchar(255)	Прізвище користувача
firstname	varchar(255)	Ім'я користувача
address	text	Адреса користувач

Таблиця 2.3

products – містить дані про товари

Назва	Тип	Опис
id	int(11)	Унікальний ідентифікатор
title	varchar(255)	Назва товару
description	text	Опис товару
price	int(11)	Ціна товару
datetime	datetime	Дата створення товару
datetime_lastedit	datetime	Дата останнього редагування товару
photo	varchar(255)	Фотографія товару

user_id	int(11)	Ідентифікатор користувача, що створив товар
category_id	int(11)	Ідентифікатор категорії

Таблиця 2.4

categories – містить дані про категорії товарів

Назва	Тип	Опис
id	int(11)	Унікальний ідентифікатор
name	varchar(50)	Назва категорії

Таблиця 2.5

chat – містить дані про повідомлення з глобального чату

Назва	Тип	Опис
id	int(11)	Унікальний ідентифікатор
message	text	Текст повідомлення
datetime	datetime	Дата створення повідомлення
user_id	int(11)	Ідентифікатор відправника повідомлення

Таблиця 2.6

comments – містить дані коментарів (відгуків) про товари

Назва	Тип	Опис
id	int(11)	Унікальний ідентифікатор

message	text	Текст коментаря
datetime	datetime	Дата створення коментаря
post_id	int(11)	Ідентифікатор товару під яким залишено коментар
user_id	int(11)	Ідентифікатор відправника коментаря

Таблиця 2.7

orders – містить дані про замовлення користувачами переліку товарів із корзини

Назва	Тип	Опис
id	int(11)	Унікальний ідентифікатор
user_id	int(11)	Ідентифікатор користувача який створив замовлення
status	varchar(255)	Статус замовлення
service	varchar(255)	Назва доставки
phone	varchar(255)	Номер телефону замовника
date	datetime	Дата створення замовлення
address	varchar(255)	Адреса куди потрібно доставити
unique_id	varchar(255)	Унікальний ідентифікатор для серверу

Таблиця 2.8

carts – містить дані про товари, додані в корзину для подальшого замовлення

Назва	Тип	Опис
user_id	int(11)	Ідентифікатор власника корзини
product_id	int(11)	Ідентифікатор товару, що додали в корзину
quantity	int(11)	Кількість доданого в корзину товару

Таблиця 2.9

purchased_carts – містить дані про товари, замовлення яких успішно сплачено

Назва	Тип	Опис
order_id	int(11)	Ідентифікатор замовлення
user_id	int(11)	Ідентифікатор користувача
product_id	int(11)	Ідентифікатор товару
quantity	int(11)	Кількість замовленого товару

Таблиця 2.10

blog – містить дані про публікації в розділі “Блог”

Назва	Тип	Опис
id	UUID	Унікальний ідентифікатор
created_at	timestamp	Дата створення новини

name	varchar(255)	Назва новини
photo	blob	Фотографія
text	varchar(1000)	Текст новини

Для остаточного завершення розробки прототипу БД, необхідно створити деякі процедури.

deleteUser() – процедура динамічного видалення користувача, що в процесі видалятиме спочатку будь-які пов’язані дані:

```
CREATE PROCEDURE deleteUser(IN userId INT)
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE tbl_name VARCHAR(255);
    DECLARE col_name VARCHAR(255);
    DECLARE cur CURSOR FOR
        SELECT
            TABLE_NAME, COLUMN_NAME
        FROM
            information_schema.KEY_COLUMN_USAGE
        WHERE
            REFERENCED_TABLE_NAME = 'users'
            AND REFERENCED_COLUMN_NAME = 'id';

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    DELETE comm
    FROM comments comm
        JOIN products prod ON comm.post_id = prod.id
    WHERE prod.user_id = userId;

    OPEN cur;

    read_loop: LOOP
        FETCH cur INTO tbl_name, col_name;
        IF done THEN
            LEAVE read_loop;
        END IF;

        SET @del_stmt = CONCAT('DELETE FROM `', tbl_name, '` WHERE `',
col_name, '` = ', userId);
        PREPARE stmt FROM @del_stmt;
        EXECUTE stmt;
        DEALLOCATE PREPARE stmt;
    END LOOP;

    CLOSE cur;

    DELETE FROM users WHERE `id` = userId;
END
```

deleteProduct() – процедура видалення товару, що видаляє лише ті товари, котрі відсутні серед переліку замовлень сплачених користувачем, але ще не оброблені адміністратором. Для зручності повертає результат своєї роботи у вигляді змінної isDeleted = (true/false):

```
CREATE PROCEDURE deleteProduct(IN productId INT, OUT isDeleted BOOLEAN)
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE tbl_name VARCHAR(255);
    DECLARE col_name VARCHAR(255);

    DECLARE cur CURSOR FOR
        SELECT
            TABLE_NAME, COLUMN_NAME
        FROM
            information_schema.KEY_COLUMN_USAGE
        WHERE
            REFERENCED_TABLE_NAME = 'products'
            AND REFERENCED_COLUMN_NAME = 'id';

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
    IF EXISTS (
        SELECT 1
        FROM purchased_carts
        WHERE product_id = productId
    ) THEN
        SET isDeleted = false;
    ELSE
        OPEN cur;

        read_loop: LOOP
            FETCH cur INTO tbl_name, col_name;
            IF done THEN
                LEAVE read_loop;
            END IF;

            SET @del_stmt = CONCAT('DELETE FROM `', tbl_name, '` WHERE `',
col_name, '` = ', productId);
            PREPARE stmt FROM @del_stmt;
            EXECUTE stmt;
            DEALLOCATE PREPARE stmt;
        END LOOP;

        CLOSE cur;

        DELETE FROM products WHERE `id` = productId;

        SET isDeleted = true;
    END IF;
END
```

createOrder() – процедура створення замовлення на основі користувацького запиту з даними про контакти, адресу, спосіб доставки та вміст корзини, що замовляється:

```
CREATE PROCEDURE createOrder(
    IN p_user_id INT,
    IN p_service VARCHAR(255),
    IN p_address VARCHAR(255),
    IN p_phone VARCHAR(255),
    IN p_unique_id VARCHAR(255)
)
BEGIN
    INSERT INTO orders (user_id, status, service, phone, address,
        unique_id)
    VALUES (p_user_id, 'Сформовано', p_service, p_phone, p_address,
        p_unique_id);
END
```

Згодом, в процесі розробки контролерів та моделей сайту, можуть знадобитись доопрацювання функціоналу БД, зокрема додавання критично важливих тригерів і подій, звичайне розширення таблиць, зв'язків тощо.

2.3. Проектування та реалізація алгоритмів роботи системи

Архітектура MVC є сучасним підходом до розробки програмного забезпечення, який широко використовується в розробці веб-додатків. В цій архітектурі кожен компонент має чітко визначені ролі, що сприяє кращому розподілу і спрощує тестування та підтримку коду.

Схему архітектури наведено на рис. 2.6.

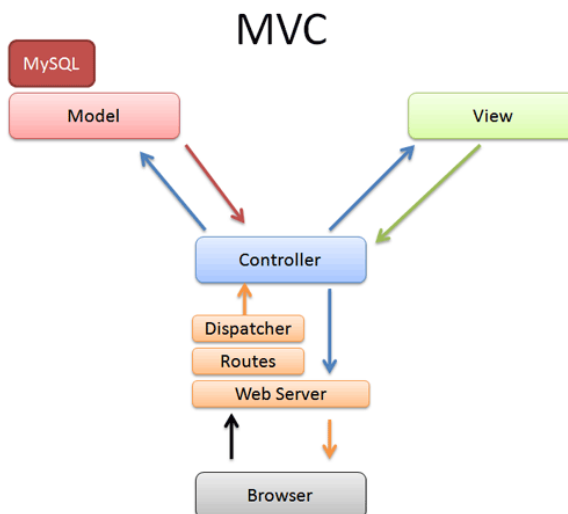


Рисунок 2.6 - Візуальна репрезентація архітектури MVC

Компонент Model відповідає за управління даними та бізнес-логікою додатку. Він містить структури даних і методи для доступу до цих даних, такі як запити до бази даних або виклики API. Model є незалежним від інших компонентів і не взаємодіє безпосередньо з користувацьким інтерфейсом, що забезпечує чітке розмежування даних та їхнього представлення. Модель також відповідає за валідацію даних і забезпечує їх правильність перед передачею в інші частини системи.

Компонент View відповідає за відображення даних користувачу. Він генерує інтерфейс, через який користувач може взаємодіяти з додатком. Представлення отримує інформацію від Моделі через Контролер і показує її у вигляді, зручному для користувача. View не містить логіки обробки або зміни даних, а лише відповідає за їхню візуалізацію та представлення в певному форматі, як-от HTML-сторінки, форми або графіки.

Компонент Controller відповідає за взаємодію між користувачем, Моделлю та Представленням. Контролер приймає вхідні запити від користувача, обробляє їх і визначає, які дії потрібно виконати. Він звертається до Моделі для отримання або зміни даних, а потім передає ці дані у Представлення для відображення. Контролер виконує роль координатора, забезпечуючи правильний потік інформації між Моделлю та Представленням, а також контролюючи взаємодію користувача з додатком.

Архітектура MVC сприяє створенню структурованого та підтримуваного коду, забезпечуючи чітке розділення обов'язків між трьома основними компонентами: Моделлю, Представленням та Контролером. Це дозволяє легко тестувати бізнес-логіку та взаємодію з даними окремо від користувацького інтерфейсу. Завдяки такій структурі, додаток можна легко розширювати та модифікувати, оскільки кожен компонент може еволюціонувати незалежно від інших. MVC також сприяє повторному використанню коду, адже логіка представлення та дані не переплітаються, що робить систему більш масштабованою і спрощує підтримку.

2.4. Реалізація веб-додатку

Розпочнемо реалізацію веб-додатку з розробки базових речей, компонентів ядра, налаштування підключень до БД і поступово перейдемо до розширення функціоналу, додавання онлайн-оплат та іншого.

Почнемо з налаштування маршрутизації. Для цього в корені проекту створимо файл `.htaccess`, де запишемо наступні команди:

```
RewriteEngine on
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule ^(.*)$ index.php?route=$1 [NC,L,QSA]
```

Створимо файл `database.php` в якому будуть знаходитись параметри нашої бази даних:

```
<?php
define('DATABASE_HOST','localhost');
define('DATABASE_LOGIN','root');
define('DATABASE_PASSWORD','');
define('DATABASE_BASENAME','jellycake');
```

Далі створюється клас `Core` який буде запускати сайт і обробляти дані нашого сайту:

```
class Core
{
    private function __construct()
    {
        global $pageParams;
        $this->app = [];
        $this->pageParams = $pageParams;
        $this->requestMethod = $_SERVER['REQUEST_METHOD'];
    }
    public static function getInstance() {
        if (empty(self::$instance)) {
            self::$instance = new self();
        }
        return self::$instance;
    }
    public function Initialize() {
        session_start();
        $this->db = new DB(DATABASE_HOST, DATABASE_LOGIN, DATABASE_PASSWORD,
DATABASE_BASENAME);
        $this->requestMethod = $_SERVER['REQUEST_METHOD'];
    }
}
```

І потім ці параметри і клас застосовуються на головній сторінці (index.php), тим самим включаючи проект:

```
<?php
include('config/database.php');
include('config/params.php');
require_once('LiqPay.php');
spl_autoload_register(function ($className) {
    $path = $className . '.php';
    if (is_file($path))
        require($path);
});
$core = \core\Core::getInstance();
$core->Initialize();
$core->Run();
$core->Done();
```

Для взаємодії з БД слід розробити основні методи, що необхідні для відправки базових SQL-запитів, таких як SELECT, UPDATE, INSERT, DELETE та інші. Створимо файл Core/DB.php – ще один важливий компонент ядра системи:

```
class DB
{
    protected $pdo;
    public function __construct($hostname, $login, $password, $database) {
        $this->pdo = new \PDO("mysql: host={$hostname};dbname={$database}", $login,
$password);
    }
    public function getConnection()
    {
        return $this->pdo;
    }
    public function insert($tableName, $newRowArray) {
        $fieldsArray = array_keys($newRowArray);
        $fieldsListString = implode(', ', $fieldsArray);
        $paramsArray = [];
        foreach ($newRowArray as $key => $value) {
            $paramsArray [] = ':' . $key;
        }
        $valuesListString = implode(', ', $paramsArray);
        $res = $this->pdo->prepare("INSERT INTO {$tableName} ($fieldsListString)
VALUES ($valuesListString)");
        $res->execute($newRowArray);
    }
    // Всі інші методи взаємодії з БД }
```

