

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:

«Сайт для замовлення їжі»

Виконав студент групи ІПЗ-19-3
ЛІСОВИЙ Максим Сергійович

Керівник роботи:
КРАВЧЕНКО Світлана Миколаївна

Рецензент:
ЄФРЕМОВ Юрій Миколайович

Житомир – 2023

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

«ЗАТВЕРДЖУЮ»

В.о. зав. кафедри інженерії
програмного забезпечення
Андрій МОРОЗОВ
«02» серпня 2023 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ЛІСОВИЙ Максим Сергійович**

Керівник роботи: **КРАВЧЕНКО Світлана Миколаївна**

Тема роботи: **«Сайт для замовлення їжі»**,

затверджена Наказом закладу вищої освіти від **«02» серпня 2023 р. №326с**

Вихідні дані для роботи: використання сучасних веб-технологій для розробки веб-сайту для замовлення їжі.

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

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Кравченко С.М.	02.08.2023р.	02.08.2023р.
2	Кравченко С.М.	02.08.2023р.	02.08.2023р.
3	Кравченко С.М.	02.08.2023р.	02.08.2023р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускової кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Аналізування вимог до веб-сайту	2 серпня 2023 – 5 серпня 2023	Виконано
2	Пошук, огляд та аналіз аналогічних сайтів	6 серпня 2023 – 10 серпня 2023	Виконано
3	Формулювання технічного завдання та опрацювання літературних джерел	11 серпня 2023 – 13 серпня 2023	Виконано
4	Проектування структури сайту	14 серпня 2023 – 16 серпня 2023	Виконано
5	Написання програмного коду	17 серпня 2023 – 22 серпня 2023	Виконано
6	Оформлення пояснювальної записки	23 серпня 2023 – 25 серпня 2023	Виконано
7	Попередній захист	25 серпня 2023	Виконано

РЕФЕРАТ

Випускна робота бакалавра складається з веб-сайту для замовлення їжі та пояснювальної записки. Пояснювальна записка до випускної роботи містить 50 сторінок, 20 ілюстрацій та 2 таблиці.

Метою роботи є розробка веб-сайту для замовлення їжі, що містить модулі реалізації оформлення замовлень, редагування, додавання та видалення елементів меню, реєстрація та авторизація.

В роботі визначено основні завдання на створення сайту для замовлення їжі, проаналізовано аналоги розробленого функціоналу, обґрунтовано вибір архітектурного патерну для реалізації завдання - MVC. Наведено сценарії роботи веб-сайту, загальна структура роботи сайту, опис бази даних системи (СУБД MySQL), опис алгоритмів взаємодії окремих модулів системи, опис об'єктної структури системи (діаграма класів), алгоритми роботи основних модулів. Також було проведено та наведено результати тестування роботи сайту.

КЛЮЧОВІ СЛОВА: MVC, МЕНЮ, ОФОРМЛЕННЯ ЗАМОВЛЕННЯ, КОРИСТУВАЧІ, АДМІНІСТРАТОР.

					ПЗ.КР.Б – 121 – 23 – ПЗ			
Зм.	Арк.	№ докум.	Підпис	Дата	Сайт для замовлення їжі Пояснювальна записка	Літ.	Арк.	Аркушів
Розроб.	М. Лісовий						4	
Керівник	С. Кравченко					Житомирська політехніка, група ПЗ-19-3		
Н.Контр.	О. Тичина							
Зав. каф.	А. Морозов							

ABSTRACT

A bachelor's thesis consists of a website for ordering food and an explanatory note. The explanatory note to the final thesis contains 49 pages, 20 illustrations and 2 tables.

The purpose of the work is to develop a website for ordering food, which includes modules for the implementation of ordering, editing, adding and removing menu items, registration and authorization.

The work defines the main tasks for creating a website for ordering food, analyzes the analogues of the developed functionality, and substantiates the choice of an architectural pattern for the implementation of the task - MVC. Website operation scenarios, the general structure of the site operation, a description of the system database (MySQL DBMS), a description of the interaction algorithms of individual system modules, a description of the object structure of the system (class diagram), and the operation algorithms of the main modules are provided. The results of testing the site's operation were also conducted and given.

KEY WORDS: MVC, MENU, ORDERING, USERS, ADMINISTRATOR.

ЗМІСТ

ВСТУП	8
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ ВЕБ-САЙТУ	10
1.1. Постановка задачі	10
1.2. Аналіз аналогів програмного продукту	11
1.3. Вибір архітектури веб-сайту для замовлення їжі.....	11
1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення.....	16
Висновки до розділу 1	18
РОЗДІЛ 2. ПРОЕКТУВАННЯ ВЕБ-САЙТУ ДЛЯ ЗАМОВЛЕННЯ ЇЖІ.....	20
2.1. Визначення функціоналу сайту.....	20
2.2. Розробка бази даних для сайту.....	22
2.3. Проектування та реалізація алгоритмів роботи сайту	27
2.4 Реалізація сайту для замовлення їжі	29
Висновки до розділу 2	34
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБ-САЙТОМ ДЛЯ ЗАМОВЛЕННЯ ЇЖІ	35
3.1. Порядок створення та налаштування сайту.....	35
3.2. Структура інтерфейсу. Інтерфейс та порядок роботи з сайтом для за.	36
3.3. Тестування роботи сайту	36
Висновки до розділу 3	46
ВИСНОВКИ.....	47
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	49
ДОДАТКИ.....	51

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

CRUD – create, read, update, delete (з *англ.* створити, прочитати, оновити, видалити)

БД – база даних

MCM – модуль створення меню

МСЗ – модуль створення замовлення

ERM – entity-relationship model (з *англ.* модель сутність — зв'язок)

MVC – Model–View–Controller (з *англ.* Модель–Представлення–Контролер)

ВСТУП

Актуальність теми. У наш час сайт може бути візитною карткою підприємства чи фірми. Існує купа Інтернет-магазинів, файлообмінників, соціальних мереж, онлайн сервісів, тощо. За допомогою сайту можна робити купу корисних речей, таких як: поширення інформації та її коригування, таким чином, люди, які використовують сайт завжди володіють відповідною інформацією або даними. За допомогою сайту виникає можливість донести величезну кількість інформації до людей.

На сьогоднішній день розвиток інформаційних технологій дозволяє задовольнити різні сфери життя, однією з яких є замовлення харчування. Створення сайту є обов'язковим для компаній в нашому теперішньому часі, де все змінюється щохвилини, враховуючи швидкий темп життя в якому ми живемо. Зокрема ІТ-технології, яку дуже швидко розвиваються кожного дня. Використання веб-технологій постійно зростає, постійно з'являється щось нове в дизайні чи програмуванні. Створення веб-додатку по доставці їжі дуже актуально для нашого часу, адже кожному з нас буде зручніше зробити замовлення через сайт, сидячи на робочому місці або вдома.

Метою випускної роботи є створення Інтернет-сервісу для замовлення їжі, який буде орієнтуватися на можливості користувачів обирати та замовляти їжу з пропонованого меню.

Об'єктом дослідження розробка, проектування та реалізація сайту.

Предметом дослідження є використання сучасних веб-технологій для створення даного сервісу.

В роботі було використано методи об'єктно орієнтованого проектування, теорії реляційних баз даних, веб-програмування, об'єктно-орієнтованого програмування та дизайну інтерфейсів.

Розроблений сайт може бути придатним для роботи та надання користувачам можливості для швидкого та зрозумілого замовлення їжі.

Пояснювальна записка до випускної роботи містить 50 сторінок, 20 ілюстрацій та 2 таблиці, список літературних джерел містить 15 найменування.

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ ВЕБ-САЙТУ

1.1. Постановка задачі

Основною метою створення веб-сайту для замовлення їжі є забезпечення користувачів зручним додатком для створення замовлень обираючи бажане з меню. Користувачі матимуть можливість створити обліковий запис та мати доступ до своїх замовлень, бачити перелік страв та обирати з нього. Адміністрація сайту, в свою чергу, зможе вносити корективи в меню. Адмін сайту зможе додавати, редагувати або видаляти меню з переліку вже існуючих.

Щоб реалізувати поставленні задачі необхідно розробити інформаційний веб-сайт для замовлення їжі, створити базу даних та мати можливість керувати нею, видаляючи або додаючи конкретну інформацію. Розробити вигляду веб-сайту.

Основними етапами при виконанні завдання є:

1. Розробити дизайн кожної сторінки сайту для зручного використання клієнтами.
2. Реалізувати систему управління веб-сайтом. Реалізувати реєстрацію та авторизацію користувачів в системі. Створити ролі користувача та адміністратора для редагування та взаємодії з контентом сайту.
3. Створити базу даних за допомогою реалізованих CRUD операцій для кожної таблиці БД.
4. Реалізувати структуру заповнити бази даних інформацією про користувачів та контент сайту.
5. Розробити план тестування системи на наявність помилок. Зокрема передбачити такі види тестування як: функціональне, кросбраузерне та тестування безпеки даних.

Результатом виконання поставленого завдання є зрозумілий користувачам та простий в використанні веб-додаток, який містить повністю

					ІПЗ.КР.Б – 121 – 23 – ІЗ	
						10

функціональну серверну структуру збереження даних, клієнтську сторінку для реалізації можливостей та вимог сайту, а також засоби контролю та керування доступом.

1.2. Аналіз аналогів програмного продукту

На сьогоднішній день існує безліч сайтів для замовлення чи доставки їжі. Майже кожна компанія створює свій особистий сайт, адже це допомагає в розвитку комерційного бізнесу. Мати можливість замовити бажане зі смартфона лежачи вдома, або з робочого місця в офісі – основна перевага таких сайтів. Перед розробкою даного веб-сайту буде зроблено попередній аналіз сайтів для замовлення їжі. На основі зібраної інформації сформується чітке розуміння всіх недоліків та плюсів конкурентів, а також зроблено висновок.

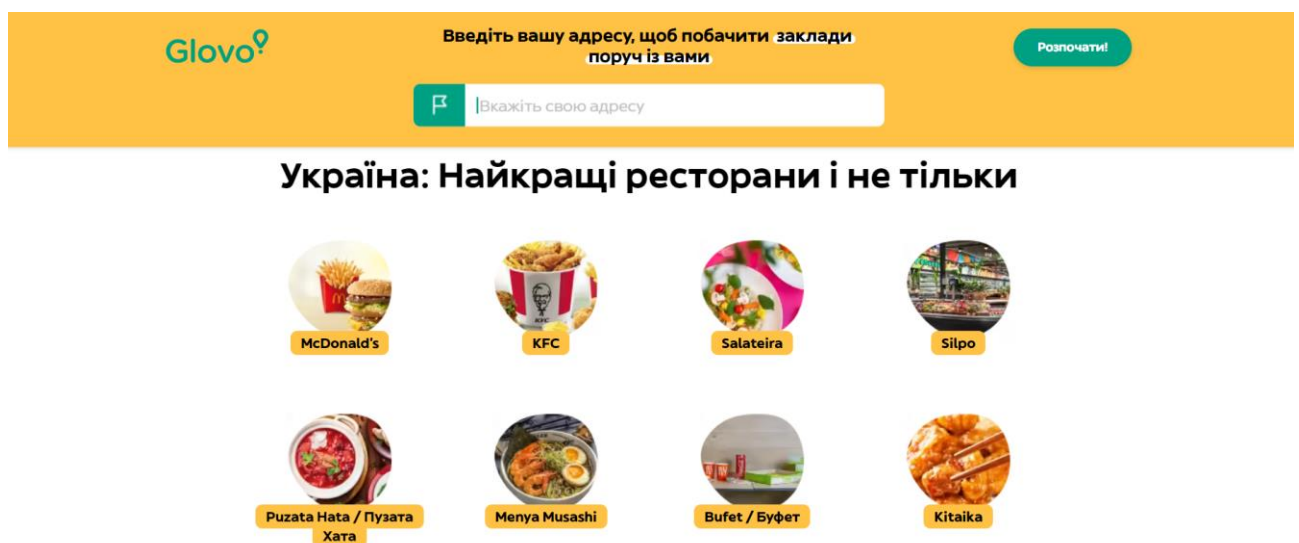


Рис.1.1. Головне меню сайту Glovo.com

Ми бачимо досить зручне меню сайту та зрозумілий інтерфейс. Є можливість обрати їжу відомих брендів та доставити її. Одним з найбільших плюсів сайту є вбудована в нього геолокація, яка дозволяє знайти заклади поруч з користувачем, якщо той надасть інформацію стосовно своєї адреси. Мінусом сайту є його відсутність реєстрації користувачів, що не дозволяє мати

власний профіль, де можна побачити та відредагувати як інформацію про себе, так і інформацію стосовно замовлення.