Доповнимо Core/DB.php методами для виклику SQL-процедур. Перший повинен просто викликати процедуру за її іменем та передавати необхідні параметри:

```
public function call($procName, $params = []){
    $placeholders = implode(' ',
        array_fill(0, count($params), '?'));
    $sql = "CALL $procName($placeholders)";
    $res = $this->pdo->prepare($sql);
    $res->execute($params);
    return $res->fetchAll(\PDO::FETCH_ASSOC);
}
```

Другий метод створимо спеціально для виклику специфічних процедур, котрі після завершення своєї роботи повертають boolean значення в задекларовану змінну (таким чином процедура звітує про результат своєї роботи, як-от deleteProduct()):

```
public function callOut($procName, $id, $outParam): bool
{
    $stmt = $this->pdo->prepare("SET $outParam = FALSE");
    $stmt->execute();
    $sql = "CALL $procName(?, $outParam)";
    $stmt = $this->pdo->prepare($sql);
    $stmt->execute([$id]);
    $query = $this->pdo->query("SELECT $outParam");
    $result = $query->fetch(\PDO::FETCH_ASSOC);
    return (bool) current($result);
}
```

Після успішного підключення БД до нашого проекту, треба створити клас для відображення даних наших сторінок, клас «Контролер», який є одним з головним компонентів патерну проектування MVC:

```
<?php
namespace core;
class Controller
{
    protected $viewPath;
    protected $moduleName;
    protected $actionName;
    public function __construct()
    {
        $this->moduleName = \core\Core::getInstance()->app['moduleName'];
        $this->actionName = \core\Core::getInstance()->app['actionName'];
        $this->viewPath = "views/{$this->moduleName}/{ $this->actionName}.php";
    }
}
```

```

        public function render($viewPath = null, $params = null) {
            if (empty($viewPath))
                $viewPath = $this->viewPath;
            $tpl = new Template($viewPath);
            if (!empty($params))
                $tpl->setParams($params);
            return $tpl->getHTML();
        }
    }
}

```

Для відображення сторінок треба створити дочірні класи-контролери які будуть наслідувати основний клас «Контролер», в дочірніх класах-контролерах будуть оброблятися дані і відображення сторінки. Нижче наведено приклад такого класу-контролера:

```

<?php
namespace controllers;
use core\Controller;
use models\User;

class SiteController extends Controller
{
    public function indexAction() {
        $user = User::GetCurrentUser();
        return $this->render(null,
            [
                'user' => $user,
            ]
        );
    }
    public function privacyAction()
    {
        return $this->render('views/site/privacy.php');
    }
    public function errorAction($code) {
        switch ($code) {
            case 404:
                return $this->render('views/site/error-404.php');
                break;
            case 403:
                return $this->render('views/site/error-403.php');
                break;
        }
    }
}

```

Також для кожної з таблиць потрібно розробити модель таблиці, де будуть знаходитись функції/CRUD-запити які обробляють запити до БД і повертають

потрібні значення, майже для кожної з таблиць потрібно мати таку модель. Нижче наведено приклад моделі для таблиці “Blog”:

```
class Blog
{
    protected static $tableName = 'blog';
    public static function addBlog($name,$text, $photoPath)
    {
        // логіка додавання поста в блог
    }

    public static function changePhoto($id, $newPhoto)
    {
        // алгоритм зміни фото поста
    }
}
```

Нарешті, підключимо онлайн-оплати. Для цього зареєструємось на сайті LiqPay, вивчимо документацію по роботі з API, отримаємо тестові public і private ключі. Встановимо офіційний LiqPay SDK згідно з офіційною документацією.

Тепер потрібно налаштувати ProductController, котрий, окрім базового функціоналу для керування сторінкою “Товари” також буде відповідати за створення замовлення, передачу ключових параметрів на сервер LiqPay, перенаправлення користувача на сторінку оплати та обробку статусу цієї оплати.

Розширимо базові методи контролера можливістю створення замовлень. Таким чином отримаємо потужний метод, що створює і записує в БД замовлення з унікальним ідентифікатором, упакує дані про нього в JSON, та, після шифрування, формує кнопку для переходу на сторінку оплати:

```
public function orderAction($params)
{
    // ініціалізація
    }
    $order_id = uniqid('order_');
    Core::getInstance()->db->call('createOrder', $params);
    $json_data = [
        "public_key" => $this->public_key,
        "version"    => "3",
        "action"     => "pay",
        "amount"     => (string)$totalAmount,                "currency"
=> "UAH",
```

```

        "description" => "Оплата замовлення",
        "order_id"     => $order_id,
        "result_url"   => "http://localhost/product/paymentResult/$order_id"
    ];
    $json_string = json_encode($json_data);
    $data = base64_encode($json_string);
    $signature = base64_encode(sha1($this->private_key . $data . $this->private_key, true));

    $form_html = '<br><br><br><br>
    <form      method="POST"      action="https://www.liqpay.ua/api/3/checkout"      accept-
    charset="utf-8">
        <input type="hidden" name="data" value="'. $data.'" />
        <input type="hidden" name="signature" value="'. $signature.'" />
        <input type="image" src="//static.liqpay.ua/buttons/payUk.png" />
    </form>';

    echo $form_html;
    } else {
        return $this->render(null, [
            'totalAmount' => 0,
        ]);
    }
}

```

Не забуваємо обробити результат оплати та актуалізувати дані в своїй базі, додавши відповідний метод в контролер:

```

public function paymentResultAction($params)
{
    $order_id = strval($params[0]);
    $liqpay = new LiqPay($this->public_key, $this->private_key);
    $res = $liqpay->api("request", array(
        'action'    => 'status',
        'version'   => '3',
        'order_id'  => $order_id
    ));
    if (is_object($res) && $res->status === 'success') {
        Order::updateOrderStatus($order_id, 'Сплачено');
        return $this->redirect('/user/orders');
    } else {
        Order::updateOrderStatus($order_id, 'Помилка');
        return $this->redirect('/user/orders');
    }
}

```

Зміна статусу оплати замовлення повинна оброблятися в межах БД. При успішній оплаті потрібно не тільки записати статус у відповідний рядок таблиці

orders, а й організувати переміщення пов'язаних даних про сплачені користувачем товари з таблиці carts до таблиці purchased_carts. Найпростіший спосіб – обробка таблиці orders тригером типу after update:

```
CREATE TRIGGER after_order_update
AFTER UPDATE ON orders
FOR EACH ROW
BEGIN
    IF LOWER(NEW.status) = 'сплачено' THEN
        INSERT INTO purchased_carts (order_id, user_id, product_id,
quantity)
        SELECT NEW.id, c.user_id, c.product_id, c.quantity
        FROM carts c
        WHERE c.user_id = NEW.user_id;
        DELETE FROM carts WHERE user_id = NEW.user_id;
    END IF;
END
```

Тепер, після успішної оплати замовлення, користувач в особистому кабінеті побачить деталі замовлення зі списком сплачених у цьому замовленні товарів. Адміністратор, в свою чергу, отримає дані про термінове замовлення і повинен швидко його обробити. В результаті, користувачу потрібно просто дочекатись доставки замовлення за вказаною адресою.

Висновки до другого розділу

Було виконано проектування веб-додатку, зокрема визначено варіанти використання, розроблено об'єктно-орієнтовану структуру системи та спроектовано базу даних. Розроблено діаграми варіантів використання, таблиці з описом основних варіантів та реалізовано архітектуру MVC, яка забезпечує чітке розподілення обов'язків між різними частинами додатку.

Показано реалізацію базових компонентів ядра – Core.php, Controller.php, DB.php. Також, детально описано роль важливих методів роботи ядра та контролерів, як-от orderAction, paymentResultAction, call, callOut та процедур БД – createOrder, deleteUser та deleteProduct. Продемонстровано інтеграцію SDK LiqPay та взаємодію з їх API в ході реалізації онлайн-платежів.

Це дозволило створити надійну основу для функціонування додатку та забезпечити його зручність і ефективність у використанні.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ДОДАТКОМ

3.1. Структура інтерфейсу інтернет-магазину

Коли користувач відкриває веб-сайт, його зустрічає головна сторінка, де в шапці знаходиться головне меню та інформативний контент з маркетинговими хитрощами трохи нижче. Зовнішній вигляд головного екрану наведено на рис. 3.1.

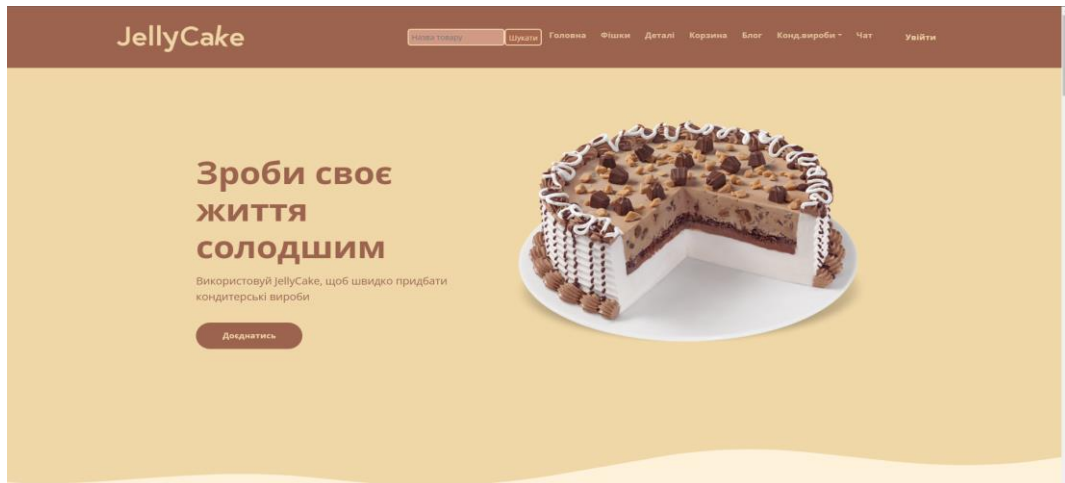


Рисунок 3.1 - Загальний вигляд сайту

На головній сторінці можна побачити загальне представлення сайту, товари, що пропонуються, відгуки експертів та контактні дані поруч із футером. Для того щоб отримати доступ до основного функціоналу сайту, потрібно авторизуватись (рис. 3.2).

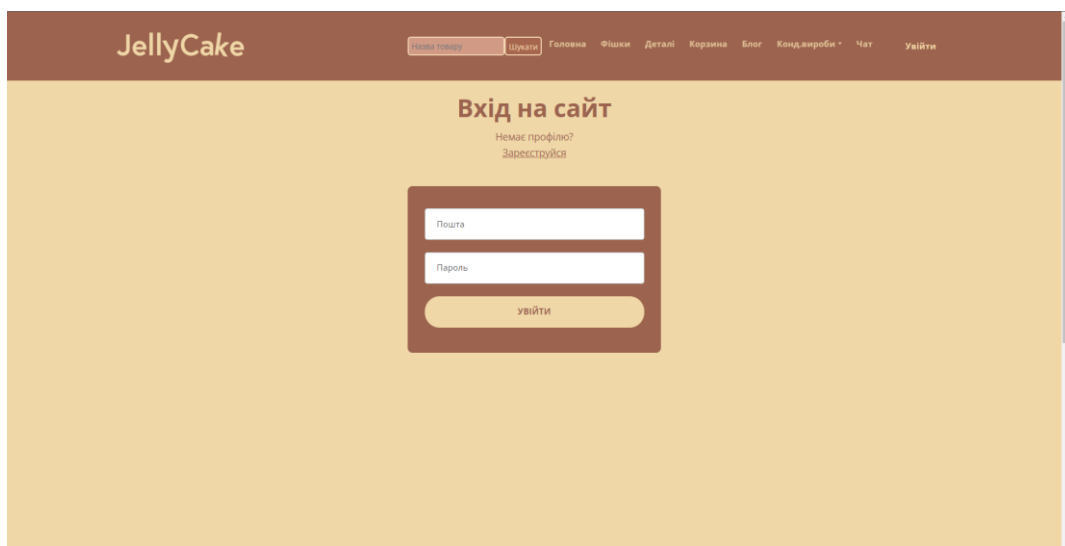


Рисунок 3.2 - Сторінка авторизації

Після авторизації користувач може: замовити товари, перейти у глобальний чат або відредагувати свої дані (рис. 3.3 – 3.5).