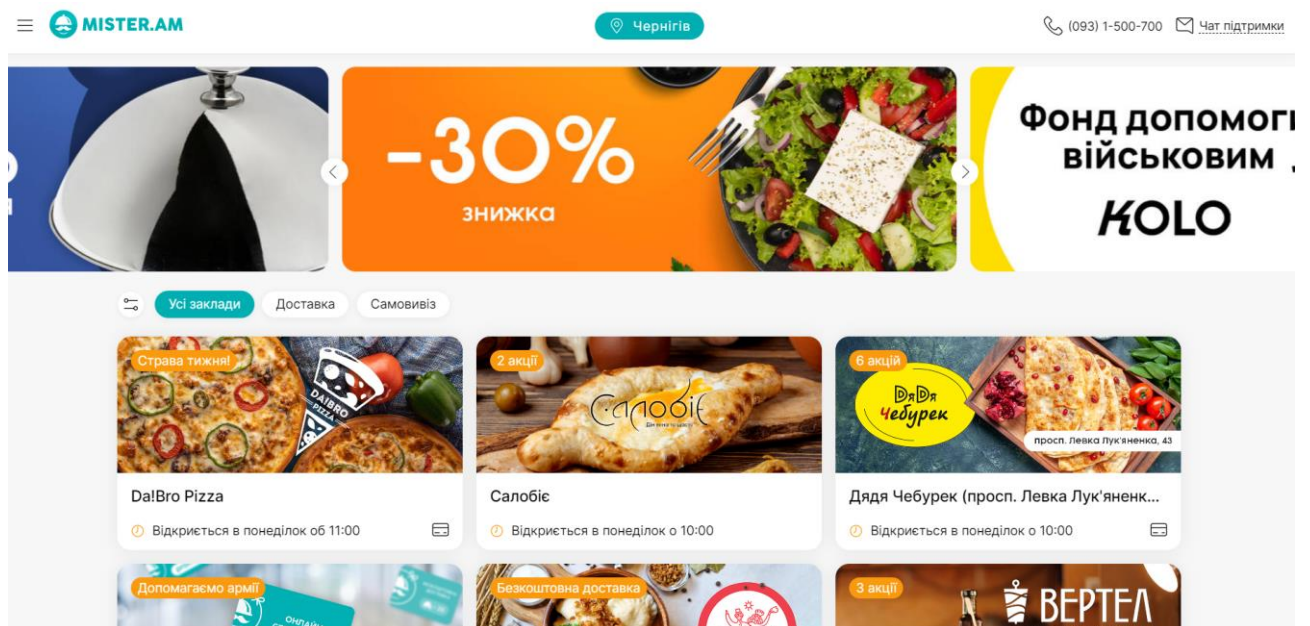


Рис.1.2. Головне меню сайту misteram.com

Цей веб-сайт має багатфункціональне, хоча й досить проблемне для розуміння головне меню. Є можливість обрати бажану страву на вибір та одразу оплатити її. Сайт має систему реєстрації та авторизації, що дає змогу бачити список замовлень у власному профілі. Також є можливість завантаження особистої інформації в профіль, наприклад адресу чи номер телефону. З недоліків можна виділити відсутню функцію пошуку найближчих закладів до користувача.

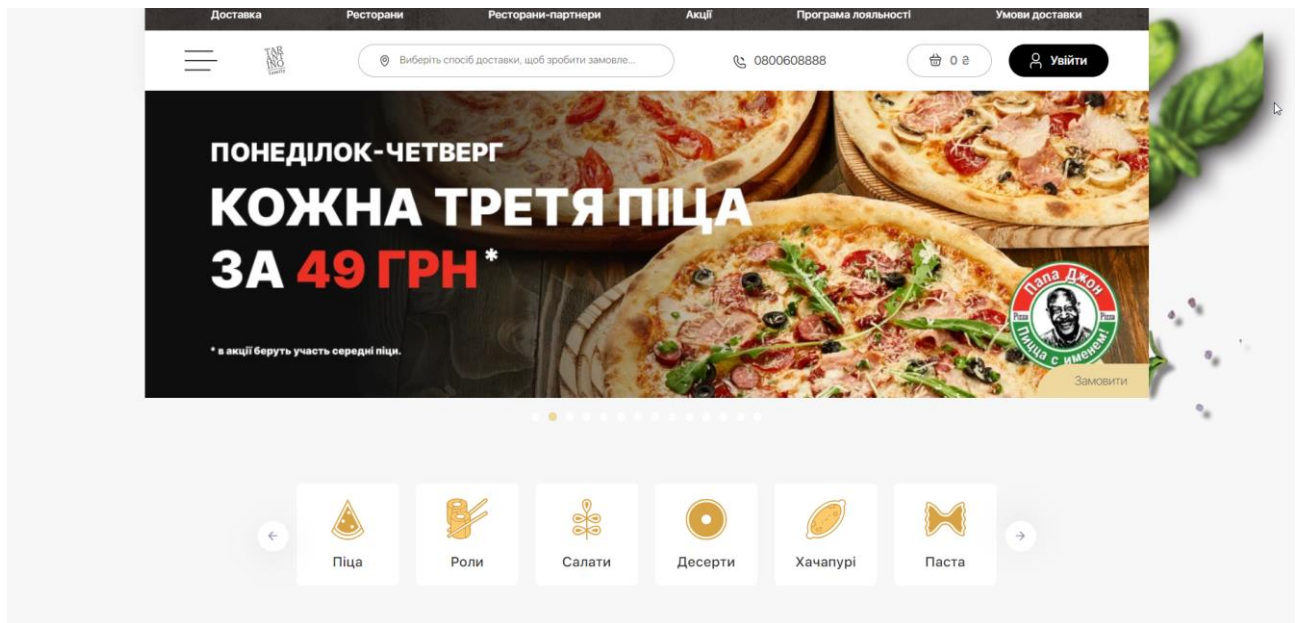


Рис.1.3. Головне меню сайту tarantino-family.com

Даний аналог має зручне у використанні меню та зрозумілий інтерфейс. Є можливість вибрати конкретну категорію їжі, яка цікавить користувачів. Веб-сайт дає можливість створити обліковий запис та користуватися всіма перевагами, такими як, наприклад, надання інформації стосовно адреси та перегляд замовлень. Також тут є система, яка показує заклади їжі біля користувача, за умови що було надано адресу.

Для зручності було створено таблицю, щоб проаналізувати аналоги.

Таблиця 1.1

Порівняння аналогів

Ресурс	Наявність реєстрації користувачів	Зручність інтерфейсу та якісне меню страв	Зрозумілий інтерфейс	Можливість побачити найближчі до користувача заклади харчування
glovo.com	-	+	+	+
misteram.com	+	+	-	-

tarantino-family.com	+	+	+	+
----------------------	---	---	---	---

Після проведеного аналізу можна відокремити та побачити як позитивні так і негативні якості на кожному з сайтів. Цю інформацію було проаналізовано та використано при створенні дипломного проекту.

1.3. Вибір архітектури веб-сайту для замовлення їжі

Для реалізації веб-сайту для замовлення їжі було обрано архітектурний патерн MVC. Інші патерни, які були розглянуті для виконання дипломної роботи, не дають такого функціоналу та гнучкості щодо реалізації поставлених завдань.

Модель–вигляд–контролер (MVC, Модель–представлення–контролер, англ. Model-view-controller) — архітектурний шаблон, який використовується під час проєктування та розробки програмного забезпечення.

Цей шаблон передбачає поділ системи на три взаємопов'язані частини: модель даних, вигляд (інтерфейс користувача) та модуль керування. Застосовується для відокремлення даних (моделі) від інтерфейсу користувача (вигляду) так, щоб зміни інтерфейсу користувача мінімально впливали на роботу з даними, а зміни в моделі даних могли здійснюватися без змін інтерфейсу користувача.

Мета шаблону — гнучкий дизайн програмного забезпечення, який повинен полегшувати подальші зміни чи розширення програм, а також надавати можливість повторного використання окремих компонентів програми. Крім того використання цього шаблону у великих системах сприяє впорядкованості їхньої структури і робить їх більш зрозумілими за рахунок зменшення складності.

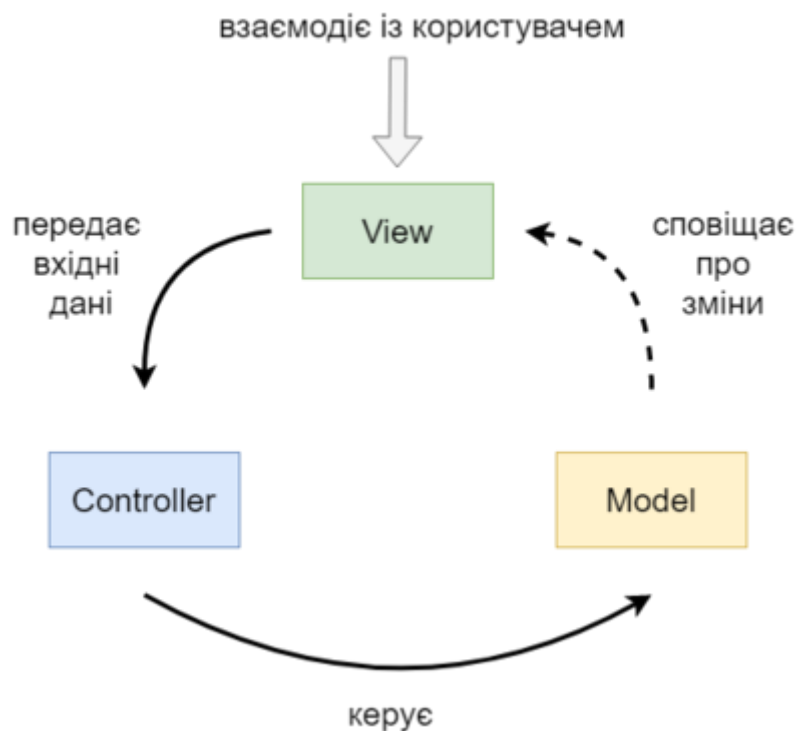


Рис.1.4. Діаграма взаємодії між компонентами шаблону MVC

У рамках архітектурного шаблону модель–вигляд–контролер (MVC) програма поділяється на три окремі, але взаємопов'язані частини з розподілом функцій між компонентами. Модель (Model) відповідає за зберігання даних та їх структуру. Вигляд (View) відповідальний за представлення цих даних користувачеві, тобто інтерфейс програми. Контролер (Controller) керує компонентами, отримує сигнали у вигляді реакції на дії користувача (зміна положення курсора миші, натискання кнопки, ввід даних в текстове поле) і передає дані у модель.

Модель є центральним компонентом шаблону MVC і відображає поведінку застосунку, незалежну від інтерфейсу користувача. Модель стосується прямого керування даними, логікою та правилами застосунку.

- Активна модель - вигляд відстежує зміни в моделі та реагує на них.
- Пасивна модель - вигляд оновлюється через контролер.
- Вигляд може являти собою будь-яке представлення інформації, одержуване на виході, наприклад графік чи діаграму. Одночасно можуть співіснувати кілька виглядів (представлень) однієї і тієї ж

інформації, наприклад гістограма для керівництва компанії й таблиці для бухгалтерії.

- Контролер отримує вхідні дані й перетворює їх на команди для моделі чи вигляду.

Модель інкапсулює ядро даних і основний функціонал їхньої обробки, і не залежить від процесу вводу чи виводу даних.

Вигляд може мати декілька взаємопов'язаних областей, наприклад різні таблиці і поля форм, в яких відображаються дані.

У функції контролера входить відстеження визначених подій, що виникають в результаті дій користувача. Контролер дозволяє структурувати код шляхом групування пов'язаних дій в окремий клас. Наприклад у типовому MVC-проекті може бути користувацький контролер, що містить групу методів, пов'язаних з управлінням обліковим записом користувача, таких як реєстрація, авторизація, редагування профілю та зміна пароля.

Зареєстровані події транслуються в різні запити, що спрямовуються компонентам моделі або об'єктам, відповідальним за відображення даних. Відокремлення моделі від вигляду даних дозволяє незалежно використовувати різні компоненти для відображення інформації. Таким чином, якщо користувач через контролер внесе зміни до моделі даних, то інформація, подана одним або декількома візуальними компонентами, буде автоматично відкоригована відповідно до змін, що відбулися.

1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Для написання програмного продукту були використані: мова програмування C#, JavaScript, HTML та CSS, SQL Server Managment Studio для створення та керування базою даних, Identity Framework для створення та налаштування форм реєстрації та авторизації, CRUD операції для створення таблиць БД. Було обрано Microsoft Visual Studio для реалізації програмного продукту з використанням вищеперерахованих інструментів розробки.

C# – об'єктно-орієнтована мова програмування з безпечною системою типізації для платформи .NET. Розроблена Андерсом Гейлсбергом, Скотом Вілтамутом та Пітером Гольде під егідою Microsoft Research (належить Microsoft). Синтаксис C# близький до C++ і Java. Мова має строгу статичну типізацію, підтримує поліморфізм, перевантаження операторів, вказівники на функції-члени класів, атрибути, події, властивості, винятки, коментарі у форматі XML. Перейнявши багато від своїх попередників — мов C++, Object Pascal і Smalltalk — C#, спираючись на практику їхнього використання, виключає деякі моделі, що зарекомендували себе як проблематичні при розробці програмних систем, наприклад, мова C#, на відміну від C++, не передбачає множинне успадкування класів.

JavaScript – динамічна, об'єктно-орієнтована прототипна мова програмування. Реалізація стандарту ECMAScript. Найчастіше використовується для створення сценаріїв вебсторінок, що надає можливість на боці клієнта (пристрої кінцевого користувача) взаємодіяти з користувачем, керувати браузером, асинхронно обмінюватися даними з сервером, змінювати структуру та зовнішній вигляд вебсторінки. JavaScript класифікують як прототипну (підмножина об'єктно-орієнтованої), скриптову мову програмування з динамічною типізацією.

HTML (Hypertext Markup Language) – також відома в ІТ сфері як мова розмітки гіпертекстових документів, по факту є набором спеціальних елементів - тегів. Вони допомагають структурувати кожну сторінку Web сайту.

CSS (Cascading Style Sheets) – мова за допомогою якої оформлюються усі веб сторінки. Для того, щоб підключити CSS стиль до певного HTML тегу чи сукупності тегів використовуються спеціальні HTML атрибути class чи id.