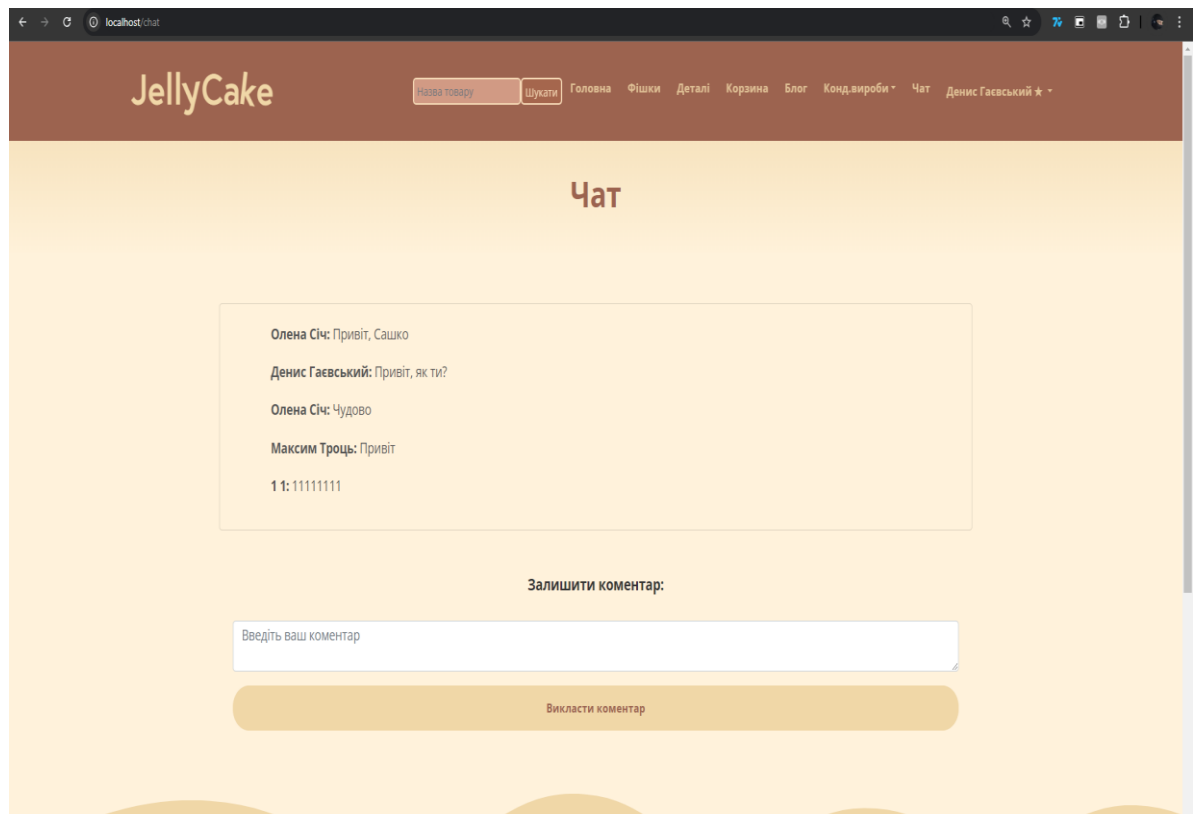


Рисунок 3.3 - Глобальний чат

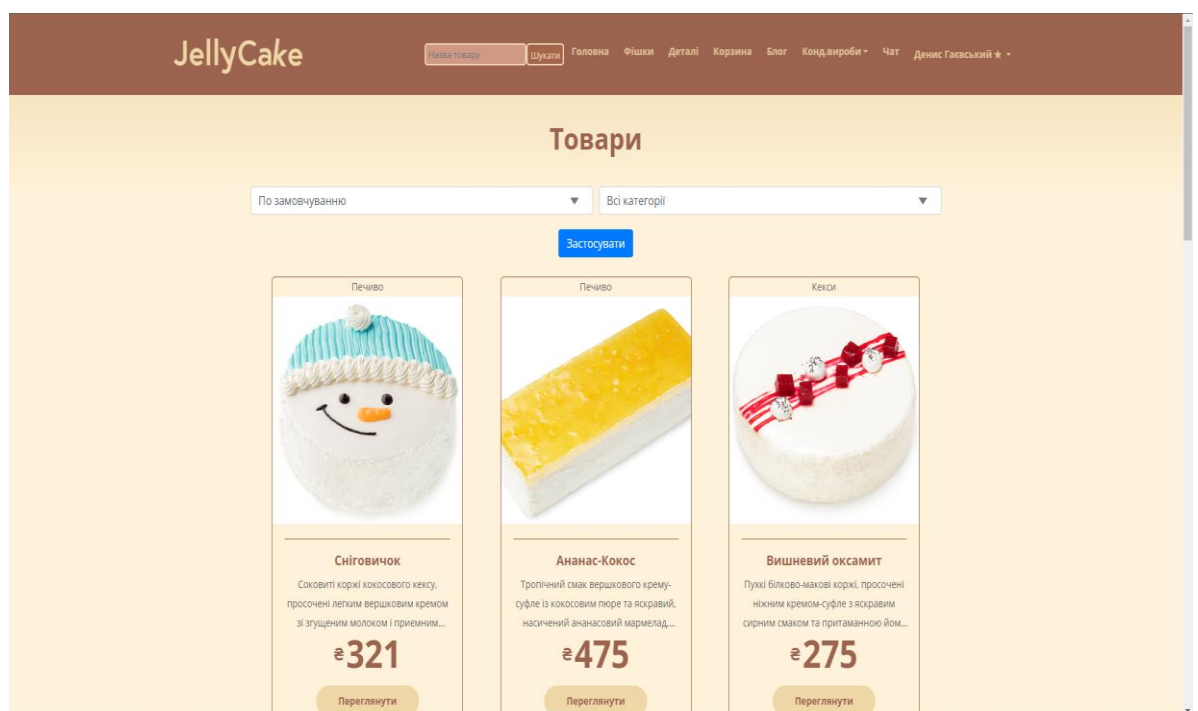


Рисунок 3.4 - Каталог товарів

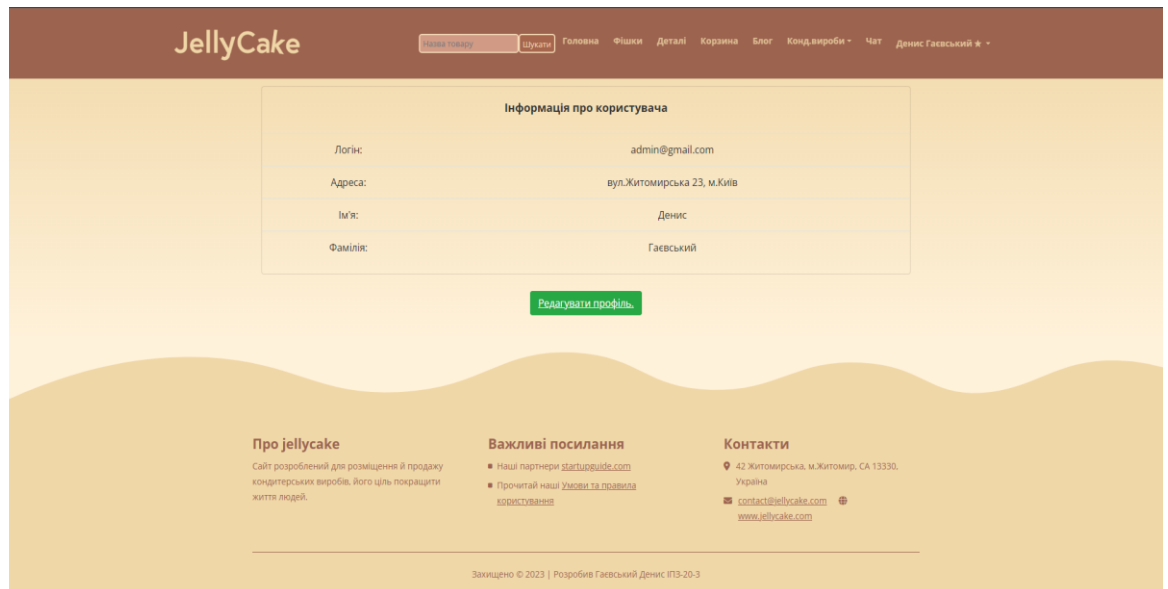


Рисунок 3.5 - Сторінка облікового запису

Після авторизації користувач може додавати в корзину необмежену кількість різноманітних товарів із загального асортименту солодоців, доступних для покупки (рис. 3.6).

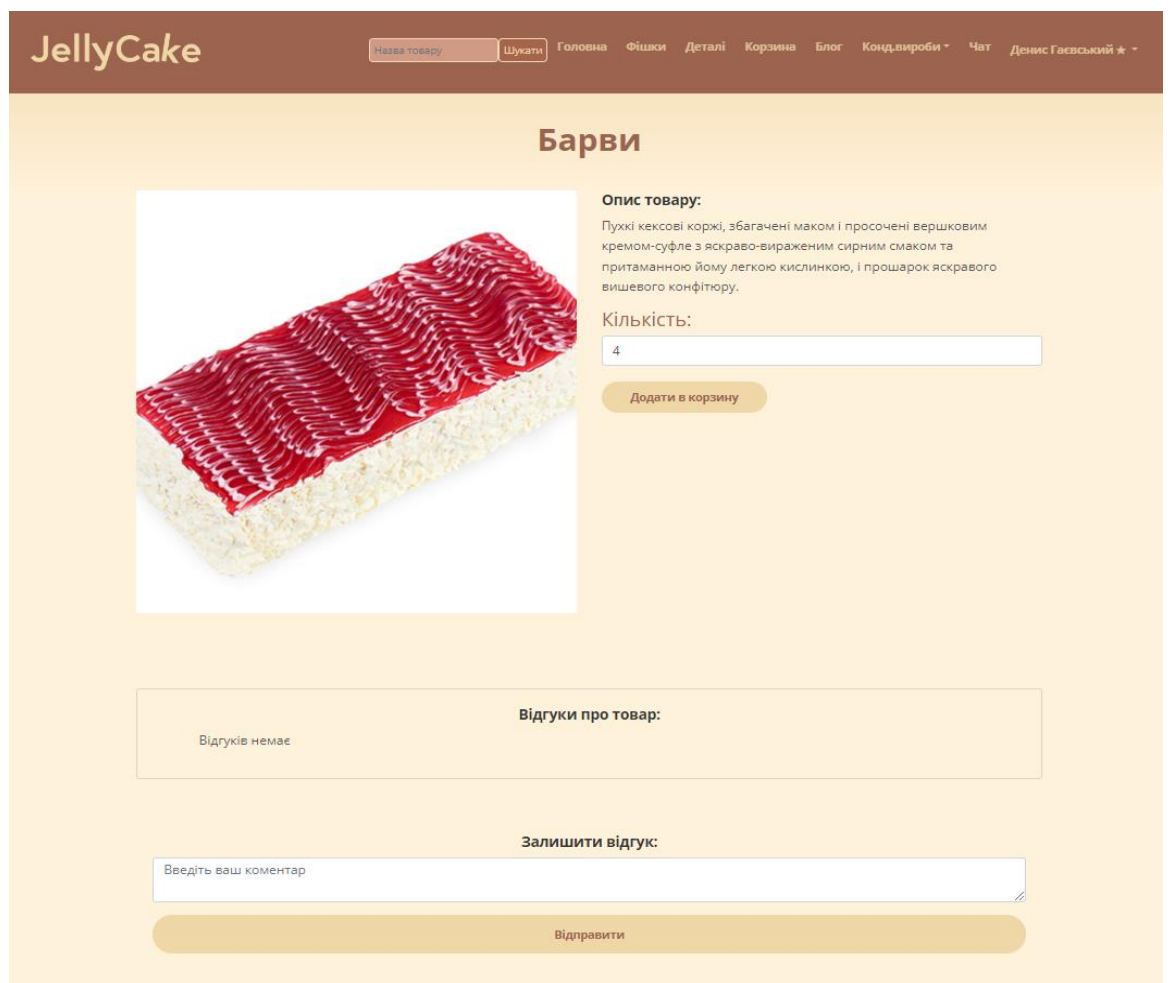


Рисунок 3.6 - Сторінка товару

Зібравши корзину товарів (рис. 3.7), користувач може відредагувати кількість прямо в кошику і перейти до оплати замовлення. В кошику детально відображається кожне найменування майбутнього замовлення: фото, назва, ціна, кількість, загальна ціна за кожен товар у заданій кількості та кінцева вартість замовлення загалом.