Microsoft SQL Server Management Studio (SSMS) — це програмний додаток, розроблений корпорацією Майкрософт, який використовується для налаштування, керування та адміністрування всіх компонентів Microsoft SQL Server. Вперше запущений як Microsoft SQL Server 2005, він є наступником Enterprise Manager у SQL 2000. Інструмент містить як редактори сценаріїв, так і графічні інструменти, які працюють з об'єктами та функціями сервера. Головною функцією SSMS є Object Explorer, який дозволяє користувачеві

переглядати, вибирати та діяти з будь-якими об'єктами на сервері. Останні версії SSMS повністю здатні підключатися до будь-якого екземпляра SQL Server Express і керувати ним.

ASP.NET Identity — вбудована в ASP.NET система автентифікації та авторизації. Ця система дозволяє користувачам створювати облікові записи, автентифікуватися, керувати обліковими записами або використовувати для входу на сайт облікові записи зовнішніх провайдерів, таких як Facebook, Google, Microsoft, Twitter та інші..

CRUD – це основні операції, які використовуються для управління даними в більшості вебзастосунків та систем управління інформацією. Ці операції дають змогу працювати з базою даних, тобто створювати, читати, оновлювати та видаляти дані. Це дає змогу швидко створювати застосунки та системи управління інформацією, використовуючи наявні бази даних.

Microsoft Visual Studio – серія продуктів фірми Майкрософт, які містять інтегроване середовище розробки програмного забезпечення та низку інших інструментальних засобів. Ці продукти дають змогу розробляти як консольні програми, так і програми з графічним інтерфейсом, включно з підтримкою технології Windows Forms, а також вебсайти, вебзастосунки, вебслужби як у рідному, так і в керованому кодах для всіх платформ, що підтримуються Microsoft Windows, Windows Mobile, Windows Phone, Windows CE, .NET Framework, .NET Compact Framework та Microsoft Silverlight.

Висновки до розділу 1

Проведений аналіз предметної області дав змогу визначити основні вимоги та напрямки розробки веб-сайту для замовлення їжі. Було відстежено та помічено всі позитивні та негативні сторони аналогічних продуктів на ринку. Зроблено висновок про необхідність реалізувати такий функціонал сайту як:

- Система реєстрації та авторизації користувачів
- Створення ролей для редагування інформації сайту
- Реалізована можливість відвідувати особисту сторінку користувача з наявним списком замовлень
- Реалізація меню сайту зі зрозумілою навігацією

- Можливість вказати адресу користувача для спрощення пошуку найближчих закладів харчування

Було визначено з інструментами розробки та реалізації даного завдання. Побудовано модель та план розробки веб-сайту. Детальний аналіз схожих сайтів замовлення їжі продемонстрував потенціал та можливості програмного продукту.

Проведено аналіз та обґрунтування вибору наступних інструментів реалізації завдання:

- Мови програмування C#, JavaScript, HTML та CSS;
- Вбудована система автентифікації ASP.NET Identity;
- Програмний додаток Microsoft SQL Server Management Studio;
- Операції CRUD;
- Середовище розробки Microsoft Visual Studio;

РОЗДІЛ 2. ПРОЕКТУВАННЯ ВЕБ-САЙТУ ДЛЯ ЗАМОВЛЕННЯ ЇЖИ

2.1. Визначення функціоналу сайту

Веб-сайт створюється з метою надання користувачам зручного та зрозумілого функціоналу, який дає можливість обрати та замовити їжу з переліку меню.

Вимоги користувачів

Зовнішні користувачі отримують доступ до наступного функціоналу:

- Отримання інформації стосовно меню та переліку страв на сайті
- Створення та використання власного облікового запису
- Зміна інформації у власному обліковому записі
- Перегляд особистої сторінки, яка має інформацію стосовно замовлень користувача
- Видалення та редагування власних замовлень
- Отримати інформацію стосовно найближчого закладу харчування, надаючи свою адресу

Внутрішні користувачі (адміністратори сайту) мають доступ до додаткових функцій, таких як:

- Редагування та видалення страв з меню сайту
- Створення нових страв в меню
- Редагувати та мати доступ до інформації БД

Функціональні вимоги:

1. Реєстрація та авторизація користувачів в системі з подальшим присвоєнням йому відповідної ролі(користувач або адмін).
2. Можливість замовити бажану страву та видалити своє замовлення у разі виникнення потреби.
3. Можливість збереження інформації в базі даних з подальшим її використанням у роботі веб-додатку.

					ІПЗ.КР.Б – 121 – 23 – ІПЗ	
						20

4. Відображення вмісту бази даних на сторінках сайту без дефектів та збоїв.
5. Можливість видалення об'єктів з бази даних.

Нефункціональні вимоги:

1. Навчання роботи з системою:
 - інтерфейс повинен бути зручним у використанні та зрозумілим для користувачів;
 - час відповіді системи для звичайних запитів не повинен перевищувати 2 секунд, а для більш складних – 4 секунди;
 - надійність та безпомилкова робота бази даних;
 - середній час безперервної роботи – 20 робочих днів;
 - максимальна норма помилок та дефектів в роботі додатку – 10 помилок на 1000 запитів.
2. Продуктивність. Система повинна одночасно підтримувати 100 користувачів, пов'язаних зі спільною БД.
3. Можливість експлуатації:
 - система повинна мати можливість збільшувати потужності зі збільшенням кількості користувачів, при цьому без втрат у її роботі;
 - оновлення версій та підтримка сайту повинна бути наявною залежно від вподобань та бажань користувачів. Розширення функціоналу також є необхідністю;

Для реалізації даного завдання було обрано технологію MVC та мову програмування C# як основну. Також було використано такі мови програмування як: JavaScript, HTML та CSS.

2.2. Розробка бази даних для сайту

Для створення веб-сайту для замовлення їжі було використано додаток SQL Server Management Studio. В цій програмі було реалізовано базу даних, яка складається з 10 таблиць, що містять у собі дані для роботи веб-додатку. Назва бази даних – DPLM

Таблиця 2.1

Інформація по таблицям бази даних DPLM

Назва таблиці	Призначення
Address	Таблиця адреси
RoleClaims	Можливості ролей
Roles	Таблиця ролей
UserClaims	Можливості юзера
UserLogins	Авторизація користувачів
UserRoles	Таблиця ролі конкретного користувача
Users	Таблиця користувачів
UserTokens	Таблиця підтвердження користувачів
Menus	Таблиця меню
Orders	Таблиця замовлень

SQL Server Management Studio має можливість надати весь необхідний список інструментів, які дають змогу реалізувати поставлені задачі та відповідати всім вимогам даної роботи. БД має 10 таблиць, кожна з яких відіграє свою роль та надає необхідний функціонал. В даному розділі буде описано та продемонстровано кожен з цих таблиць разом з тим, за що вони відповідають.

Далі наведено структуру БД на прикладі діаграми бази даних.

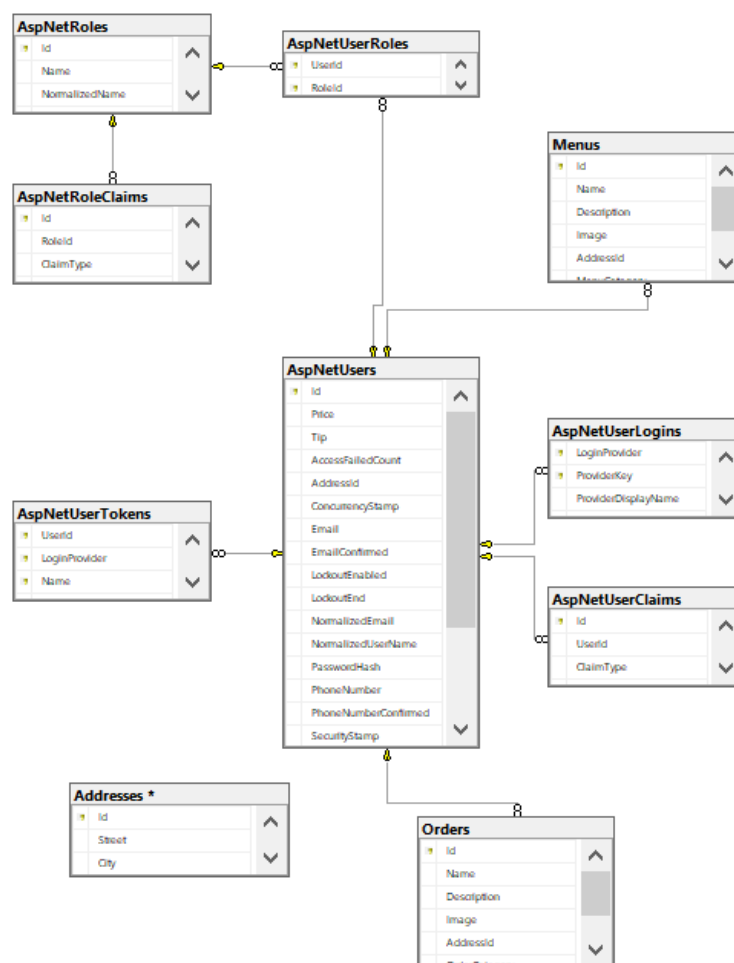


Рис.2.1. Структурна схема БД

Далі наведено опис основних таблиць бази даних.

Таблиця «Address» призначена для збереження адреси користувачів. В дану таблицю вводяться дані про місто та вулицю, такі як назва. Структура таблиці наведена нижче:

Таблиця 2.2

Опис полів таблиці«Address»

Назва	Тип даних	ПК	ЗП	Опис поля
id	Int	+	-	Id
City	nvarchar(MAX)	-	-	Назва міста
Street	nvarchar(MAX)	-	-	Назва вулиці

Таблиця «RoleClaims» призначена для перевірки можливості ролей. В дану таблицю вводяться дані про роль, такі як Id ролі. Структура таблиці наведена нижче:

Таблиця 2.3

Опис полів таблиці «RoleClaims»

Назва	Тип даних	ПК	ЗП	Опис поля
id	Int	+	-	Id користувача
RoleId	nvarchar(450)	-	-	Id ролі
ClaimType	nvarchar(MAX)	-	-	Тип ролі користувача
ClaimValue	nvarchar(MAX)	-	-	Значення ролі користувача

Таблиця «Roles» призначена для збереження даних стосовно ролей. В дану таблицю вводяться дані ролей сайту. Структура таблиці наведена нижче:

Таблиця 2.4

Опис полів таблиці «Roles»

Назва	Тип даних	ПК	ЗП	Опис поля
id	Int	+	-	Id ролі
Name	nvarchar(256)	-	-	Назва ролі
NormalizedName	nvarchar(256)	-	-	Нормалізація імені ролі
ConcurrencyStamp	nvarchar(MAX)	-	-	

Таблиця «UserClaims» призначена для знадання інформації стосовно можливості користувача. Структура таблиці наведена нижче:

Таблиця 2.5

Опис полів таблиці «UserClaims»

Назва	Тип даних	ПК	ЗП	Опис поля
id	Int	+	-	Id
UserId	nvarchar(450)	-	-	Id користувача
ClaimType	nvarchar(MAX)	-	-	Тип ролі
ClaimValue	nvarchar(MAX)	-	-	Значення ролі

Таблиця «UserLogins» призначена для авторизації користувача на сайті. В дану таблицю вводяться дані користувача. Структура таблиці наведена нижче:

Таблиця 2.6

Опис полів таблиці «UserLogins»

Назва	Тип даних	ПК	ЗП	Опис поля
LoginProvider	nvarchar(450)	+	-	Надання входу
ProviderKey	nvarchar(450)	-	-	Надання ключа користувачу
ProviderDisplayName	nvarchar(256)	-	-	Отримання або встановлення ім'я для постачальника
UserId	Int	-	-	Id користувача

Таблиця «UserRoles» призначена для зберігання інформації стосовно ролей конкретного користувача. В дану таблицю вводяться дані користувача та ролі. Структура таблиці наведена нижче:

Таблиця 2.7

Опис полів таблиці «UserRoles»

Назва	Тип даних	ПК	ЗП	Опис поля
UserId	nvarchar(450)	+	-	Id користувача
RoleId	nvarchar(450)	-	-	Id ролі

Таблиця «Users» призначена для зберігання користувачів та всієї інформації. В дану таблицю вводяться дані користувача. Структура таблиці наведена нижче:

Таблиця 2.8

Опис полів таблиці «Users»

Назва	Тип даних	ПК	ЗП	Опис поля
Id	int	+	-	Id
Price	int	-	-	Ціна замовлень
Tip	int	-	-	Чайові
AddressId	int	-	-	Id адреси

Email	nvarchar(256)	-	-	Електронна пошта
EmailConfirmed	bit	-	-	Підтвердження пошти
Password	nvarchar(MAX)	-	-	Пароль
UserName	nvarchar(MAX)	-	-	Юзернейм користувача
City	nvarchar(MAX)	-	-	Місто
Street	nvarchar(MAX)	-	-	Вулиця
ProfileImgUrl	nvarchar(MAX)	-	-	Посилання на картинку профіля

Таблиця «UserTokens» призначена для перевірки користувача на правильність даних на сайті. В дану таблицю вводяться дані користувача. Структура таблиці наведена нижче:

Таблиця 2.9

Опис полів таблиці «UserTokens»

Назва	Тип даних	ПК	ЗП	Опис поля
UserId	nvarchar(450)	+	-	Id
LoginProvider	nvarchar(450)	-	-	Надання входу
Name	nvarchar(450)	-	-	Ім' користувача
Value	nvarchar(MAX)	-	-	Роль користувача

Таблиця «Menus» призначена для зберігання даних меню та їх обробки. Структура таблиці наведена нижче:

Таблиця 2.10

Опис полів таблиці «Menus»

Назва	Тип даних	ПК	ЗП	Опис поля
Id	int	+	-	Id
Name	nvarchar(MAX)	-	-	Назва меню
Description	nvarchar(MAX)			Опис меню
Image	nvarchar(MAX)			Картинка меню
AddressId	int			Id адреси
MenuCategory	int			Категорія меню
AppUserId	nvarchar(450)			Id коистувача

Таблиця «Orders» призначена для зберігання даних замовлень та їх обробки. Структура таблиці наведена нижче:

Таблиця 2.11

Опис полів таблиці «Orders»

Назва	Тип даних	ПК	ЗП	Опис поля
Id	int	+	-	Id
Name	nvarchar(MAX)	-	-	Назва замовлення
Description	nvarchar(MAX)	-	-	Опис замовлення
Image	nvarchar(MAX)	-	-	Картинка замовлення
AddressId	int	-	-	Id адреси
OrderCategory	int	-	-	Категорія замовлення
AppUserId	nvarchar(450)	-	-	Id користувача

Отже реалізована база даних відповідає вимогам веб-додатку та спроможна виконувати всі поставлені задачі. База даних складається з 10 таблиць. Основними є: Users, Menus, Orders, Address, Roles та UserLogins. Допоміжними є: UserTokens, UserRoles, UserClaims, RoleClaims.

2.3. Проектування та реалізація алгоритмів роботи сайту

Основними модулями системи є модулі оформлення замовлень зі зберіганням даних у базі даних та реєстрація користувача з подальшим використанням функціоналу сайту.

Після завантаження головної сторінки веб-сайту для замовлення їжі користувач може виконати наступне: зареєструватися в системі та передивитися список меню. Щоб оформити замовлення користувачу потрібно створити обліковий запис, лише тоді з'явиться можливість зробити замовлення. Для створення облікового запису потрібно ввести електронну пошту та пароль, після чого підтвердити ці дані в полях авторизації користувачів. Якщо дані пройшли валідацію користувач потрапить на сторінку замовлень та буде мати

доступ до особистого кабінету, в якому буде інформація стосовно його замовлень.

Для оформлення замовлення потрібно обрати бажане меню та заповнити форму. Після цього буде перевірка на валідність даних і в разі успішного проходження перевірки, користувач побачить замовлення в своєму профілі. Тут він зможе його видалити при потребі.

Під час оформлення замовлення користувач вводить назву меню, яке бажає замовити, зазначає адресу тощо. Після проходження валідації даними, які користувач ввів, результат буде відображатися у власному кабінеті користувача.

Після того як користувач ввів та надіслав відповідні дані, система формує запит. Цей запит відсилається до бази даних де відбувається формування відповідної таблиці. Таблиця заповнюється даними та відбувається обробка інформації. Якщо операція була виконана успішно, результат повертається на сторінку користувача.

При реалізації операцій передбачена взаємодія таких контролерів як: AccountController, DashboardController, HomeController, MenuController, OrderController. Основними методами для роботи з веб-додатком є RegisterViewModel – для реєстрації користувачів в системі та збереження інформації в базі даних. LoginViewModel – для перевірки валідності даних введених користувачем та авторизації в системі. CreateOrderViewModel – метод, який надає можливість оформлення замовлення та введення з подальшим збереженням для використання даних в БД. CreateOrderMenuModel – метод, який надає можливість створення меню та введення з подальшим збереженням для використання даних в БД. DashboardViewModel – для відображення даних з БД на сторінці користувача. EditMenuViewModel – для редагування створеного меню. EditOrderViewModel – для редагування замовлення користувачів.

Таким чином було розглянуто алгоритми основних процесів сайту для замовлення їжі та проаналізовано взаємодію класів при оформленні та

створенні відповідних об'єктів БД.

2.4 Реалізація сайту для замовлення їжі

Для реалізації всього вищеописаного функціоналу сайту було створено базу даних в додатку SQL Server Management Studio, яку далі було підключено до проекту в файлі appsettings.json.

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Data Source=DESKTOP-4FP5MAV\\SQLEXPRESS;Initial Catalog=DPLM;Integrated
Security=True;Connect Timeout=30;Encrypt=False;Trust Server Certificate=False;Application Intent=ReadWrite;Multi
Subnet Failover=False"
  }
}
```

Для створення та оновлення бази даних були використані міграції. За створення першочергової БД використовувався файл InitialCreate. Тут було реалізовано оформлення та збереження замовлень з меню в базі даних. Лістинг наведено нижче:

```
using Microsoft.EntityFrameworkCore.Migrations;

#nullable disable

namespace DPLM.Migrations
{
    /// <inheritdoc />
    public partial class InitialCreate : Migration
    {
        /// <inheritdoc />
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.CreateTable(
                name: "Addresses",
                columns: table => new
                {
                    Id = table.Column<int>(type: "int", nullable: false)
                        .Annotation("SqlServer:Identity", "1, 1"),
                    Street = table.Column<string>(type: "nvarchar(max)", nullable: false),
                    City = table.Column<string>(type: "nvarchar(max)", nullable: false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Addresses", x => x.Id);
                });

            migrationBuilder.CreateTable(
                name: "AppUser",
                columns: table => new
                {
                    Id = table.Column<string>(type: "nvarchar(450)", nullable: false),
                    Price = table.Column<int>(type: "int", nullable: true),
                    Tip = table.Column<int>(type: "int", nullable: true),
                    AddressId = table.Column<int>(type: "int", nullable: true)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_AppUser", x => x.Id);
                });
        }
    }
}
```

```

constraints: table =>
{
    table.PrimaryKey("PK_AppUser", x => x.Id);
    table.ForeignKey(
        name: "FK_AppUser_Addresses_AddressId",
        column: x => x.AddressId,
        principalTable: "Addresses",
        principalColumn: "Id");
});

migrationBuilder.CreateTable(
    name: "Menus",
    columns: table => new
    {
        Id = table.Column<int>(type: "int", nullable: false)
            .Annotation("SqlServer:Identity", "1, 1"),
        Name = table.Column<string>(type: "nvarchar(max)", nullable: false),
        Description = table.Column<string>(type: "nvarchar(max)", nullable: false),
        Image = table.Column<string>(type: "nvarchar(max)", nullable: false),
        AddressId = table.Column<int>(type: "int", nullable: false),
        MenuCategory = table.Column<int>(type: "int", nullable: false),
        AppUserId = table.Column<string>(type: "nvarchar(450)", nullable: true)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_Menus", x => x.Id);
        table.ForeignKey(
            name: "FK_Menus_Addresses_AddressId",
            column: x => x.AddressId,
            principalTable: "Addresses",
            principalColumn: "Id",
            onDelete: ReferentialAction.Cascade);
        table.ForeignKey(
            name: "FK_Menus_AppUser_AppUserId",
            column: x => x.AppUserId,
            principalTable: "AppUser",
            principalColumn: "Id");
    });

migrationBuilder.CreateTable(
    name: "Orders",
    columns: table => new
    {
        Id = table.Column<int>(type: "int", nullable: false)
            .Annotation("SqlServer:Identity", "1, 1"),
        Name = table.Column<string>(type: "nvarchar(max)", nullable: false),
        Description = table.Column<string>(type: "nvarchar(max)", nullable: false),
        Image = table.Column<string>(type: "nvarchar(max)", nullable: false),
        AddressId = table.Column<int>(type: "int", nullable: false),
        OrderCategory = table.Column<int>(type: "int", nullable: false),
        AppUserId = table.Column<string>(type: "nvarchar(450)", nullable: true)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_Orders", x => x.Id);
        table.ForeignKey(
            name: "FK_Orders_Addresses_AddressId",
            column: x => x.AddressId,
            principalTable: "Addresses",
            principalColumn: "Id",
            onDelete: ReferentialAction.Cascade);
        table.ForeignKey(
            name: "FK_Orders_AppUser_AppUserId",
            column: x => x.AppUserId,
            principalTable: "AppUser",
            principalColumn: "Id");
    });

```

```

});

migrationBuilder.CreateIndex(
    name: "IX_AppUser_AddressId",
    table: "AppUser",
    column: "AddressId");

migrationBuilder.CreateIndex(
    name: "IX_Menus_AddressId",
    table: "Menus",
    column: "AddressId");

migrationBuilder.CreateIndex(
    name: "IX_Menus_AppUserId",
    table: "Menus",
    column: "AppUserId");

migrationBuilder.CreateIndex(
    name: "IX_Orders_AddressId",
    table: "Orders",
    column: "AddressId");

migrationBuilder.CreateIndex(
    name: "IX_Orders_AppUserId",
    table: "Orders",
    column: "AppUserId");
}

/// <inheritdoc />
protected override void Down(MigrationBuilder migrationBuilder)
{
    migrationBuilder.DropTable(
        name: "Menus");

    migrationBuilder.DropTable(
        name: "Orders");

    migrationBuilder.DropTable(
        name: "AppUser");

    migrationBuilder.DropTable(
        name: "Addresses");
}
}
}

```

За реєстрацію та авторизацію користувачів використовувався файл IdentityC, який дає змогу створити всі необхідні таблиці реєстрації та валідації користувачів в БД. Даний файл також відповідає за призначення та керування ролями користувачів.

Функції для створення меню були описані на створені в файлі MenuRepository. Даний файл надає можливості для створення редагування та видалення меню з бази даних.

Листинг наведено нижче:

```

using DPLM.Data;
using DPLM.Interfaces;

```

```

using DPLM.Models;
using Microsoft.EntityFrameworkCore;

namespace DPLM.Repository
{
    public class MenuRepository : IMenuRepository
    {
        private readonly ApplicationDbContext _context;
        public MenuRepository(ApplicationDbContext context)
        {
            _context = context;
        }
        public bool Add(Menu menu)
        {
            _context.Add(menu);
            return Save();
        }

        public bool Delete(Menu menu)
        {
            _context.Remove(menu);
            return Save();
        }

        public async Task<IEnumerable<Menu>> GetAll()
        {
            return await _context.Menus.ToListAsync();
        }

        public async Task<Menu> GetByIdAsync(int id)
        {
            return await _context.Menus.Include(i => i.Address).FirstOrDefaultAsync(i => i.Id == id);
        }

        public async Task<Menu> GetByIdAsyncNoTracking(int id)
        {
            return await _context.Menus.Include(i => i.Address).AsNoTracking().FirstOrDefaultAsync(i => i.Id == id);
        }

        public bool Save()
        {
            var saved = _context.SaveChanges();
            return saved > 0 ? true : false;
        }

        public bool Update(Menu menu)
        {
            _context.Update(menu);
            return Save();
        }
    }
}

```

За функціонал замовлень, їх створення, редагування та видалення, відповідає аналогічний файл OrderRepository. Лістинг файлу є аналогічним до вищенаведеного прикладу.

Файл DashRepository надає змогу розміщення та збереження інформації на сторінці користувача, її редагування та видалення. Також є можливість

оформлення замовлень для звичайних користувачів, та робота з меню для адміністраторів сайту(редагування, додавання чи видалення меню з БД)

Лістинг наведено нижче:

```
using DPLM.Data;
using DPLM.Interfaces;
using DPLM.Models;

namespace DPLM.Repository
{
    public class DashboardRepository : IDashboardRepository
    {
        private readonly ApplicationDbContext _context;
        private readonly IHttpContextAccessor _httpContextAccessor;

        public DashboardRepository(ApplicationDbContext context, IHttpContextAccessor httpContextAccessor)
        {
            _context = context;
            _httpContextAccessor = httpContextAccessor;
        }

        public async Task<List<Menu>> GetAllUserMenus()
        {
            var curUser = _httpContextAccessor.HttpContext?.User.GetUserId();
            var userMenus = _context.Menus.Where(r => r.AppUser.Id == curUser);
            return userMenus.ToList();
        }

        public async Task<List<Order>> GetAllUserOrders()
        {
            var curUser = _httpContextAccessor.HttpContext?.User.GetUserId();
            var userOrders = _context.Orders.Where(r => r.AppUser.Id == curUser);
            return userOrders.ToList();
        }

        public Task<AppUser> GetByIdNoTracking(string id)
        {
            throw new NotImplementedException();
        }

        public async Task<AppUser> GetUserById(string id)
        {
            return await _context.Users.FindAsync(id);
        }

        public bool Save()
        {
            throw new NotImplementedException();
        }