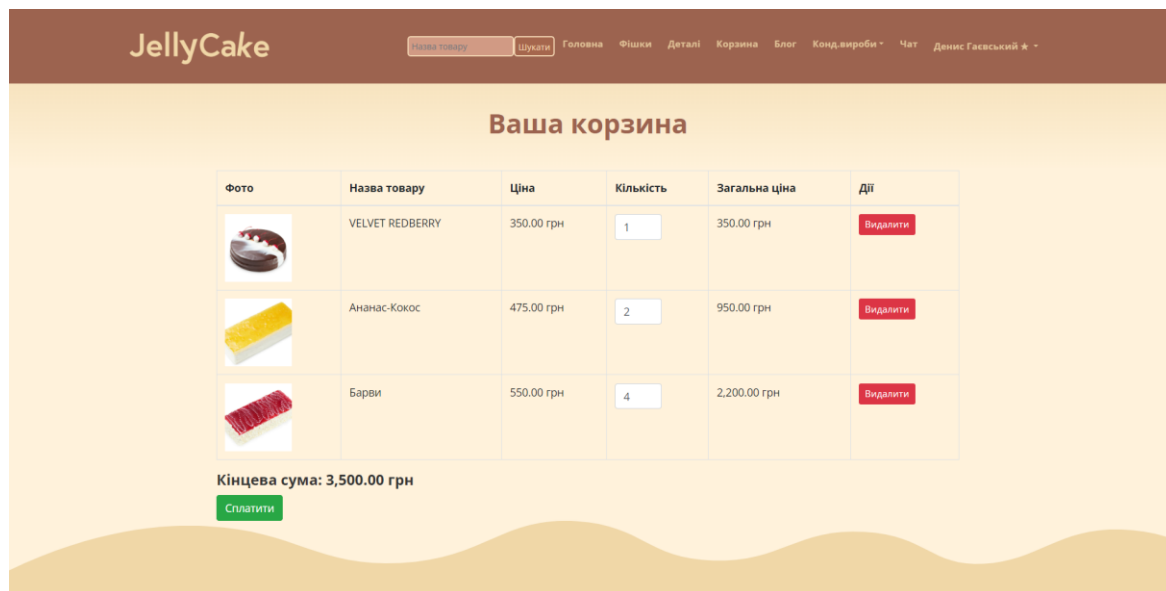


Рисунок 3.7 - Кошик обраних товарів

Якщо користувач хоче замовити ці товари, то він перейде на сторінку оформлення замовлення (рис. 3.8), де заповнить відповідні дані.

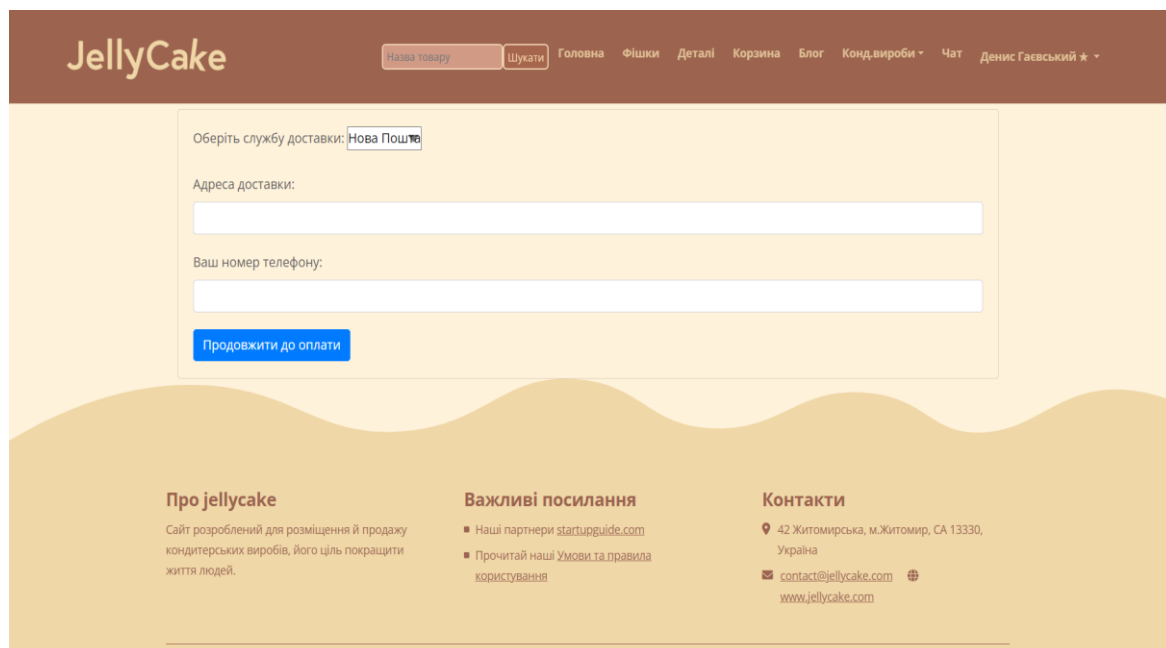


Рисунок 3.8 - Сторінка оформлення замовлення

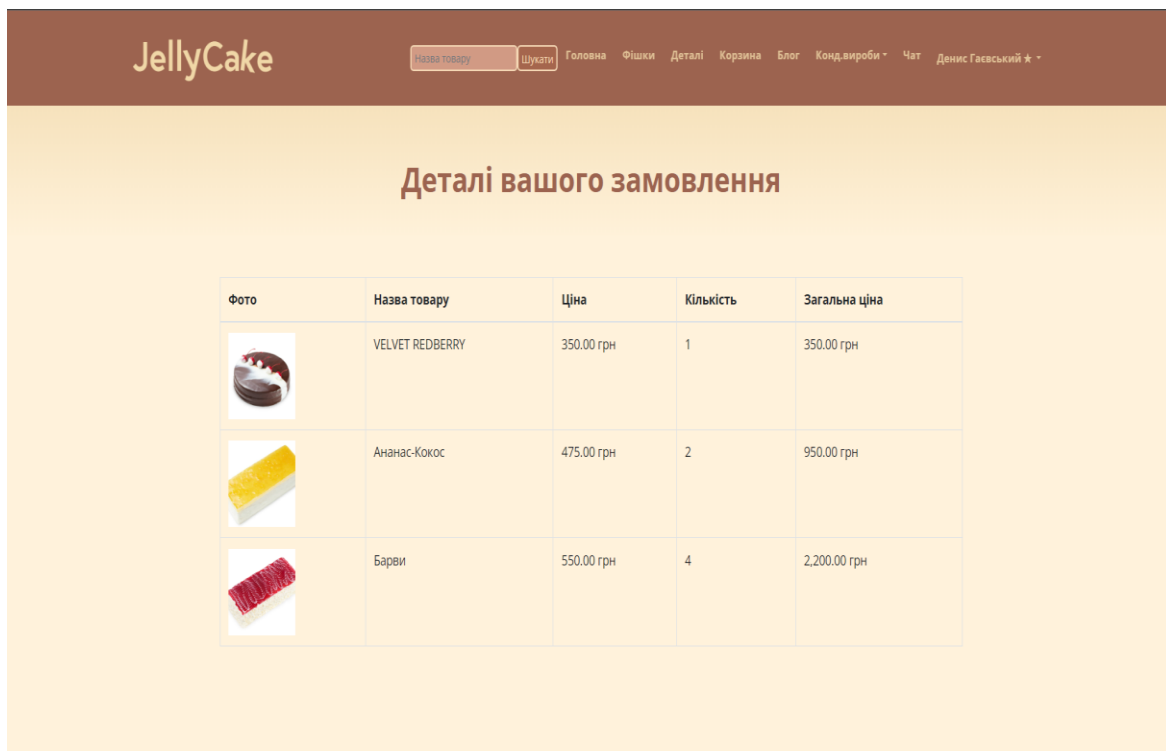


Рисунок 3.11 - Деталі одного з замовлень

Якщо користувач має права адміністратора додатку, тоді він має такі можливості: додавати, редагувати та видаляти товари, редагувати та видаляти коментарі до товарів. Має також доступ до адмін-панелі (рис. 3.12, 3.13), де знаходяться дані про всіх користувачів та замовлення.

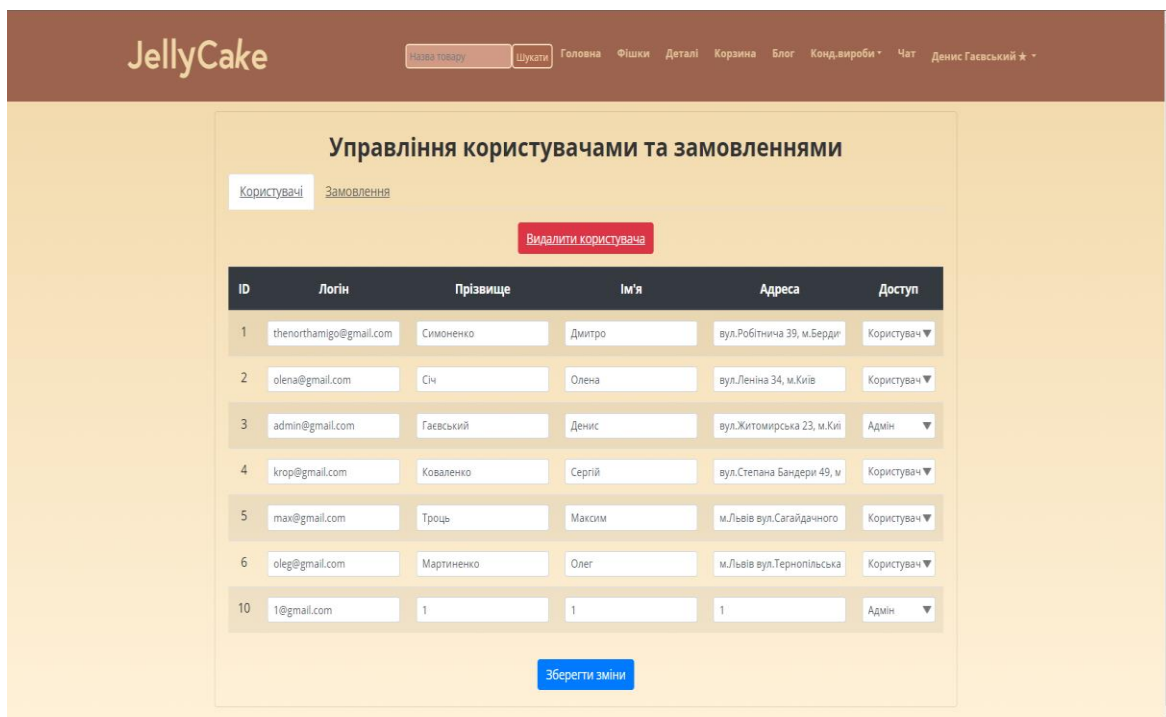


Рисунок 3.12 - Адмін панель (керування даними користувачів)

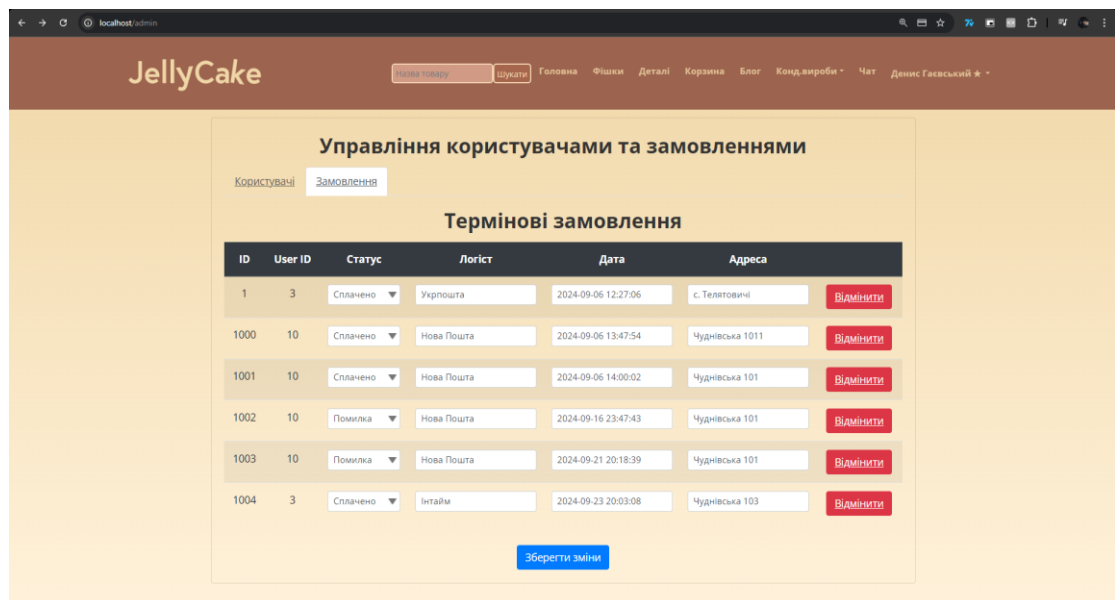


Рисунок 3.13 - Адмін панель (керування замовленнями)

3.2. Тестування

Мануальне тестування є одним з важливих етапів розробки програмного забезпечення, особливо для веб додатків, які взаємодіють з кінцевими користувачами в реальному часі. Метою мануального тестування є виявлення багів, недоліків інтерфейсу, а також перевірка функціональності додатку в різних умовах використання. У цьому розділі ми розглянемо підхід до мануального тестування розробленого веб додатку для продажу кондитерських виробів.