        public bool Update(AppUser user)
        {
            throw new NotImplementedException();
        }
    }
}
```

Висновки до розділу 2

Сайт для замовлення їжі реалізує безліч функцій, основними з яких є: оформлення замовлення користувачем зі збереженням для подальшого користування в БД, створення та редагування меню для адміністраторів сайту, створення нових користувачів та надання їм конкретних ролей на даному сайті. Всі ці функції реалізовані в контролерах Account, Home, Menu та Order.

В розділі реалізована та описана процедура створення бази даних, що є необхідним для вимог поточного завдання та виконання поставлених задач. База даних складається з 10 таблиць. Основними є: Users, Menus, Orders, Address, Roles та UserLogins. Допоміжними є: UserTokens, UserRoles, UserClaims, RoleClaims.

Було розглянуто алгоритми основних процесів сайту для замовлення їжі та проаналізовано взаємодію всіх вищеописаних класів під час таких процесів як: оформлення замовлень та створення меню.

В останньому пункті наводиться акцентований розгляд функцій, які безпосередньо впливають на роботу з БД. Основну увагу було приділено функціям збереження даних в таблиці бази даних та їх редагування.

.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБ-САЙТОМ ДЛЯ ЗАМОВЛЕННЯ ЇЖІ

3.1. Порядок створення та налаштування сайту

Сайт для замовлення їжі було написано мовою C# з використанням Visual Studio. При створенні сайту було написано декілька файлів які відповідають за взаємодію між функціями та базу даних. Для написання сайту було використано SQL Server Management Studio, а отже, якщо виникне питання про перенесення сайту на хостинг, необхідно буде переносити базу даних на сервер та оновлювати дані. За бажанням це можна зробити за наступним порядком дій:

- Створити копії бази даних на старому сервері.
- Створити бази даних на новому сервері.
- Імпортувати дані бази даних.
- Налаштувати базу даних за допомогою міграційних файлів, які були описані в попередньому пункті

Для правильної роботи сайту і незалежності від операційної системи користувача, даний веб-сайт потребує наявності сервера. При бажанні комп'ютер користувача може виконувати роль сервера для роботи сайту.

Створений веб-сайт коректно працює на наступних операційних системах: Windows XP, Windows 7, Windows 8, Windows 10 та Windows 11. При бажанні можливе налаштування під Linux подібні системи.

Конфігурація комп'ютера не відіграє ролі в роботі веб-сайту, що дає змогу використовувати створений продукт без обмежень. Крім цього, розроблений веб-додаток не потребує спеціалізованого апаратного забезпечення, додаткових налаштувань чи засобів розгортання крім стандартних для більшості сучасних сайтів.

3.2. Структура інтерфейсу. Інтерфейс та порядок роботи з сайтом для замовлення їжі

При завантаженні стартової сторінки користувач бачить головну сторінку сайту, де наведені новинки та посилання на інші сторінки (рис 3.1).

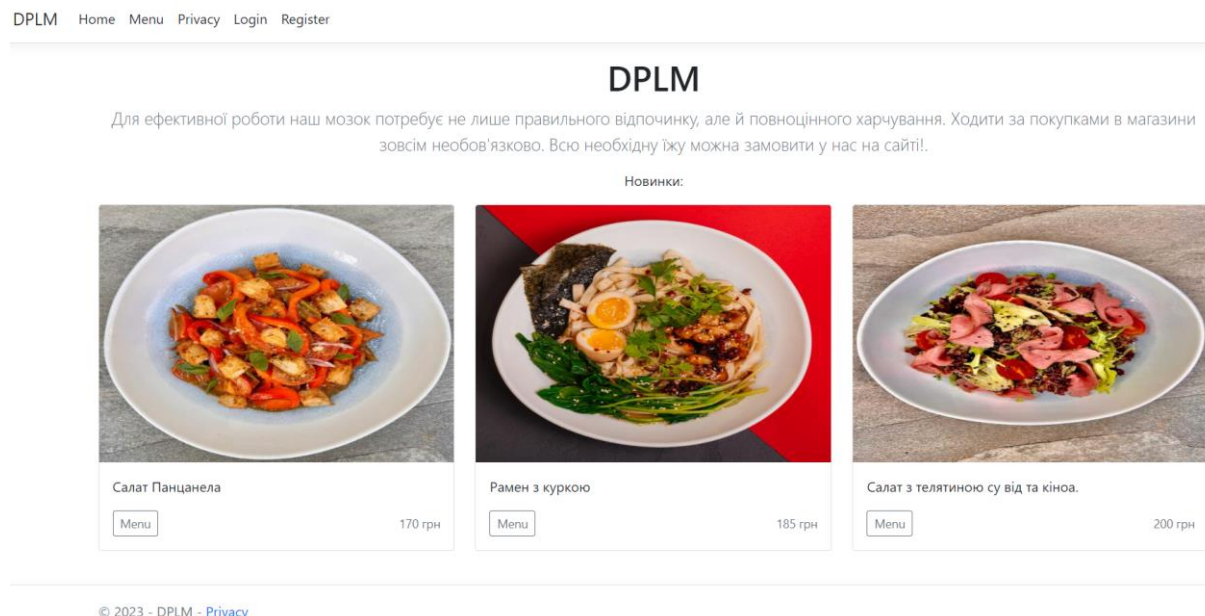


Рис 3.1. Головна сторінка сайту

За допомогою верхнього меню користувач має доступ до більшості сторінок на сайті. Ще 1 сторінка, для оформлення замовлень, з'явиться в меню після створення користувачем облікового запису.

Для того щоб переглянути меню сайту користувач повинен обрати відповідний пункт у верхній частині, або ж перейти за посиланням, яке наведено під новими товарами (рис 3.2).

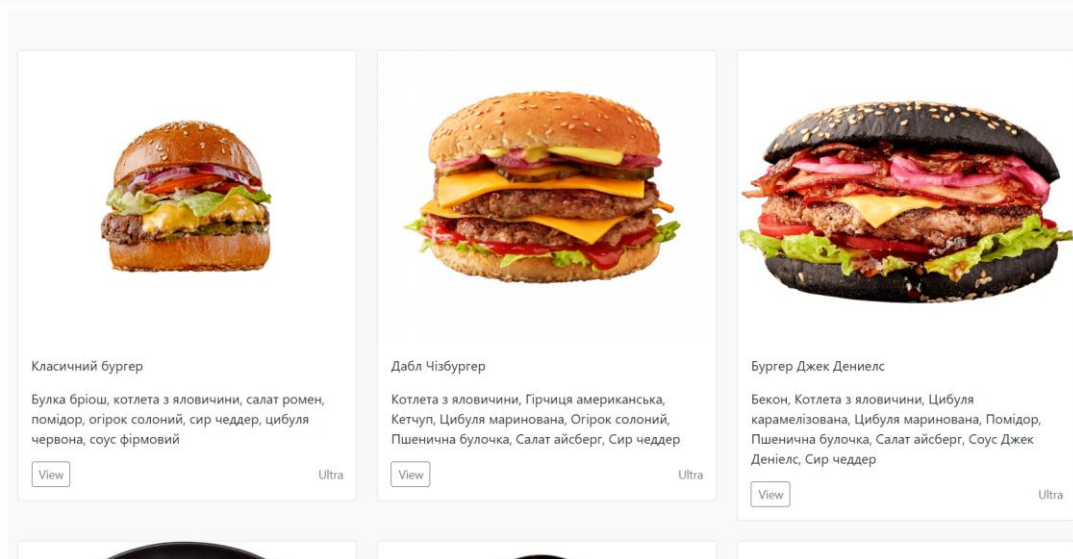


Рис 3.2. Меню сайту

Кожну страву можна детально оглянути, якщо натиснути на кнопку “View”. Користувач потрапить на окрему сторінку де зможе отримати більш точну та якісну інформацію(рис 3.3).

Зручно, зрозуміло та смачно



Класичний бургер

Булка бріош, котлета з яловичини, салат ромен, помідор, огірок солоний, сир чеддер, цибуля червона, соус фірмовий

Details

- Об'єм: 320 г
- Ціна: 200 грн
- Ultra

Рис 3.3. Сторінка “View”.

Щоб оформити замовлення користувач повинен створити обліковий запис нажавши на кнопку “Register” (рис 3.4).

Sign up for a new account

Email Address
test12@gmail.com

Password

Confirm Password

Sign up Cancel

Рис 3.4. Сторінка реєстрації

Після проходження валідації даних потрібно увійти до свого облікового запису натиснувши кнопку “Login”(рис 3.5).

Log in to your account

Email Address
test12@gmail.com

Password

Log in Cancel

Рис 3.5. Користувач входить в обліковий запис

На верхній панелі з’явилося посилання на сторінку “Order”, що дозволяє оформити замовлення, а також кнопка “Logout” для виходу з облікового запису користувачем(рис 3.6).

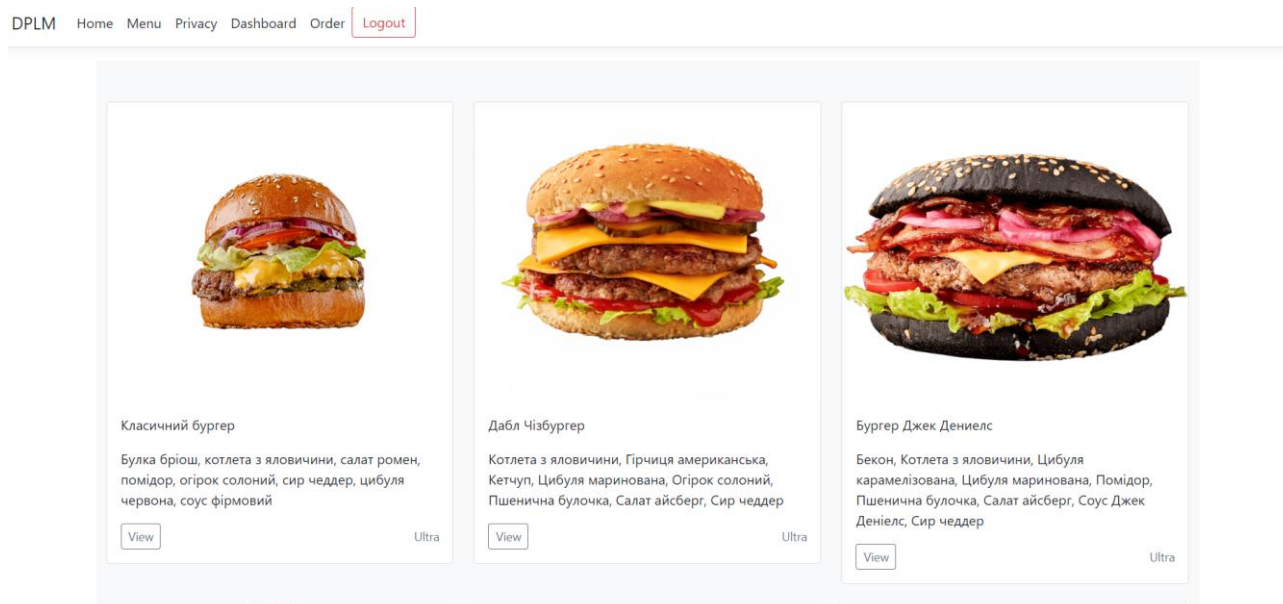


Рис 3.6. Зміни на панелі посилань

Натиснувши на “Order” користувач потрапляю на список страв, які можна замовити на даний момент(рис 3.7)

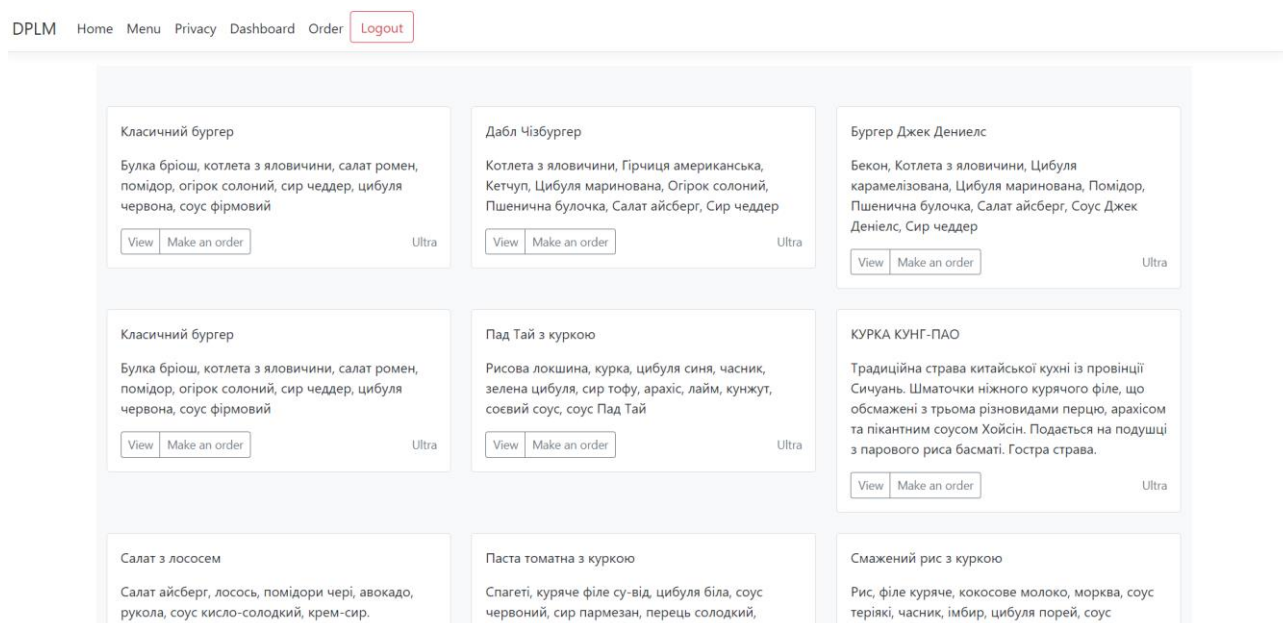


Рис 3.7. Сторінка “Order”

Кожну страву можна оглянути задля отримання детальної інформації, натиснувши кнопку “View” аналогічно описаному вище методу в меню.

Щоб зробити замовлення потрібно натиснути “Make an order” та заповнити відповідні поля адреси в формі. Після цього натиснути “Submot”(рис 3.8).

Title
Класичний бургер

Description
Булка бриш, котлета з яловичини, салат ромен, помідор, огірок солоний, сир чеддер, цибуля червона, соус фірмовий

Image
<https://assets.dots.live/misteram-public/5e98e4acfeac3361e556e97701fbd1b0-826x0.png>

Street
Чуднівська 50

City
Житомир

[Submit](#)

Рис 3.8. Оформлення замовлення

По завершенню користувача переправить на сторінку “Order” для здійснення подальших замовлень.

Створений веб-сайт також має функцію адміністратора, який має більше привілеїв ніж звичайний користувач. Наприклад, адміністратор може додавати, змінювати чи видаляти страви з меню. Щоб отримати таку можливість потрібно увійти в обліковий запис який має роль “admin”(рис 3.9).

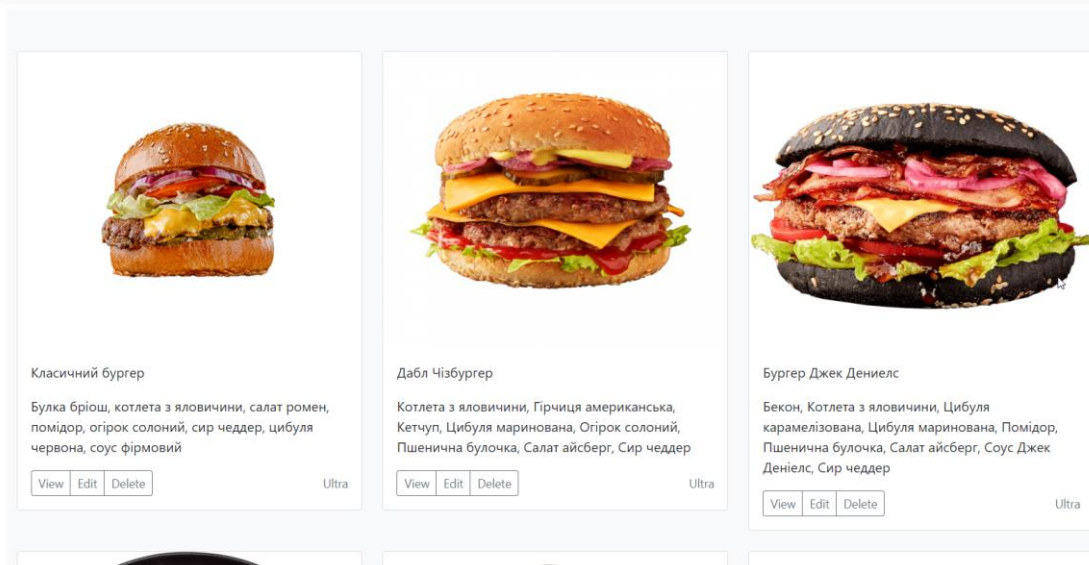


Рис 3.9. Меню адміністратора сайту

Адміністратор може видаляти та редагувати страви меню, а також, за бажанням, додавати туди новий асортимент(рис 3.10).

DPLM

Home

Menu

Privacy

Dashboard

Order

Logout

Are you sure you want to delete this order?

Name

Класичний бургер

Description

Булка бріош, котлета з яловичини, салат ромен, помідор, огірок солоний, сир чеддер, цибуля червона, соус фірмовий

Order Category

Ultra

Street

200 грн

City

320 г

Image

https://assets.dots.live/misteram-public/5e98e4acfeac3361e556e97701fbd1b0-826x0.png

Confirm

© 2023 - DPLM - [Privacy](#)

Рис 3.10. Видалення “Класичного бургера” з меню

Після натискання “Confirm” страва зникає з меню(рис 3.11).

DPLM

Home

Menu

Privacy

Dashboard

Order

Logout



Дабл Чізбургер

Котлета з яловичини, Гірчиця американська, Кетчуп, Цибуля маринована, Огірок солоний, Пшенична булочка, Салат айсберг, Сир чеддер

View Edit Delete

Ultra



Бургер Джек Дениелс

Бекон, Котлета з яловичини, Цибуля карамелізована, Цибуля маринована, Помідор, Пшенична булочка, Салат айсберг, Соус Джек Деніелс, Сир чеддер

View Edit Delete

Ultra



Курка в кисло-солодкому соусі

Стегно куряче, мука пшенична, яйце, соус сливовий, соус солодкий чілі, сіль морська дрібна.

View Edit Delete

Ultra

Рис 3.11. Змінене меню після видалення

Аналогічно відбувається редагування страв, якщо натиснути “Edit”.

ІПЗ.КР.Б – 121 – 23 – ІПЗ |

41

Для створення меню чи запитів замовлень, адміністратор має сторінку “Dashboard”, де буде можливість створювати, а також відстежити що саме було створено даним адміністратором сайту(рис 3.12).

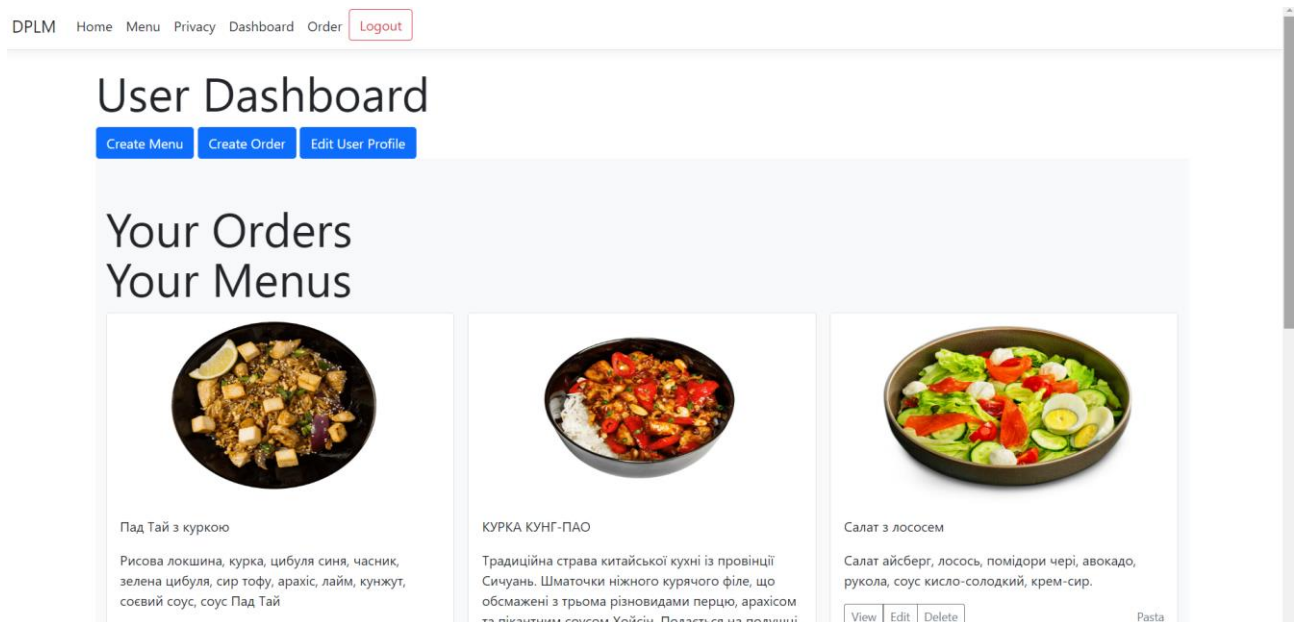


Рис 3.12. Dashboard адміністратора

На малюнку можна побачити список “Your Menus”, який показує страви, що були додані саме цим конкретним адміністратором.

Для створення нової страви в меню чи замовленнях, потрібна натиснути “Create”. Для прикладу буде створено видалений “Класичний бургер”(рис 3.13 та рис 3.14).

Name

Класичний бургер

Description

Котлета з яловичини, Гірчиця американська, Кетчуп, Цибуля маринована, Огірок солоний, Пшенична булочка, Салат айсберг, Сир чеддер

Image

https://assets.dots.live/misteram-public/5e98e4acfeac3361e556e97701fbd1b0-826x0.png

MenuCategory Pizza

Street

300 грн

City

320 г

Submit

Рис 3.13. Створення страви

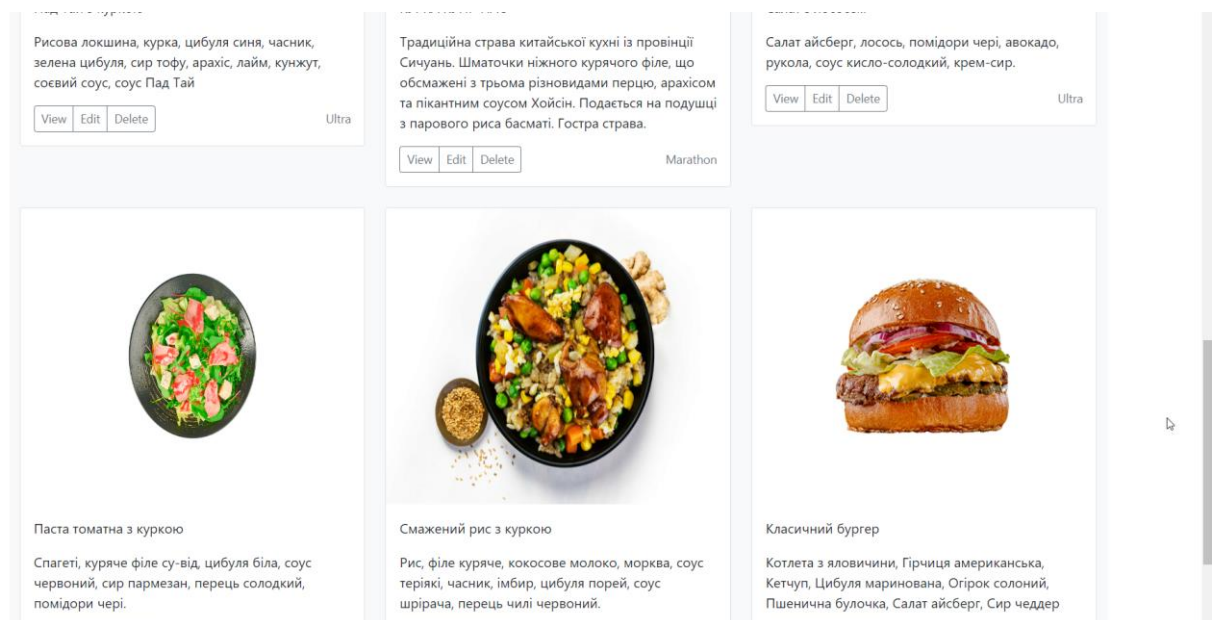


Рис 3.14. Створений бургер

Після створення, відповідний об'єкт з'являється також в Dashboard адміністратора(рис 3.15).

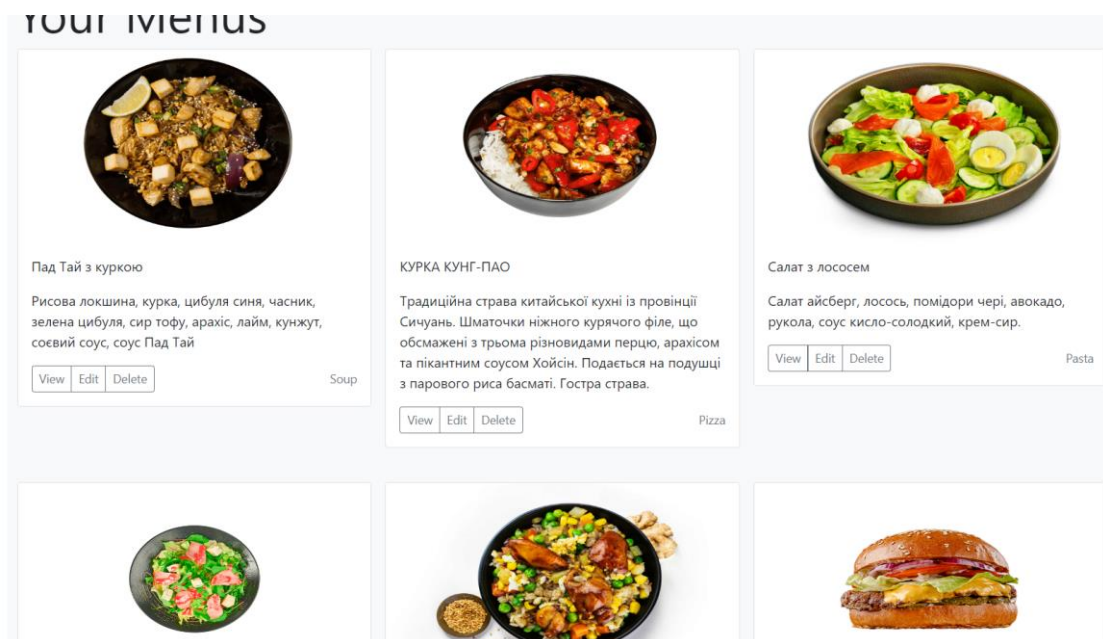


Рис 3.15. Оновлений Dashboard

3.3. Тестування роботи сайту

Було застосовано декілька етапів тестування для знаходження та виявлення дефектів у роботі сайту. Всі знайдені проблеми були або вирішені по мірі їх знаходження, або занотовані, у разі неможливості виправити. На кожному етапі буде проведено детальний аналіз функціоналу, можливостям та несправностям сайту.