Перед початком тестування було визначено основні функціональні вимоги додатку, які підлягатимуть перевірці. Серед них:

- 1) Відсутність потреби в автентифікації.
- 2) Можливість пошуку товарів за короткими словами.
- 3) Можливість спілкування в живому чаті, та надання рекомендацій в коментарях.
- 4) Коректне заповнення кошику та оформлення замовлень.
- 5) Перегляд історії замовлень та їх статусу.
- 6) Коректна робота адмін-панелі.

Для прикладу тестування ми візьмемо форму оплати замовлень.

При введенні даних і натисканні на кнопку “Перейти до оплати” (рис. 3.14), в нашій БД створюється замовлення зі статусом “Сформовано” (рис. 3.15).

Рисунок 3.14 - Дані про тестове замовлення

1000 3 Сформовано Нова Пошта +38 098 123 4567 2024-09-25 10:06:08 Чуднівська 103 order_66f3b6607f7c8

Рисунок 3.15 - Запис замовлення в таблиці БД

Далі ми переходимо на сторінку LiqPay (рис. 3.18), де вводимо дані для проведення платежу. Для тестування ми візьмемо номери карток з офіційної документації LiqPay (рис. 3.17). В залежності від того, яку картку ми введемо, отримаємо результат.

Картки для тестування	
4242424242424242	Успішна оплата
40000000000003063	Успішна оплата з 3DS
40000000000003089	Успішна оплата з OTP
40000000000003055	Успішна оплата з CVV
40000000000000002	Не успішна оплата. Код помилки - limit
40000000000009995	Не успішна оплата. Код помилки - 9859
sandbox_token	Успішна оплата по токени

Рисунок 3.17 - Картки для тестування

Для тестування введемо картку з номером “4000000000009995”.

QR-код для оплати

Використовуйте Privat24

LIQPAY

Тестовий режим

Дані про оплату

Оплата замовлення

До оплати: 299.00 UAH

Pay Privat24

Оплатити через Pay Privat24

або

Картка Privat24 Рахунок

Номер картки

4000 0000 0000 9995 VISA

Термін дії 05/55 CV2

Ім'я власника картки* Денис

Прізвище власника картки* Гаєвський

☐ Відправити квитанцію на e-mail

Натискаючи на кнопку «Оплатити», ви підтверджуєте що ознайомлені з переліком інформації про послугу та приймаєте умови публічного договору

Оплатити 299.00 UAH

Скасувати оплату

Рисунок 3.18 - Сторінка оплати LiqPay

При поверненні на сайт, в статусі замовлення буде “Помилка” (рис. 3.19, 3.20). Для того щоб не було “Помилки”, треба повторно оплатити замовлення, на цей раз візьмемо картку з номером “4242424242424242” з офіційної документації.

Тестовий режим

Виникла помилка.
Недостатньо коштів

Спробувати ще раз

інші способи оплати

Картка Privat24

Повернутися до сайту

Рисунок 3.19 - Результат помилкової оплати

Ваші замовлення				
Дата	Адреса	Статус	Телефон	Доставка
2024-09-24 08:38:26 с.	Телятовичі	Сплачено	+380989898989	Укрпошта Перевірка замовлення , Деталі замовлення .
2024-09-25 10:06:08	Чуднівська 103	Помилка	+38 098 123 4567	Нова Пошта Перевірка замовлення , Деталі замовлення .

Рисунок 3.20 - Зміна статусу після оплати

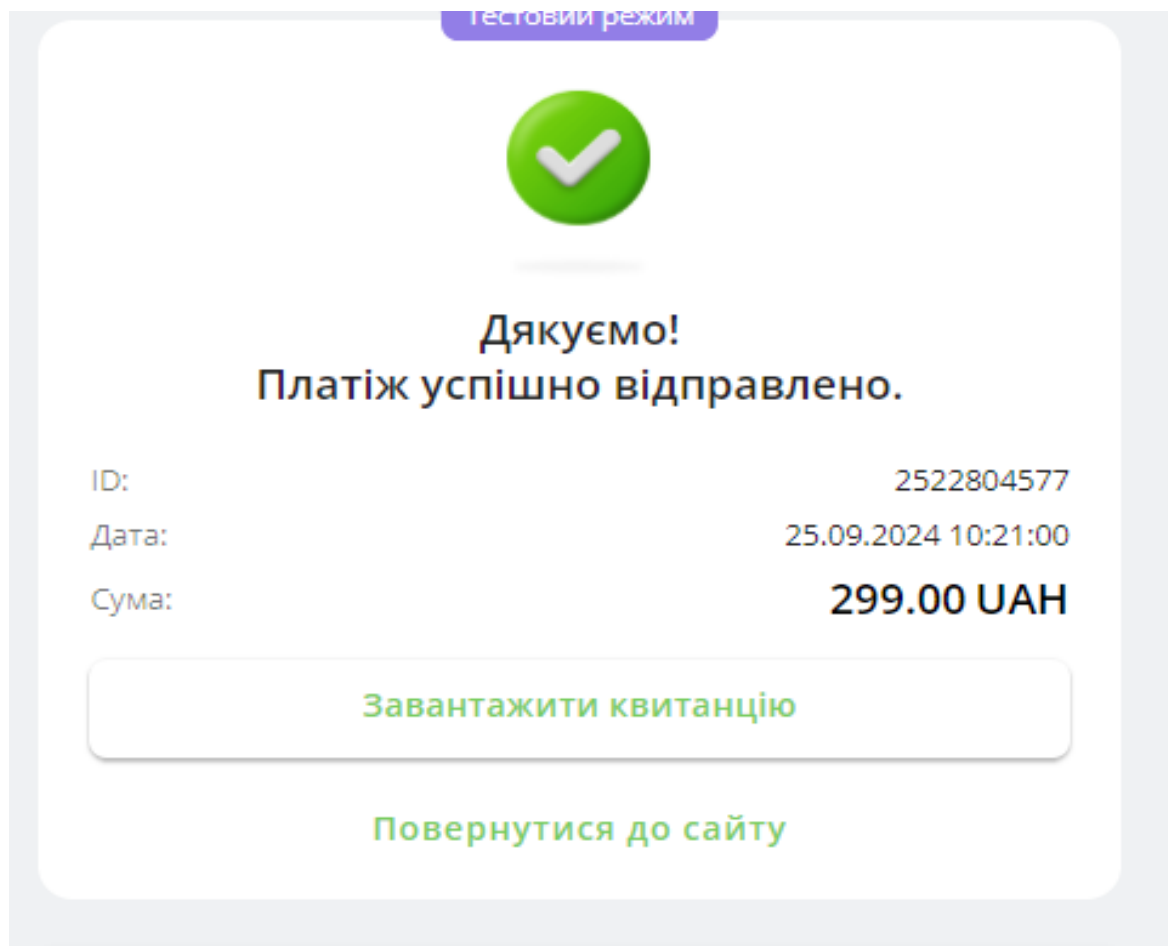


Рисунок 3.21 - Результат успішної оплати

Ваші замовлення				
Дата	Адреса	Статус	Телефон	Доставка
2024-09-24 08:38:26 с.	Телятовичі	Сплачено	+380989898989	Укрпошта Перевірка замовлення , Деталі замовлення .
2024-09-25 10:06:08	Чуднівська 103	Помилка	+38 098 123 4567	Нова Пошта Перевірка замовлення , Деталі замовлення .
2024-09-25 10:19:20	Чуднівська 103	Сплачено	+38 098 123 4567	Нова Пошта Перевірка замовлення , Деталі замовлення .

Рисунок 3.22 - Зміна статусу після оплати

Висновки до третього розділу

У третьому розділі було розглянуто інтерфейс та порядок роботи з додатком. Було описано структуру інтерфейсу інтернет-магазину, включаючи головну сторінку, кнопки (пошук, перегляд каталогу товарів, обліковий запис, замовлення, кошик, чат, блог). Проведено тестування інтернет-магазину, виявлено та виправлено критичні баги, що забезпечить його стабільну та надійну роботу.

ВИСНОВКИ

Під час реалізації першого розділу було вивчено сайти зі схожою тематикою та їх функціонал. Визначено головні критерії для веб-додатку після аналізу аналогів. Водночас, були знайдені способи досягнення кращих результатів, ніж у конкурентів. Крім цього, були сформовані основні вимоги для власної платформи. На основі відповідних вимог та критеріїв був сформований набір основних технологій і засобів реалізації для розробки.

У другому розділі була виконано проектування інтернет-магазину, зокрема визначено варіанти використання, розроблено об'єктно-орієнтовану структуру системи та спроектовано базу даних. Розроблено діаграми варіантів використання, таблиці з описом основних варіантів та реалізовано архітектуру MVC, яка забезпечує чітке розподілення обов'язків між різними частинами додатку. Показано реалізацію базових компонентів. Також, детально описано роль важливих методів роботи ядра та контролерів.

Під час виконання третього розділу було приділено особливу увагу інтерфейсу. Проведено тестування критично важливого функціоналу для виявлення можливих помилок чи вразливостей веб-сайту.

В результаті створено оптимізований інтернет-магазин кондитерських виробів. Розроблена структура програмного продукту гарантує ефективну обробку інформації та забезпечує зручний інтерфейс для користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. PHP documentation [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.php.net/docs.php>
2. MySQL documentation [Електронний ресурс] – Режим доступу до ресурсу:
<https://dev.mysql.com/doc>
3. OSP documentation [Електронний ресурс] – Режим доступу до ресурсу:
<https://ospanel.io/docs>
4. Pro PHP 8 MVC [Електронний ресурс] – Режим доступу до ресурсу:
<https://doi.org/10.1007/978-1-4842-6957-2>
5. C. Bell, "Introducing Innodb Cluster: Learning the Mysql High Availability Stack.", Apress 2018.
6. J. Duckett, "PHP & Mysql: Server-Side Web Development." Wiley, 2020.
7. W. Dougherty, "PHP Programming Language for Server-Side Web Development." Independently Published, 2022.
8. V. M. Grippa and S. Kuzmichev, "Learning MySQL.", O'Reilly Media, Incorporated, 2021.
9. E. Vanier, B. Shah and T. Malepati, "Advanced Mysql 8: Discover the Full Potential of Mysql and Ensure High Performance of Your Database." Packt Publishing, Limited, 2019.
10. E. Harbeson, "PHP Programming : Take Your Skills to Another Level by Learning Programming in PHP: PHP Object Oriented Programming." Independently Published, 2021.
11. K. Marks, K. Ferguson and O. Merida, "PHP Web Development With Mysql", 2021.