Функціональне тестування. Цілями даного тестування є визначення функціональних помилок, невідповідностей завданню або очікуванням користувачів.

1. Реєстрація та авторизація користувачів:

- Реєстрація користувача ролі “user”
- Авторизація користувача ролі “user”
- Авторизація користувача ролі “admin”

2. Оформлення замовлень

- Введення даних для оформлення замовлення

- Введення неправильних або взагалі ігнорування деяких полів введення для створення замовлення.

3. Створення меню за допомогою користувача “admin”

- Додавання меню
- Введення неправильних або взагалі ігнорування деяких полів введення для створення меню

4. Зміна меню та замовлення за допомогою користувача “admin”

- Видалення замовлення
- Редагування замовлення
- Видалення меню
- Редагування меню

Тестування в різних середовищах. Метою цього тестування є виявлення дефектів при використанні різних браузерів чи розрешень монітора, а також перевірка на коректність роботи сайту.

Таблиця 3.1

Параметри конфігураційного тестування

Браузер / Розрішення	1024 * 768	1280 * 800	1680 * 1050
Internet Explorer 6	Так	Так	Так
Internet Explorer 7	Так	Так	Так
Internet Explorer 8	Так	Так	Так
Internet Explorer 9	Так	Так	Так
Firefox 30	Так	Так	Так
Firefox 41	Так	Так	Так
Google Chrome 22	Так	Так	Так
Opera 10	Так	Так	Так

Отже було проведено тестування веб-сайту для замовлення їжі на виявлення дефектів та коректності роботи. Були проведені такі етапи тестування як: функціональне та тестування в різних середовищах. За

результатами тестування було виявлено 5 функціональних помилок середнього пріоритету та 6 помилок адаптивного дизайну. Всі помилки було виправлено та взято до уваги. На даний момент сайт працює коректно та справляється з виконанням всіх поставлених задач.

Висновки до розділу 3

В даному розділі було наведено порядок налаштування сайту та визначено, що конфігурація комп'ютера не відіграє ролі в роботі веб-сайту. Це дає змогу використовувати створений продукт без обмежень. Крім цього, розроблений веб-додаток не потребує спеціалізованого апаратного забезпечення, додаткових налаштувань чи засобів розгортання крім стандартних для більшості сучасних сайтів.

Реалізований веб-сайт містить інтерфейси користувача та адміністратора. Для адміністратора було створено наступні сторінки: створення редагування та видалення меню, створення редагування та видалення об'єктів замовлення, Dashboard для контролю веб-сайтом та відстеження змін на ньому. Для користувачів було створено сторінки перегляду меню сайту, оформлення замовлень та детального ознайомлення з ними.

Також було проведено тестування веб-сайту для замовлення їжі на виявлення дефектів та коректності роботи. Були проведені такі етапи тестування як: функціональне та тестування в різних середовищах. За результатами тестування було виявлено 5 функціональних помилок середнього пріоритету та 6 помилок адаптивного дизайну. Всі помилки було виправлено та взято до уваги. На даний момент сайт працює коректно та справляється з виконанням всіх поставлених задач.

.

ВИСНОВКИ

Під час виконання випускної роботи бакалавра було спроектовано та реалізовано сайт для замовлення їжі. Під час роботи над поставленою задачею було виконано наступні кроки:

1. Було проведено тестування веб-сайту для замовлення їжі на виявлення дефектів та коректності роботи. Були проведені такі етапи тестування як: функціональне та тестування в різних середовищах. За результатами тестування було виявлено 5 функціональних помилок середнього пріоритету та 6 помилок адаптивного дизайну. Всі помилки було виправлено та взято до уваги. На даний момент сайт працює коректно та справляється з виконанням всіх поставлених задач.

2. Реалізовано безліч функцій, основними з яких є: оформлення замовлення користувачем зі збереженням для подальшого користування в БД, створення та редагування меню для адміністраторів сайту, створення нових користувачів та надання їм конкретних ролей на даному сайті. Всі ці функції реалізовані в контролерах Account, Home, Menu та Order. Реалізована та описана процедура створення бази даних, що є необхідним для вимог поточного завдання та виконання поставлених задач. База даних складається з 10 таблиць. Основними є: Users, Menus, Orders, Address, Roles та UserLogins. Допоміжними є: UserTokens, UserRoles, UserClaims, RoleClaims.

3. Було розглянуто алгоритми основних процесів сайту для замовлення їжі та проаналізовано взаємодію всіх вищеописаних класів під час таких процесів як: оформлення замовлень та створення меню. Також наведено акцентований розгляд функцій, які безпосередньо впливають на роботу з БД. Основну увагу було приділено функціям збереження даних в таблиці бази даних та їх редагування

4. Було наведено порядок налаштування сайту та визначено, що конфігурація комп'ютера не відіграє ролі в роботі веб-сайту. Це дає змогу використовувати створений продукт без обмежень. Крім цього, розроблений

веб-додаток не потребує спеціалізованого апаратного забезпечення, додаткових налаштувань чи засобів розгортання крім стандартних для більшості сучасних сайтів. Реалізований веб-сайт містить інтерфейси користувача та адміністратора. Для адміністратора було створено наступні сторінки: створення редагування та видалення меню, створення редагування та видалення об'єктів замовлення, Dashboard для контролю веб-сайтом та відстеження змін на ньому. Для користувачів було створено сторінки перегляду меню сайту, оформлення замовлень та детального ознайомлення з ними.

5. Також було проведено тестування веб-сайту для замовлення їжі на виявлення дефектів та коректності роботи. Були проведені такі етапи тестування як: функціональне та тестування в різних середовищах. За результатами тестування було виявлено 5 функціональних помилок середнього пріоритету та 6 помилок адаптивного дизайну. Всі помилки було виправлено та взято до уваги. На даний момент сайт працює коректно та справляється з виконанням всіх поставлених задач.

Створений веб-сайт готовий до використання та виконання поставлених перед ним задач. Може бути застосований як сайт для замовлення чи доставки їжі.

В подальшому даний сайт може бути вдосконалений шляхом впровадження нових функцій та можливостей, таких як відгуки або надання користувачам можливості редагувати профіль.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. What is a baseline model in machine learning? [Електронний ресурс] – Режим доступу до ресурсу: <https://stephenallwright.com/baseline-machine-learning-models/>.
2. Identity в ASP.NET [Електронний ресурс] – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/aspnet/identity/overview/getting-started/introduction-to-aspnet-identity>.
3. SQL Server Management Studio [Електронний ресурс] – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/sql/ssms/sql-server-management-studio-ssms?view=sql-server-ver16>.
4. Migrations in ASP.NET MVC [Електронний ресурс] – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/aspnet/mvc/overview/getting-started/getting-started-with-ef-using-mvc/migrations-and-deployment-with-the-entity-framework-in-an-asp-net-mvc-application>.
5. Stack Overflow [Електронний ресурс] – Режим доступу до ресурсу: <https://stackoverflow.com/>.
6. C# in depth Types guide [Електронний ресурс] – Режим доступу до ресурсу: <https://code-maze.com/csharp-value-vs-reference-types/>.
7. Cook Drive [Електронний ресурс] – Режим доступу до ресурсу: <https://cookdrive.com.ua/>.
8. Register and Login in C# [Електронний ресурс] – Режим доступу до ресурсу: <https://www.c-sharpcorner.com/article/create-loginsign-in-and-registration-sign-up-form-in-c-sharp-windows-form-with-da/>
9. MisterAM [Електронний ресурс] – Режим доступу до ресурсу: <https://misteram.com.ua/>
10. HTML tutorial [Електронний ресурс] – Режим доступу до ресурсу: <https://www.w3schools.com/html/>.
11. CSS tutorial [Електронний ресурс] – Режим доступу до ресурсу: <https://www.w3schools.com/css/>.

12. JavaScript info [Электронный ресурс] – Режим доступа до ресурсу:
<https://javascript.info/>.
13. C# interfaces [Электронный ресурс] – Режим доступа до ресурсу:
<https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/types/interfaces>
14. NuGet Package Manager [Электронный ресурс] – Режим доступа до
ресурсу: [https://learn.microsoft.com/en-us/nuget/consume-packages/install-
use-packages-visual-studio](https://learn.microsoft.com/en-us/nuget/consume-packages/install-use-packages-visual-studio).
15. Classes and objects in C# [Электронный ресурс] – Режим доступа до
ресурсу: https://www.w3schools.com/cs/cs_classes.php.

ДОДАТКИ

					ІІЗ.КР.Б – 121 – 23 – ІІЗ	
						51

Лістинг програми:

MenuController.cs