ДОДАТКИ

					ПЗ.КР.Б-121-24-ПЗ	
						58

Лістинг SQL

```

#region SETUP
SET SQL MODE = "NO AUTO VALUE ON ZERO";
START TRANSACTION;
SET time_zone = "+02:00";
DROP DATABASE IF EXISTS `jellycake`;
CREATE DATABASE `jellycake` DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
USE jellycake;
#endregion

#region tables creation
CREATE TABLE `users` (
    `id` int(11) NOT NULL,
    `login` varchar(50) NOT NULL,
    `password` varchar(255) NOT NULL,
    `access` int(11) DEFAULT NULL,
    `lastname` varchar(255) NOT NULL,
    `firstname` varchar(255) NOT NULL,
    `address` text NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
CREATE TABLE `categories` (
    `id` int(11) NOT NULL,
    `name` varchar(50) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
CREATE TABLE `products` (
    `id` int(11) NOT NULL,
    `title` varchar(255) NOT NULL,
    `description` text NOT NULL,
    `price` int(11) NOT NULL,
    `datetime` datetime NOT NULL,
    `datetime_lastedit` datetime DEFAULT NULL,
    `photo` varchar(255) NOT NULL,
    `user_id` int(11) NOT NULL,
    `category_id` int(11)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
CREATE TABLE carts (
    user_id int(11) NOT NULL,
    product_id int(11) NOT NULL,
    quantity int(11) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
CREATE TABLE purchased_carts (
    order_id int(11) NOT NULL,
    user_id int(11) NOT NULL,
    product_id int(11) NOT NULL,
    quantity int(11) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
CREATE TABLE `comments` (
    `id` int(11) NOT NULL,
    `message` text NOT NULL,
    `datetime` datetime NOT NULL,
    `post_id` int(11) NOT NULL,
    `user_id` int(11) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
CREATE TABLE `chat` (
    `id` int(11) NOT NULL,
    `message` text NOT NULL,
    `datetime` datetime NOT NULL,
    `user_id` int(11) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
CREATE TABLE orders (
    `id` int(11) NOT NULL,
    `user_id` int(11) NOT NULL,
    `status` varchar(255) NOT NULL,
    `service` varchar(255) NOT NULL,
    phone varchar(255) NOT NULL,
    `date` datetime NOT NULL DEFAULT current_timestamp(),
    `address` varchar(255) NOT NULL,
    unique_id varchar(255)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
CREATE TABLE `blog` (
    `id` int(11) NOT NULL,
    `created_at` timestamp NOT NULL DEFAULT current_timestamp(),
    `name` varchar(255) DEFAULT NULL,
    `photo` blob DEFAULT NULL,
    `text` varchar(1000) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;

```

```
#endregion

#region inserts
INSERT INTO `users` (`id`, `login`, `password`, `access`, `lastname`, `firstname`, `address`) VALUES
(1,
'thenorthamigo@gmail.com', '268622c271c2309bd73d20309d2c8669', 0, 'Симоненко', 'Дмитро',
'вул.Робітнича 39, м.Бердичів'),
(2,
'olena@gmail.com', '46929ec3d692ebf1607c23f53bfa566e', 0, 'Січ', 'Олена', 'вул.Леніна 34, м.Київ'),
(3,
'admin@gmail.com', '21232f297a57a5a743894a0e4a801fc3', 1, 'Гаєвський', 'Денис', 'вул.Житомирська 23,
м.Київ'),
(4,
'krop@gmail.com', '50e65e6128bbba393694c074ecba8c18', 0, 'Коваленко', 'Сергій', 'вул.Степана Бандери
49, м.Львів'),
(5,
'max@gmail.com', '2ffe4e77325d9a7152f7086ea7aa5114', 0, 'Троць', 'Максим', 'м.Львів вул.Сагайдачного
432'),
(6,
'oleg@gmail.com', '045b9e4d8b96dce053950297a8a39665', 0, 'Мартиненко', 'Олег', 'м.Львів
вул.Тернопільська 57'),);
INSERT INTO categories (`id`, `name`) VALUES
(21, 'Тістечка'),
(22, 'Морозиво'),
(23, 'Цукерки'),
(1, 'Торти'),
(2, 'Кекси'),
(3, 'Печиво');
INSERT INTO `products` (`id`, `title`, `description`, `price`, `datetime`, `datetime_lastedit`,
`photo`, `user id`, category id) VALUES
(11, 'Медовик', 'Медовик за особливим домашнім рецептом - ароматні
медові коржі із заварного тіста, просочені кремом на основі крем-сиру з притаманною йому кислінкою.
На довершення - яскравий післясмак медового різнотрав'я.', 299, '2022-02-08 09:51:35', '2024-07-28
15:59:54', '11_66a64edadfe38.jpg', 3, 1),
(12, 'Барвінок', 'Вишуканий торт у більшості складу входить саме
желе й крем, фрукти всередині. Стане чудовою прикрасою на вашому святковому столі а також досить
смачним делікатесом для всіх гостей.', 550, '2022-02-09 21:08:43', '2023-12-15 01:00:33',
'12_657b7afe997d5_b.jpg', 1, 2),
(13, 'VELVET REDBERRY', 'Ідеальний смак насиченого шоколадного кексу
та яскравої вишні, що гармонічно підкреслюється ніжними нотками білого шоколаду.', 350, '2022-02-09
21:10:36', '2023-12-15 00:57:43', '13_657b7a56e3706_b.jpg', 1, 1),
(14, 'Тропікана', 'Пухкі коржі жовтого бісквіту, просочені фруктовим
сиропом та легким кремом із додаванням фруктового наповнювача зі шматочками ананасу. Прикрашений
мармеладом з ківі і скибками ананасу, вкритими тонким шаром желе.', 450, '2022-02-09 21:12:36',
'2023-12-15 00:56:27', '14_657b7a0aaa4c8_b.jpg', 1, 1),
(15, 'Барви', 'Пухкі кексові коржі, збагачені маком і просочені
вершковим кремом-суфле з яскраво-вираженим сирним смаком та притаманною йому легкою кислінкою, і
прошарок яскравого вишневого конфітуру.', 550, '2022-02-09 21:53:03', '2023-12-15 00:55:32',
'15_657b79d2d3381_b.jpg', 2, 2),
(16, 'Вишневий оксамит', 'Пухкі білково-макові коржі, просочені
ніжним кремом-суфле з яскравим сирним смаком та притаманною йому приємною кислінкою, і прошарок
мармеладу з цільними ягодами вишні.', 275, '2022-02-09 21:54:35', '2023-12-15
00:54:04', '16_657b797a15650_b.jpg', 2, 2),
(18, 'Ананас-Кокос', 'Тропічний смак вершкового крему-суфле із
кокосовим пюре та яскравий, насичений ананасовий мармелад, викладені на соковиті коржі жовтого
бісквіту.', 475, '2022-12-09 19:31:42', '2023-12-15 00:52:54', '18_657b79361b9eb_b.jpg', 3, 3),
(19, 'Сніговичок', 'Соковиті коржі кокосового кексу, просочені
легким вершковим кремом зі згущеним молоком і приємним смаком білого шоколаду та нотками ванілі.
Ззовні бережно огорнутий кокосовою стружкою та прикрашений кокосовими кульками.', 321, '2022-12-11
02:12:30', '2023-12-15 00:47:38', '19_657b77f9b5c6b_b.jpg', 5, 3);
INSERT INTO carts (user id, `product id`, `quantity`) VALUES
(1, 11, 1),
(2, 12, 2),
(3, 13, 1),
(4, 14, 4),
(5, 15, 1),
(6, 16, 1),
(7, 18, 4),
(10, 19, 5);
INSERT INTO orders (id, user_id, status, service, phone, address) VALUES
```

```

(1, 3, 'Сплачено', 'Укрпошта',
'+380989898989', 'с. Телятовичі');
INSERT INTO purchased_carts (order_id, user_id, `product_id`, `quantity`) VALUES
(1, 3, 11, 1);
INSERT INTO `comments` (`id`, `message`, `datetime`, `post_id`, `user_id`) VALUES
(1, 'Доволі не погані
джинси за свою ціну, якісний матеріал...', '2022-11-30 21:10:59', 16, 3),
(2, 'Класний костюм',
'2022-12-11 02:09:03', 18, 5),
(3, 'Досить тепла
кофта', '2022-12-11 02:09:58', 14, 5),
(4, 'Смачнючий торт',
'2023-12-15 00:26:49', 19, 6),
(5, 'Ыыыы', '2024-07-
19 16:03:48', 18, 3);
INSERT INTO `chat` (`id`, `message`, `datetime`, `user_id`) VALUES
(1, 'Привіт, Сашко', '2022-11-10
04:18:19', 2),
(2, 'Привіт, як ти?', '2022-11-28
18:47:12', 3),
(3, 'Чудово', '2022-12-09 09:40:35',
2),
(4, 'Привіт', '2022-12-11 02:01:45',
5);
INSERT INTO `blog` (`id`, `created_at`, `name`, `photo`, `text`) VALUES
(0, '2024-08-10 13:43:51', 'Медаль Солодкого Майстра', 0x363662373665393732376533392e6a7067, '');
#endregion

#region primary keys
ALTER TABLE `users`
ADD PRIMARY KEY (`id`);
ALTER TABLE `categories`
ADD PRIMARY KEY (`id`);
ALTER TABLE `products`
ADD PRIMARY KEY (`id`);
ALTER TABLE orders
ADD PRIMARY KEY (`id`),
ADD KEY `customer` (`user_id`),
MODIFY `id` int(11) NOT NULL AUTO INCREMENT, AUTO INCREMENT=1000;
ALTER TABLE `comments`
ADD PRIMARY KEY (`id`);
ALTER TABLE `chat`
ADD PRIMARY KEY (`id`);
ALTER TABLE `blog`
ADD PRIMARY KEY (`id`);
#endregion

#region auto increments
ALTER TABLE `chat`
MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=5;
ALTER TABLE `categories`
MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=1;
ALTER TABLE `comments`
MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=6;
ALTER TABLE `products`
MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=21;
ALTER TABLE `users`
MODIFY `id` int(11) NOT NULL AUTO INCREMENT, AUTO INCREMENT=11;
ALTER TABLE `blog`
MODIFY `id` int(11) NOT NULL AUTO INCREMENT, AUTO INCREMENT=1;
#endregion

#region constraints
ALTER TABLE `products`
ADD CONSTRAINT `user_fk` FOREIGN KEY (`user_id`) REFERENCES users (`id`) ON DELETE CASCADE,
ADD CONSTRAINT `category_product` FOREIGN KEY (`category_id`) REFERENCES categories (`id`) ON
DELETE CASCADE;
ALTER TABLE `comments`
ADD CONSTRAINT `user_comment` FOREIGN KEY (`user_id`) REFERENCES users (`id`) ON DELETE CASCADE,
ADD CONSTRAINT `product comment` FOREIGN KEY (`post_id`) REFERENCES products (id) ON DELETE
CASCADE;
ALTER TABLE `carts`
ADD CONSTRAINT `user_cart` FOREIGN KEY (user_id) REFERENCES users (id) ON DELETE CASCADE,
ADD CONSTRAINT `product cart` FOREIGN KEY (product_id) REFERENCES products (id) ON DELETE
CASCADE;
ALTER TABLE `purchased_carts`
ADD CONSTRAINT `order` FOREIGN KEY (order_id) REFERENCES orders (id) ON DELETE CASCADE,
ADD CONSTRAINT `purchaser` FOREIGN KEY (user_id) REFERENCES users (id) ON DELETE CASCADE,

```

```

    ADD CONSTRAINT `purchased product` FOREIGN KEY (product_id) REFERENCES products (id) ON DELETE
    CASCADE;
    ALTER TABLE `chat`
        ADD CONSTRAINT `user_chat` FOREIGN KEY (`user_id`) REFERENCES users (`id`) ON DELETE CASCADE;
    ALTER TABLE orders
        ADD CONSTRAINT `customer` FOREIGN KEY (`user_id`) REFERENCES users (`id`);
#endregion

#region TRIGGERS
DELIMITER //
CREATE TRIGGER after_order_update
    AFTER UPDATE ON orders
    FOR EACH ROW
BEGIN
    IF LOWER(NEW.status) = 'списано' THEN
        INSERT INTO purchased_carts (order_id, user_id, product_id, quantity)
        SELECT NEW.id, c.user_id, c.product_id, c.quantity
        FROM carts c
        WHERE c.user_id = NEW.user_id;
        DELETE FROM carts WHERE user_id = NEW.user_id;
    END IF;
END//
DELIMITER ;
#endregion

#region PROCEDURES
DELIMITER //
CREATE PROCEDURE deleteUser(IN userId INT)
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE tbl_name VARCHAR(255);
    DECLARE col_name VARCHAR(255);

    DECLARE cur CURSOR FOR
        SELECT
            TABLE_NAME, COLUMN_NAME
        FROM
            information schema.KEY COLUMN USAGE
        WHERE
            REFERENCED_TABLE_NAME = 'users'
            AND REFERENCED_COLUMN_NAME = 'id';

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    DELETE comm
    FROM comments comm
        JOIN products prod ON comm.post_id = prod.id
    WHERE prod.user_id = userId;

    OPEN cur;

    read loop: LOOP
        FETCH cur INTO tbl_name, col_name;
        IF done THEN
            LEAVE read_loop;
        END IF;

        SET @del_stmt = CONCAT('DELETE FROM `', tbl_name, '` WHERE `', col_name, '` = ', userId);
        PREPARE stmt FROM @del_stmt;
        EXECUTE stmt;
        DEALLOCATE PREPARE stmt;
    END LOOP;

    CLOSE cur;

    DELETE FROM users WHERE `id` = userId;
END //
DELIMITER ;

DELIMITER //
CREATE PROCEDURE deleteProduct(IN productId INT, OUT isDeleted BOOLEAN)
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE tbl_name VARCHAR(255);
    DECLARE col_name VARCHAR(255);