```

using DPLM.Data;
using DPLM.Interfaces;
using DPLM.Models;
using DPLM.ViewModels;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace DPLM.Controllers
{
    public class MenuController : Controller
    {
        private readonly IMenuRepository _menuRepository;
        private readonly IHttpContextAccessor _httpContextAccessor;

        public MenuController(IMenuRepository menuRepository, IHttpContextAccessor httpContextAccessor)
        {
            _menuRepository = menuRepository;
            _httpContextAccessor = httpContextAccessor;
        }

        public async Task<IActionResult> Index()
        {
            IEnumerable<Menu> menus = await _menuRepository.GetAll();
            return View(menus);
        }

        public async Task<IActionResult> Detail(int id)
        {
            Menu menu = await _menuRepository.GetByIdAsync(id);
            return View(menu);
        }

        public IActionResult Create()
        {
            var curUserId = _httpContextAccessor.HttpContext.User.GetUserId();
            var createMenuViewModel = new CreateMenuViewModel { AppUserId = curUserId };
            return View(createMenuViewModel);
        }

        [HttpPost]
        public async Task<IActionResult> Create(CreateMenuViewModel menuVM)
        {
            if (ModelState.IsValid)
            {
                var menu = new Menu
                {
                    Name = menuVM.Name,
                    Description = menuVM.Description,
                    Image = menuVM.Image,
                    //result.Url.ToString(),
                    MenuCategory = menuVM.MenuCategory,
                    AppUserId = menuVM.AppUserId,
                    Address = new Address
                    {
                        Street = menuVM.Address.Street,
                        City = menuVM.Address.City,
                    }
                };
            }
        }
    }
}

```

```

        _menuRepository.Add(menu);
        return RedirectToAction("Index");
    }

    return View(menuVM);
}

public async Task<IActionResult> Edit(int id)
{
    var menu = await _menuRepository.GetByIdAsync(id);
    if (menu == null) return View("Error");
    var menuVM = new EditMenuViewModel
    {
        Name = menu.Name,
        Description = menu.Description,
        Image = menu.Image,
        AddressId = menu.AddressId,
        Address = menu.Address,
        MenuCategory = menu.MenuCategory,
    };
    return View(menuVM);
}

[HttpPost]
public async Task<IActionResult> Edit(int id, EditMenuViewModel menuVM)
{
    if (!ModelState.IsValid)
    {
        ModelState.AddModelError("", "Failed to edit menu");
        return View("Edit", menuVM);
    }
    var userMenu = await _menuRepository.GetByIdAsyncNoTracking(id);

    if (userMenu != null)
    {
        var menu = new Menu
        {
            Id = id,
            Name = menuVM.Name,
            Description = menuVM.Description,
            Image = menuVM.Image,
            //photoResult.Url.ToString(),
            //AddressId = menuVM.AddressId,
            //Address = menuVM.Address,
        };
        // _menuRepository.Add(menu);
        _menuRepository.Update(menu);
        return RedirectToAction("Index");
    }
    else
    {
        return View(menuVM);
    }
}

public async Task<IActionResult> Delete(int id)
{
    var menuDetails = await _menuRepository.GetByIdAsync(id);
    if (menuDetails == null) return View("Error");
    return View(menuDetails);
}

[HttpPost, ActionName("Delete")]
public async Task<IActionResult> DeleteMenu(int id)
{
    var menuDetails = await _menuRepository.GetByIdAsync(id);
    if (menuDetails == null) return View("Error");
}

```

```

        _menuRepository.Delete(menuDetails);
        return RedirectToAction("Index");
    }
}

```

OrderController.cs

```

using DPLM.Data;
using DPLM.Interfaces;
using DPLM.Models;
using DPLM.Repository;
using DPLM.ViewModels;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System.Diagnostics;

namespace DPLM.Controllers
{
    public class OrderController : Controller
    {
        private readonly IOrderRepository _orderRepository;
        private readonly IHttpContextAccessor _httpContextAccessor;

        public OrderController(IOrderRepository orderRepository, IHttpContextAccessor httpContextAccessor,
            IAllOrdersRepository allOrders)
        {
            _orderRepository = orderRepository;
            _httpContextAccessor = httpContextAccessor;
        }

        public async Task<IActionResult> Index()
        {
            IEnumerable<Order> orders = await _orderRepository.GetAll();
            return View(orders);
        }

        public async Task<IActionResult> Detail(int id)
        {
            Order order = await _orderRepository.GetByIdAsync(id);
            return View(order);
        }

        public IActionResult Create()
        {
            var curUserId = _httpContextAccessor.HttpContext?.User.GetUserId();
            var createOrderViewModel = new CreateOrderViewModel { AppUserId = curUserId };
            return View(createOrderViewModel);
        }

        [HttpPost]
        public async Task<IActionResult> Create(CreateOrderViewModel orderVM)
        {
            if (ModelState.IsValid)
            {
                var order = new Order
                {
                    Name = orderVM.Name,
                    Description = orderVM.Description,
                    Image = orderVM.Image,
                    //result.Url.ToString(),
                    AppUserId = orderVM.AppUserId,
                    OrderCategory = orderVM.OrderCategory,
                    Address = new Address
                    {

```

```

        Street = orderVM.Address.Street,
        City = orderVM.Address.City,
    }
};
_orderRepository.Add(order);
return RedirectToAction("Index");
}
return View(orderVM);
}
public async Task<IActionResult> Edit(int id)
{
    var order = await _orderRepository.GetByIdAsync(id);
    if (order == null) return View("Error");
    var orderVM = new EditOrderViewModel
    {
        Name = order.Name,
        Description = order.Description,
        Image = order.Image,
        AddressId = order.AddressId,
        Address = order.Address,
        OrderCategory = order.OrderCategory,
    };
    return View(orderVM);
}

[HttpPost]
public async Task<IActionResult> Edit(int id, EditOrderViewModel orderVM)
{
    if (!ModelState.IsValid)
    {
        ModelState.AddModelError("", "Failed to edit order");
        return View("Edit", orderVM);
    }
    var userOrder = await _orderRepository.GetByIdAsyncNoTracking(id);

    if (userOrder != null)
    {
        //try
        //{
        //    await _photoService.DeletePhotoAsync(userOrder.Image);
        //}
        //catch (Exception ex)
        //{
        //    ModelState.AddModelError("", "Could not delete photo");
        //    return View(orderVM);
        //}
        //var photoResult = await _photoService.AddPhotoAsync(orderVM.Image);
        var order = new Order
        {
            Id = id,
            Name = orderVM.Name,
            Description = orderVM.Description,
            Image = orderVM.Image,
            //photoResult.Url.ToString(),
            AddressId = orderVM.AddressId,
            Address = orderVM.Address,
        };