```

```

DECLARE cur CURSOR FOR
SELECT
    TABLE_NAME, COLUMN_NAME
FROM
    information_schema.KEY_COLUMN_USAGE
WHERE
    REFERENCED TABLE NAME = 'products'
    AND REFERENCED_COLUMN_NAME = 'id';

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
IF EXISTS (
    SELECT 1
    FROM purchased_carts
    WHERE product_id = productId
) THEN
    SET isDeleted = false;
ELSE
    OPEN cur;

    read_loop: LOOP
        FETCH cur INTO tbl name, col name;
        IF done THEN
            LEAVE read_loop;
        END IF;

        SET @del stmt = CONCAT('DELETE FROM `', tbl name, '` WHERE `', col name, '` = ',
productId);
        PREPARE stmt FROM @del stmt;
        EXECUTE stmt;
        DEALLOCATE PREPARE stmt;
    END LOOP;

    CLOSE cur;

    DELETE FROM products WHERE `id` = productId;

    SET isDeleted = true;
END IF;
END //
DELIMITER ;

DELIMITER //
CREATE PROCEDURE createOrder(
    IN p_user_id INT,
    IN p_service VARCHAR(255),
    IN p_address VARCHAR(255),
    IN p_phone VARCHAR(255),
    IN p_unique_id VARCHAR(255)
)
BEGIN
    INSERT INTO orders (user_id, status, service, phone, address, unique_id)
    VALUES (p_user_id, 'Сформовано', p_service, p_phone, p_address, p_unique_id);
END //
DELIMITER ;
#endregion

#region END
COMMIT;
#endregion

```

Додаток Б

Лістинг Core

Core.php

```

<?php
namespace core;
use controllers\SiteController;
class Core
{
    private static $instance = null;
    public $app;
    public $pageParams;
    public $db;

```

```

public $requestMethod;
private function __construct()
{
    global $pageParams;
    $this->app = [];
    $this->pageParams = $pageParams;
    $this->requestMethod = $_SERVER['REQUEST_METHOD'];
}
public static function getInstance() {
    if (empty(self::$instance)) {
        self::$instance = new self();
    }
    return self::$instance;
}
public function Initialize() {
    session_start();
    $this->db = new DB(DATABASE_HOST, DATABASE_LOGIN, DATABASE_PASSWORD, DATABASE_BASENAME);
    $this->requestMethod = $_SERVER['REQUEST_METHOD'];
}
public function Run() {
    $route = $_GET['route'] ?? '';
    $routeParts = explode('/', $route);
    $moduleName = strtolower(array_shift($routeParts));
    $actionName = strtolower(array_shift($routeParts));
    if (empty($moduleName))
        $moduleName = "site";
    if (empty($actionName))
        $actionName = "index";
    $this->app['moduleName'] = $moduleName;
    $this->app['actionName'] = $actionName;
    $controllerName = '\\controllers\\'.ucfirst($moduleName).'Controller';
    $controllerActionName = $actionName.'Action';
    $statusCode = 200;
    if (class_exists($controllerName)) {
        $controller = new $controllerName();

        if (method_exists($controller, $controllerActionName)) {
            $actionResult = $controller->$controllerActionName($routeParts);
            if ($actionResult instanceof Error)
                $statusCode = $actionResult->code;
            $this->pageParams['content'] = $actionResult;
        }
        else {
            $statusCode = 404;
        }
    }
    else {
        $statusCode = 404;
    }
    $statusCodeType = intval($statusCode / 100);
    if ($statusCodeType == 4 || $statusCodeType == 5) {
        $mainController = new SiteController();
        $this->pageParams['content'] = $mainController->errorAction($statusCode);
    }
}
public function Done() {
    $pathToLayout = 'views/layout/index.php';
    $tpl = new Template($pathToLayout);
    $tpl->setParams($this->pageParams);
    $html = $tpl->getHTML();
    echo $html;
}
}

```

DB.php

```

<?php
namespace core;
class DB
{
    protected $pdo;
    public function __construct($hostname, $login, $password, $database) {
        $this->pdo = new \PDO("mysql: host={$hostname};dbname={$database}", $login, $password);
    }
    public function getConnection()
    {
        return $this->pdo;
    }

    public function select(string $tableName, $fieldsList = "*", $conditionArray = null)

```

```

{
    $fieldsListString = is_string($fieldsList) ? $fieldsList : implode(',', $fieldsList);
    $wherePartString = '';
    $params = [];

    if (is_array($conditionArray)) {
        $parts = [];
        foreach ($conditionArray as $key => $value) {
            $placeholder = str_replace('[~]', '', $key);
            if (strpos($key, '[~]') !== false) {
                $parts[] = "{$placeholder} LIKE :{$placeholder}";
                $params[$placeholder] = "%{$value}%";
            } else {
                $parts[] = "{$key} = :{$key}";
                $params[$key] = $value;
            }
        }
        $wherePartString = "WHERE " . implode(' AND ', $parts);
    }

    $res = $this->pdo->prepare("SELECT {$fieldsListString} FROM {$tableName} {$wherePartString}");
    $res->execute($params);
    return $res->fetchAll(PDO::FETCH_ASSOC);
}

public function update($tableName, $newValuesArray, $conditionArray) {
    $setParts = [];
    $paramsArray = [];
    foreach ($newValuesArray as $key => $value) {
        $setParts [] = "{$key} = :set{$key}";
        $paramsArray['set.' . $key] = $value;
    }
    $setPartString = implode(',', $setParts);

    $whereParts = [];
    foreach ($conditionArray as $key => $value) {
        $whereParts [] = "{$key} = :{$key}";
        $paramsArray[$key] = $value;
    }
    $wherePartString = "WHERE " . implode(' AND ', $whereParts);
    $res = $this->pdo->prepare("UPDATE {$tableName} SET {$setPartString} {$wherePartString}");
    $res->execute($paramsArray);
}

public function insert($tableName, $newRowArray) {
    $fieldsArray = array_keys($newRowArray);
    $fieldsListString = implode(',', $fieldsArray);
    $paramsArray = [];
    foreach ($newRowArray as $key => $value) {
        $paramsArray [] = ':' . $key;
    }
    $valuesListString = implode(',', $paramsArray);
    $res = $this->pdo->prepare("INSERT INTO {$tableName} ({$fieldsListString}) VALUES ({$valuesListString})");
    $res->execute($newRowArray);
}

public function delete($tableName, $conditionArray) {
    $whereParts = [];
    foreach ($conditionArray as $key => $value) {
        $whereParts [] = "{$key} = :{$key}";
        $paramsArray[$key] = $value;
    }
    $wherePartString = "WHERE " . implode(' AND ', $whereParts);
    $res = $this->pdo->prepare("DELETE FROM {$tableName} {$wherePartString}");
    $res->execute($conditionArray);
}

public function call($procName, $params = []) {
    $placeholders = implode(',', array_fill(0, count($params), '?'));
    $sql = "CALL $procName($placeholders)";
    $res = $this->pdo->prepare($sql);
    $res->execute($params);
    return $res->fetchAll(PDO::FETCH_ASSOC);
}

public function callOut($procName, $id, $outParam): bool
{
    // Спочатку оголошуємо змінну для вихідного параметра
    $stmt = $this->pdo->prepare("SET $outParam = FALSE");
}

```

```

        $stmt->execute();
        // Підготовка SQL для виклику процедури
        $sql = "CALL $procName(?, $outParam)";
        $stmt = $this->pdo->prepare($sql);
        $stmt->execute([$id]);
        // Повертаємо результат з вихідної змінної
        $query = $this->pdo->query("SELECT $outParam");
        $result = $query->fetch(\PDO::FETCH_ASSOC);
        // Повертаємо значення вихідного параметра як результат
        return (bool) current($result);
    }
    public function lastInsertId()
    {
        return $this->pdo->lastInsertId();
    }
}

```

Error.php

```

<?php
namespace core;
class Error
{
    public $code;
    public $message;
    public function __construct($code, $message)
    {
        $this->message = $message;
        $this->code = $code;
    }
    public static function debugLog($data) {
        $output = $data;
        if (is_array($output))
            $output = implode(',', $output);

        echo "<script>console.log('Debug Objects: " . $output . "' );</script>";
    }
}

```

Template.php

```

<?php
namespace core;
class Template
{
    protected $path;
    protected $params;
    public function __construct($path)
    {
        $this->path = $path;
        $this->params = [];
    }
    public function setParam($name, $value) {
        $this->params[$name] = $value;
    }
    public function setParams($params) {
        foreach ($params as $name => $value)
            $this->setParam($name, $value);
    }
    public function getHTML() {
        ob_start();
        extract($this->params);
        include($this->path);
        $html = ob_get_contents();
        ob_end_clean();
        return $html;
    }
}

```

Controller.php

```
<?php
namespace core;
class Controller
{
    protected $viewPath;
    protected $moduleName;
    protected $actionName;
    public function __construct()
    {
        $this->moduleName = \core\Core::getInstance()->app['moduleName'];
        $this->actionName = \core\Core::getInstance()->app['actionName'];
        $this->viewPath = "views/{ $this->moduleName}/{ $this->actionName}.php";
    }
    public function render($viewPath = null, $params = null) {
        if (empty($viewPath))
            $viewPath = $this->viewPath;
        $tpl = new Template($viewPath);
        if (!empty($params))
            $tpl->setParams($params);
        return $tpl->getHTML();
    }
    public function renderMessage($type, $message, $redirectUrl = null, $viewParams = [])
    {
        $types = [
            'ok' => 'alert-success',
            'warning' => 'alert-warning',
            'danger' => 'alert-danger'
        ];
        $alertClass = isset($types[$type]) ? $types[$type] : 'alert-info';
        $viewParams['MessageText'] = $message;
        $viewParams['MessageClass'] = $alertClass;
        if ($redirectUrl) {
            header("Location: $redirectUrl");
            exit();
        }
        return $this->render('/layout/message', $viewParams);
    }
    public function error($code, $message = null) {
        return new Error($code, $message);
    }
    public function redirect($url) {
        header("Location: {$url}");
        die;
    }
    public function renderView($viewName) {
        $path = "views/{ $this->moduleName}/{ $viewName}.php";
        $tpl = new Template($path);
        if (!empty($params))
            $tpl->setParams($params);
        return $tpl->getHTML();
    }
}
```

Лістинг Config

database.php

```
<?php
define('DATABASE_HOST','localhost');
define('DATABASE_LOGIN','root');
define('DATABASE_PASSWORD','');
define('DATABASE_BASENAME','jellycake');
```

params.php

```
<?php
$pageParams = [
    'siteName' => 'Інтернет-магазин',
    'title' => 'Головна сторінка',
    'newsCount' => 25
];
```

Лістинг Models

Admin.php

```
<?php
namespace models;
use core\Core;
use core\Utils;
use models\User;
class Admin
{
    protected static $tableName = 'users';
    public static function getAllUsers()
    {
        $users = Core::getInstance()->db->select(self::$tableName, '*');
        return $users;
    }
    public static function deleteUser($userId)
    {
        Core::getInstance()->db->call('deleteUser', [$userId]);
    }
    public static function updateUser($id, $updatesArray)
    {
        $updatesArray = Utils::filterArray($updatesArray, ['id', 'login', 'lastname', 'firstname', 'access', 'address']);
        $updatesArray['id'] = $id;
        Core::getInstance()->db->update(self::$tableName, $updatesArray, ['id' => $id]);
    }
    public static function updateOrder($orderId, $updatesArray)
    {
        $updatesArray = Utils::filterArray($updatesArray, ['status', 'service', 'date', 'address']);
        Core::getInstance()->db->update('orders', $updatesArray, ['id' => $orderId]);
    }
    public static function userExists($userId)
    {
        $user = Core::getInstance()->db->select(self::$tableName, '*', ['id' => $userId]);

        return !empty($user);
    }
}
```

Blog.php

```
class Blog
{
    protected static $tableName = 'blog';

    public static function addBlog($name,$text, $photoPath)
    {
        do {
            $fileName = uniqid() . '.jpg';
            $newPath = "files/blog/($fileName)";
        } while (file_exists($newPath));
        move_uploaded_file($photoPath, $newPath);
        Core::getInstance()->db->insert(self::$tableName, [
```

```

        'name' => $name,
        'photo' => $fileName,
        'text' => $text
    ]);
}

public static function changePhoto($id, $newPhoto)
{
    $row = self::getBlogById($id);
    $photoPath = 'files/blog/' . $row['photo'];
    if (is_file($photoPath))
        unlink($photoPath);
    do {
        $fileName = $id.'_'.uniqid().''.jpg';
        $newPath = "files/blog/($fileName)";
    } while (file_exists($newPath));
    move_uploaded_file($newPhoto, $newPath);
    Core::getInstance()->db->update(self::$tableName, [
        'photo' => $fileName
    ], [
        'id' => $id
    ]);
}

public static function getBlogById($id)
{
    $rows = Core::getInstance()->db->select(self::$tableName, '*', [
        'id' => $id
    ]);
    if (!empty($rows))
        return $rows[0];
    else
        return null;
}

public static function deleteBlog($id)
{
    Core::getInstance()->db->delete(self::$tableName, [
        'id' => $id
    ]);
}

public static function updateBlog($id, $newName, $text)
{
    Core::getInstance()->db->update(self::$tableName, [
        'name' => $newName,
        'text' => $text
    ], [
        'id' => $id
    ]);
}

public static function getBlog()
{
    $rows = Core::getInstance()->db->select(self::$tableName);
    return $rows;
}
}

```

Cart.php

```

class Cart
{
    protected static $tableName = 'carts';

    public static function addToCart($product_id, $quantity, $user_id)
    {
        Core::getInstance()->db->insert(self::$tableName, [
            'product id' => $product_id,
            'quantity' => $quantity,
            'user_id' => $user_id
        ]);
    }

    public static function getCartItemsByUserId($user_id)
    {
        return Core::getInstance()->db->select(self::$tableName, '*', ['user_id' => $user_id]);
    }
}

```

```

    }
    public static function getCartItemsByOrderId($order_id)
    {
        return Core::getInstance()->db->select('purchased_carts', '*', ['order_id' => $order_id]);
    }

    public static function updateCartItemQuantity($user_id, $product_id, $quantity)
    {
        Core::getInstance()->db->update(self::$tableName, ['quantity' => $quantity], ['user_id' =>
$user_id, 'product id' => $product_id]);
    }
    public static function removeCartItem($user_id,$product_id)
    {
        Core::getInstance()->db->delete(self::$tableName, [
            'product id' => $product_id,
            'user_id' => $user_id
        ]);
    }
}

```

Category.php

```

<?php
namespace models;
use core\Core;
class Category
{
    protected static $tableName = 'categories';

    public static function addCategory($name)
    {
        Core::getInstance()->db->insert(self::$tableName, [
            'name' => $name,
        ]);
    }

    public static function getCategoryById($id)
    {
        $rows = \core\Core::getInstance()->db->select(self::$tableName, '*', [
            'id' => $id
        ]);
        if (!empty($rows))
            return $rows[0];
        else
            return null;
    }

    public static function deleteCategory($id)
    {
        Core::getInstance()->db->delete(self::$tableName, [
            'id' => $id
        ]);
    }

    public static function updateCategory($id, $newName)
    {
        Core::getInstance()->db->update(self::$tableName, [
            'name' => $newName
        ], [
            'id' => $id
        ]);
    }

    public static function getCategories()
    {
        $rows = \core\Core::getInstance()->db->select(self::$tableName);
        return $rows;
    }
}

```

Chat.php

```

<?php
namespace models;
use core\Core;
class Chat
{

```

```

        protected static $tableName = 'chat';

        public static function getAllChat()
        {
            return Core::getInstance()->db->select(self::$tableName, '*');
        }

        public static function addChat($message, $user id)
        {
            Core::getInstance()->db->insert(self::$tableName, [
                'user_id' => $user id,
                'message' => $message
            ]);
        }
    }
}

```

Comments.php

```

<?php
namespace models;
use core\Core;
class Comments
{
    protected static $tableName = 'comments';
    public static function createComment($message, $post id, $user id)
    {
        Core::getInstance()->db->insert(self::$tableName, [
            'user_id' => $user id,
            'post id' => $post id,
            'message' => $message
        ]);
    }
    public static function editComment($comment_id, $message)
    {
        Core::getInstance()->db->update(self::$tableName, ['message' => $message], ['id' =>
        $comment id,]);
    }

    public static function deleteComment($comment_id)
    {
        Core::getInstance()->db->delete(self::$tableName, ['id' => $comment_id]);
    }

    public static function GetAllComments()
    {
        return Core::getInstance()->db->select(self::$tableName, '*');
    }
}

```

Order.php

```

<?php
namespace models;
use core\Core;
use core\Utils;
class Order
{
    protected static $tableName = '`orders`';
    public static function findByUniqueId($uniqueId)
    {
        return Core::getInstance()->db->select(self::$tableName, '*', ['unique id' => $uniqueId]);
    }
    public static function updateOrder($orderId, $updatesArray)
    {
        $updatesArray = Utils::filterArray($updatesArray, ['status', 'logistics', 'date',
        'address']);
        $updatesArray['id'] = $orderId;
        Core::getInstance()->db->update(self::$tableName, $updatesArray, ['id' => $orderId]);
    }
    public static function getOrdersByUserId($userId)
    {
        return Core::getInstance()->db->select(self::$tableName, '*', [
            'user_id' => $userId,
        ]);
    }
    public static function deleteOrder($orderId)
    {
    }
}

```

```

        Core::getInstance()->db->delete(self::$tableName, ['id' => $orderId]);
    }
    public static function getOrderById($orderId)
    {
        $rows = Core::getInstance()->db->select(self::$tableName, '*', [
            'id' => $orderId
        ]);
        if (!empty($rows)) {
            return $rows;
        } else {
            return null;
        }
    }
}
public static function createOrder($user id, $status, $service, $address,$phone)
{
    Core::getInstance()->db->insert(self::$tableName, [
        'user_id' => $user_id,
        'status' => $status,
        'service' => $service,
        'address' => $address,
        'phone' => $phone
    ]);
}
public static function getAllOrders()
{
    $orders = Core::getInstance()->db->select(self::$tableName, '*');
    return $orders;
}
public static function updateOrderStatus($orderId, $string)
{
    Core::getInstance()->db->update(self::$tableName,
    ['status'=>$string],['unique id'=>$orderId]);
}
public static function getCompletedOrders()
{
    return Core::getInstance()->db->select('`completed_orders`', '*');
}
}

```

Page.php

```

<?php
namespace models;
class Page
{
    public function display()
    {
        echo 'Page Display';
    }
}

```

Product.php

```

<?php
namespace models;
use core\Utils;
use core\Core;
use Imagick;

class Product
{
    protected static $tableName = 'products';
    public static function changePhoto($id, $newPhoto)
    {
        $row = self::getTovarById($id);
        if(!file_exists($newPhoto)) {
            $photoPath = 'files/product/' . $row['photo'];
            if (is_file($photoPath))
                unlink($photoPath);
        }
        do {
            $fileName = $id.'_'.uniqid().' .jpg';
            $newPath = "files/product/{ $fileName}";
        } while (file_exists($newPath));
        move_uploaded_file($newPhoto, $newPath);
        Core::getInstance()->db->update(self::$tableName, [
            'photo' => $fileName
        ], [

```

```

        'id' => $id
    });
}
public static function searchTovarsByName($title)
{
    return Core::getInstance()->db->select(self::$tableName, '*', ['title[~]' => $title]);
}

public static function AddTovar($row)
{
    $user = User::GetCurrentUser();
    if ($user == null) {
        $result = [
            'error' => true,
            'messages' => ['Користувач не аутентифікований']
        ];
        return $result;
    }
    $validateResult = Product::Validate($row);
    if (is_array($validateResult)) {
        $result = [
            'error' => true,
            'messages' => $validateResult
        ];
        return $result;
    }
    $fields = ['title', 'description', 'price', 'category_id'];
    $rowFiltered = Utils::filterArray($row, $fields);
    $rowFiltered['datetime'] = date('Y-m-d H:i:s');
    $rowFiltered['user_id'] = $user['id'];
    Core::getInstance()->db->insert(self::$tableName, $rowFiltered);
    $id = $rowFiltered;
    return [
        'error' => false,
        'id' => $id
    ];
}

public static function calculateTotalAmount($cartItems)
{
    $totalAmount = 0;
    foreach ($cartItems as $item) {
        $product = Product::getTovarById($item['product_id']);
        $totalAmount += $product['price'] * $item['quantity'];
    }
    return $totalAmount;
}

public static function GetLastTovar($count)
{
    return Core::getInstance()->db->select(self::$tableName, '*', null, ['datetime'=>'DESC'],
$count);
}

public static function GetTovarById($id)
{
    $tovar = Core::getInstance()->db->select(self::$tableName, '*', ['id'=>$id]);
    if (!empty($tovar))
        return $tovar[0];
    else
        return null;
}

public static function UpdateTovar($row, $id)
{
    $user = User::GetCurrentUser();
    if ($user == null)
        return false;
    $validateResult = Product::Validate($row);
    if (is_array($validateResult))
        return $validateResult;
    $fields = ['title', 'description', 'price', 'photo', 'category_id'];
    $rowFiltered = Utils::filterArray($row, $fields);
    $rowFiltered['datetime lastedit'] = date('Y-m-d H:i:s');
    Core::getInstance()->db->update(self::$tableName, $rowFiltered, ['id' => $id]);
    return true;
}

public static function Validate($formrow)
{

```

```

        $errors = [];
        if (empty($formrow['title']))
            $errors [] = 'Поле "Заголовок" не може бути порожнім!';
        if (empty($formrow['description']))
            $errors [] = 'Поле "Короткий опис" не може бути порожнім!';
        if (empty($formrow['price']))
            $errors [] = 'Поле "Ціна" не може бути порожнім!';
        if(count($errors)>0)
            return $errors;
        else
            return true;
    }
    public static function getProducts()
    {
        $rows = Core::getInstance()->db->select(self::$tableName);
        return $rows;
    }
}

```

User.php

```

<?php
namespace models;
use core\Utils;
use core\Core;
class User
{
    protected static $tableName = 'users';
    public static function Validate($formrow)
    {
        $errors = [];
        if (empty($formrow['login']))
            $errors [] = 'Поле "Логін" не може бути порожнім!';
        $user = User::getUserByLogin($formrow['login']);
        if (!empty($user))
            $errors [] = 'Користувач із заданим логіном вже існує!';
        if (empty($formrow['password']))
            $errors [] = 'Поле "Пароль" не може бути порожнім!';
        if ($formrow['password'] != $formrow['password2'])
            $errors [] = 'Обидва паролі повинні збігатись!';
        if (empty($formrow['firstname']))
            $errors [] = 'Поле "Ім\''я" не може бути порожнім!';
        if (empty($formrow['lastname']))
            $errors [] = 'Поле "Прізвище" не може бути порожнім!';
        if (empty($formrow['address']))
            $errors [] = 'Поле "Адреса" не може бути порожнім!';
        if(count($errors)>0)
            return $errors;
        else
            return true;
    }
    public static function isLoginExists($login) {
        $user = Core::getInstance()->db->select(self::$tableName, '*', [
            'login' => $login
        ]);
        return !empty($user);
    }
    public static function logoutUser() {
        unset($_SESSION['user']);
    }
    public static function getUserByLoginAndPassword($login, $password) {
        $user = Core::getInstance()->db->select(self::$tableName, '*', [
            'login' => $login,
            'password' => self::hashPassword($password)
        ]);
        if (!empty($user))
            return $user[0];
        return null;
    }
    public static function authenticateUser($user) {
        $_SESSION['user'] = $user;
    }
    public static function hashPassword($password) {
        return md5($password);
    }
    public static function isUserAuth() {
        return isset($_SESSION['user']);
    }
    public static function IsAuth()
    {

```

```

        return isset($_SESSION['user']);
    }
    public static function IsAdmin()
    {
        $user = self::GetCurrentUser();
        return $user && $user['access'] == 1;
    }
    public static function updateUser($id, $updatesArray)
    {
        $updatesArray = Utils::filterArray($updatesArray, ['login', 'lastname',
'firstname', 'address']);
        Core::getInstance()->db->update(self::$tableName, $updatesArray, ['id' => $id]);
    }
    public static function GetCurrentUser()
    {
        return isset($_SESSION['user']) ? $_SESSION['user'] : null;
    }

    public static function AddUser($userRow)
    {
        $validateResult = self::Validate($userRow);
        if (is_array($validateResult))
            return $validateResult;
        $fields = ['login', 'password', 'firstname', 'lastname', 'address'];
        $userRowFiltered = Utils::filterArray($userRow, $fields);
        $userRowFiltered['password'] = md5($userRowFiltered['password']);
        Core::getInstance()->db->insert(self::$tableName, $userRowFiltered);
        return true;
    }

    public static function AuthUser($login, $password)
    {
        $password = md5($password);
        $users = Core::getInstance()->db->select(self::$tableName, "*",
        [
            'login' => $login,
            'password' => $password,
        ]);
        if (count($users) == 1)
        {
            $user = $users[0];
            return $user;
        } else
            return false;
    }

    public static function GetUserByLogin($login)
    {
        $rows = Core::getInstance()->db->select(self::$tableName, '*', ['login'=> $login]);
        if (count($rows) > 0)
            return $rows[0];
        else
            return null;
    }
    public static function GetUserById($userId)
    {
        $userId = intval($userId);
        $db = Core::getInstance()->db;

        $result = $db->select(self::$tableName, '*', ['id' => $userId]);

        if ($result && count($result) > 0) {
            return $result[0];
        }

        return null;
    }
}

```