        _orderRepository.Update(order);
        return RedirectToAction("Index");
    }
    else
    {
        return View(orderVM);
    }
}

```

```

public async Task<IActionResult> Delete(int id)
{
    var orderDetails = await _orderRepository.GetByIdAsync(id);
    if (orderDetails == null) return View("Error");
    return View(orderDetails);
}
[HttpPost, ActionName("Delete")]
public async Task<IActionResult> DeleteMenu(int id)
{
    var orderDetails = await _orderRepository.GetByIdAsync(id);
    if (orderDetails == null) return View("Error");

    _orderRepository.Delete(orderDetails);
    return RedirectToAction("Index");
}
}
}

```

DashboardController.cs

```

using DPLM.Interfaces;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using DPLM.Data;
using DPLM.ViewModels;

namespace DPLM.Controllers
{
    public class DashboardController : Controller
    {
        private readonly IDashboardRepository _dashboardRepository;
        private readonly IHttpContextAccessor _httpContextAccessor;

        public DashboardController(IDashboardRepository dashboardRepository, IHttpContextAccessor httpContextAccessor)
        {
            _dashboardRepository = dashboardRepository;
        }

        public async Task<IActionResult> Index()
        {
            var userOrders = await _dashboardRepository.GetAllUserOrders();

            var userMenus = await _dashboardRepository.GetAllUserMenus();
            var userOrderDetails = await _dashboardRepository.GetAllUserOrderDetails();

            var dashboardViewModel = new DashboardViewModel()
            {
                Orders = userOrders,
                Menus = userMenus,
                Details = userOrderDetails
            };
            return View(dashboardViewModel);
        }

        public async Task<IActionResult> EditUserProfile()
        {
            var curUserId = _httpContextAccessor.HttpContext.User.GetUserId();
            var user = await _dashboardRepository.GetUserById(curUserId);
            if (user == null) return View("Error");
            var editUserViewModel = new EditUserDashboardViewModel()
            {
                Id = curUserId,
                Price = user.Price,
                Tip = user.Tip,
                ProfileImageUrl = user.ProfileImageUrl,
            }
        }
    }
}

```

```

        City = user.City,
        Street = user.Street
    };
    return View(editUserViewModel);
}
}
}

```

AccountController.cs

```

using DPLM.Data;
using DPLM.Models;
using DPLM.ViewModels;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;

namespace DPLM.Controllers
{
    public class AccountController : Controller
    {
        private readonly UserManager<AppUser> _userManager;
        private readonly SignInManager<AppUser> _signInManager;
        private readonly ApplicationDbContext _context;

        public AccountController(UserManager<AppUser> userManager, SignInManager<AppUser> signInManager,
            ApplicationDbContext context)
        {
            _context = context;
            _signInManager = signInManager;
            _userManager = userManager;
        }

        public IActionResult Login()
        {
            var response = new LoginViewModel();
            return View(response);
        }

        [HttpPost]
        public async Task<IActionResult> Login(LoginViewModel loginViewModel)
        {
            if (!ModelState.IsValid) return View(loginViewModel);

            var user = await _userManager.FindByEmailAsync(loginViewModel.EmailAddress);

            // check passwd
            if (user != null)
            {
                var passwordCheck = await _userManager.CheckPasswordAsync(user, loginViewModel.Password);
                if (passwordCheck)
                {
                    // correct
                    var result = await _signInManager.PasswordSignInAsync(user, loginViewModel.Password, false, false);
                    if (result.Succeeded)
                    {
                        return RedirectToAction("Index", "Order");
                    }
                }
                // incorrect
                TempData["Error"] = "Wrong info. Please try again";
                return View(loginViewModel);
            }
            // user not found
            TempData["Error"] = "Wrong info. Please try again";
            return View(loginViewModel);
        }
    }
}

```

```

public IActionResult Register()
{
    var response = new RegisterViewModel();
    return View(response);
}

[HttpPost]
public async Task<IActionResult> Register(RegisterViewModel registerViewModel)
{
    if (!ModelState.IsValid) return View(registerViewModel);

    var user = await _userManager.FindByEmailAsync(registerViewModel.EmailAddress);
    if (user != null)
    {
        TempData["Error"] = "This email address is already used";
        return View(registerViewModel);
    }

    var newUser = new AppUser()
    {
        Email = registerViewModel.EmailAddress,
        UserName = registerViewModel.EmailAddress
    };
    var newUserResponse = await _userManager.CreateAsync(newUser, registerViewModel.Password);

    if (newUserResponse.Succeeded)
        await _userManager.AddToRoleAsync(newUser, UserRoles.User);

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

[HttpPost]

public async Task<IActionResult> Logout()
{
    await _signInManager.SignOutAsync();
    return RedirectToAction("Index", "Order");
}
}
}

```

HomeController.cs

```

using DPLM.Models;
using Microsoft.AspNetCore.Mvc;
using System.Diagnostics;

namespace DPLM.Controllers
{
    public class HomeController : Controller
    {
        private readonly ILogger<HomeController> _logger;

        public HomeController(ILogger<HomeController> logger)
        {
            _logger = logger;
        }

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

```

```

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

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

```

DashboardRepository.cs

```

using DPLM.Data;
using DPLM.Interfaces;
using DPLM.Models;

namespace DPLM.Repository
{
    public class DashboardRepository : IDashboardRepository
    {
        private readonly ApplicationDbContext _context;
        private readonly IHttpContextAccessor _httpContextAccessor;

        public DashboardRepository(ApplicationDbContext context, IHttpContextAccessor httpContextAccessor)
        {
            _context = context;
            _httpContextAccessor = httpContextAccessor;
        }

        public async Task<List<Menu>> GetAllUserMenus()
        {
            var curUser = _httpContextAccessor.HttpContext?.User.GetUserId();
            var userMenus = _context.Menus.Where(r => r.AppUser.Id == curUser);
            return userMenus.ToList();
        }

        public async Task<List<Order>> GetAllUserOrders()
        {
            var curUser = _httpContextAccessor.HttpContext?.User.GetUserId();
            var userOrders = _context.Orders.Where(r => r.AppUser.Id == curUser);
            return userOrders.ToList();
        }

        public async Task<List<OrderDetails>> GetAllUserOrderDetails()
        {
            var curUser = _httpContextAccessor.HttpContext?.User.GetUserId();
            var userOrderDetails = _context.OrderDetails.Where(r => r.AppUser.Id == curUser);
            return userOrderDetails.ToList();
        }

        public Task<AppUser> GetByIdNoTracking(string id)
        {
            throw new NotImplementedException();
        }

        public async Task<AppUser> GetUserById(string id)
        {
            return await _context.Users.FindAsync(id);
        }

        public bool Save()
        {

```

```

        throw new NotImplementedException();
    }

    public bool Update(AppUser user)
    {
        throw new NotImplementedException();
    }
}

```

MenuRepository.cs

```

using DPLM.Data;
using DPLM.Interfaces;
using DPLM.Models;
using Microsoft.EntityFrameworkCore;

namespace DPLM.Repository
{
    public class MenuRepository : IMenuRepository
    {
        private readonly ApplicationDbContext _context;
        public MenuRepository(ApplicationDbContext context)
        {
            _context = context;
        }

        public bool Add(Menu menu)
        {
            _context.Add(menu);
            return Save();
        }

        public bool Delete(Menu menu)
        {
            _context.Remove(menu);
            return Save();
        }

        public async Task<IEnumerable<Menu>> GetAll()
        {
            return await _context.Menus.ToListAsync();
        }

        public async Task<Menu> GetByIdAsync(int id)
        {
            return await _context.Menus.Include(i => i.Address).FirstOrDefaultAsync(x => x.Id == id);
        }

        public async Task<Menu> GetByIdAsyncNoTracking(int id)
        {
            return await _context.Menus.Include(i => i.Address).AsNoTracking().FirstOrDefaultAsync(x => x.Id == id);
        }

        public bool Save()
        {
            var saved = _context.SaveChanges();
            return saved > 0 ? true : false;
        }

        public bool Update(Menu menu)
        {
            _context.Update(menu);
            return Save();
        }
    }
}

```

```
}
```

Order.Repository.cs

```
using DPLM.Data;
using DPLM.Interfaces;
using DPLM.Models;
using Microsoft.EntityFrameworkCore;

namespace DPLM.Repository
{
    public class OrderRepository : IOrderRepository
    {
        private readonly ApplicationDbContext _context;
        public OrderRepository(ApplicationDbContext context)
        {
            _context = context;
        }

        public bool Add(Order order)
        {
            _context.Add(order);
            return Save();
        }

        public bool Delete(Order order)
        {
            _context.Remove(order);
            return Save();
        }

        public async Task<IEnumerable<Order>> GetAll()
        {
            return await _context.Orders.ToListAsync();
        }

        public async Task<Order> GetByIdAsync(int id)
        {
            return await _context.Orders.Include(i => i.Address).FirstOrDefaultAsync(x => x.Id == id);
        }

        public async Task<Order> GetByIdAsyncNoTracking(int id)
        {
            return await _context.Orders.Include(i => i.Address).AsNoTracking().FirstOrDefaultAsync(x => x.Id == id);
        }

        public bool Save()
        {
            var saved = _context.SaveChanges();
            return saved > 0 ? true : false;
        }

        public bool Update(Order order)
        {
            _context.Update(order);
            return Save();
        }
    }
}
```

MenuModel.cs

```
using DPLM.Data.Enum;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
```

					ИПЗ.КР.Б – 121 – 23 – ИПЗ	
						61

```

namespace DPLM.Models
{
    public class Menu
    {
        [Key]
        public int Id { get; set; }
        public string Name { get; set; }
        public string Description { get; set; }
        public string Image { get; set; }
        [ForeignKey("Address")]
        public int AddressId { get; set; }
        public Address Address { get; set; }
        public MenuCategory MenuCategory { get; set; }
        [ForeignKey("AppUser")]
        public string? AppUserId { get; set; }
        public AppUser? AppUser { get; set; }
    }
}

```

OrderModel.cs

```

using DPLM.Data.Enum;
using System.ComponentModel.DataAnnotations.Schema;
using System.ComponentModel.DataAnnotations;

```

```

namespace DPLM.Models
{
    public class Order
    {
        [Key]
        public int Id { get; set; }
        public string Name { get; set; }
        public string Description { get; set; }
        public string Image { get; set; }
        [ForeignKey("Address")]
        public int AddressId { get; set; }
        public Address Address { get; set; }
        public OrderCategory OrderCategory { get; set; }
        [ForeignKey("AppUser")]
        public string? AppUserId { get; set; }
        public AppUser? AppUser { get; set; }
    }
}

```

AppUserModel.cs

```

using Microsoft.AspNetCore.Identity;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

```

```

namespace DPLM.Models
{
    public class AppUser : IdentityUser
    {
        public int? Price { get; set; }
        public int? Tip { get; set; }
        public string? ProfileImageUrl { get; set; }
        public string? City { get; set; }
        public string? Street { get; set; }
        [ForeignKey("Address")]
        public int? AddressId { get; set; }
    }
}

```

```

        public Address? Address { get; set; }
        public ICollection<Menu> Menus { get; set; }
        public ICollection<Order> Orders { get; set; }
    }
}

```

Program.cs

```

using DPLM.Data;
using DPLM.Interfaces;
using DPLM.Models;
using DPLM.Repository;
using Microsoft.AspNetCore.Authentication.Cookies;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllersWithViews();
builder.Services.AddScoped<IMenuRepository, MenuRepository>();
builder.Services.AddScoped<IOrderRepository, OrderRepository>();
builder.Services.AddScoped<IAllOrdersRepository, AllOrdersRepository>();
builder.Services.AddScoped<IDashboardRepository, DashboardRepository>();
builder.Services.AddDbContext<ApplicationDbContext>(options =>
{
    options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection"));
});
builder.Services.AddIdentity<AppUser, IdentityRole>()
    .AddEntityFrameworkStores<ApplicationDbContext>();
builder.Services.AddMemoryCache();
builder.Services.AddSession();
builder.Services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme)
    .AddCookie();

var app = builder.Build();

if (args.Length == 1 && args[0].ToLower() == "seeddata")
{
    await Seed.SeedUsersAndRolesAsync(app);
    //Seed.SeedData(app);
}

// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
    // The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts.
    app.UseHsts();
}

app.UseHttpsRedirection();
app.UseStaticFiles();

app.UseRouting();
app.UseAuthentication();
app.UseAuthorization();

app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");

app.Run();

```

LoginViewModel.cs

```
using System.ComponentModel.DataAnnotations;

namespace DPLM.ViewModels
{
    public class LoginViewModel
    {
        [Display(Name = "Email Address")]
        [Required(ErrorMessage = "Email address is required")]
        public string EmailAddress { get; set; }
        [Required]
        [DataType(DataType.Password)]
        public string Password { get; set; }
    }
}
```

RegisterViewModel.cs

```
using System.ComponentModel.DataAnnotations;

namespace DPLM.ViewModels
{
    public class RegisterViewModel
    {
        [Display(Name = "Email Address")]
        [Required(ErrorMessage = "Email address is required")]
        public string EmailAddress { get; set; }
        [Required]
        [DataType(DataType.Password)]
        public string Password { get; set; }
        [Display(Name = "Confirm Password")]
        [Required(ErrorMessage = "Confirm Password is required")]
        [DataType(DataType.Password)]
        [Compare("Password", ErrorMessage = "Password is not same")]
        public string ConfirmPassword { get; set; }
    }
}
```