Міністерство освіти і науки України
Університет митної справи та фінансів

Факультет інноваційних технологій
Кафедра комп'ютерних наук та інженерії програмного забезпечення

Кваліфікаційна робота бакалавра

на тему : «Розробка веб-застосунку для закладів громадського харчування»

Виконав: студент групи ІПЗ20-2
Спеціальність 121 «Інженерія програмного забезпечення»
Бабіч Віталій Сергійович
(прізвище та ініціали)
Керівник к.е.н., Яковенко Т.Ю.
(науковий ступінь, вчене звання, прізвище та ініціали)
Рецензент Університет митної справи та фінансів
(місце роботи)
Доцент кафедри кібербезпеки та інформаційних технологій
(посада)
к.т.н., доцент Савченко В.Ю.
(науковий ступінь, вчене звання, прізвище та ініціали)

Дніпро – 2024

АНОТАЦІЯ

Бабіч В. С. Розробка веб-застосунку для закладів громадського харчування.

Кваліфікаційна робота на здобуття освітнього ступеня бакалавр за спеціальністю 121 «Інженерія програмного забезпечення» – Університет митної справи та фінансів, Дніпро, 2024.

Кваліфікаційна робота присвячена розробці веб-застосунку для закладів громадського харчування з метою автоматизації процесів замовлення, оплати та доставки. Сучасні тенденції цифрової трансформації бізнесу та зростаюча потреба в автоматизації процесів обумовлюють необхідність створення ефективних веб-застосунків для ресторанів та кафе, що дозволяють підвищити ефективність роботи та якість обслуговування клієнтів.

В процесі дослідження було проаналізовано існуючі рішення та методи розробки веб-застосунків, обрано оптимальні програмні засоби для реалізації проекту. Розробка проводилася із застосуванням архітектурного патерну VIPER, який забезпечує високу модульність, тестованість та масштабованість додатку, що дозволяє чітко розділити обов'язки між компонентами, підвищуючи якість коду та зменшуючи складність підтримки.

Для реалізації back-end частини за допомогою платформи .NET, ASP.NET та мови програмування C# Розроблений веб-застосунок дозволяє здійснювати замовлення онлайн, переглядати меню, обирати страви, здійснювати оплату та відстежувати статус замовлення. Також дозволяє збирати та аналізувати дані про клієнтів, що допомагає закладам громадського харчування краще розуміти потреби своєї аудиторії та ефективніше управляти бізнесом.

Ключові слова: веб-застосунок, заклади громадського харчування, автоматизація, VIPER, .NET, ASP.NET, C#.

ABSTRACT

Babich V. S. Development of a Web Application for Catering Establishments. Bachelor's Qualification

Bachelor's thesis for obtaining a degree in Software Engineering, specialty 121. – University of Customs and Finance, Dnipro, 2024.

This qualification thesis is dedicated to the development of a web application for catering establishments with the aim of automating the processes of ordering, payment, and delivery.

Modern trends in business digital transformation and the increasing need for process automation necessitate the creation of effective web applications for restaurants and cafes that enhance operational efficiency and customer service quality.

The research analyzed existing solutions and methods for developing web applications, selecting optimal software tools for project implementation. The development was carried out using the VIPER architectural pattern, which ensures high modularity, testability, and scalability of the application, allowing clear division of responsibilities among components, improving code quality, and reducing maintenance complexity.

The back-end part was implemented using the .NET platform, ASP.NET, and the C# programming language. The developed web application allows customers to place orders online, view menus, choose dishes, make payments, and track order status. It also enables the collection and analysis of customer data, helping catering establishments better understand their audience's needs and manage their business more effectively.

Keywords: web application, catering establishments, automation, VIPER, .NET, ASP.NET, C#.

ЗМІСТ

ВСТУП

Актуальність проблеми. Розробка веб-застосунків для закладів громадського харчування зумовлена сучасними тенденціями цифрової трансформації бізнесу та зростаючою потребою в автоматизації процесів. У нинішніх умовах конкуренції ресторани та кафе шукають способи підвищити ефективність роботи та якість обслуговування клієнтів. Впровадження веб-застосунків дозволяє автоматизувати процеси замовлення, оплати та доставки, що сприяє зменшенню витрат, підвищенню продуктивності персоналу та забезпеченню високого рівня задоволення клієнтів.

Сучасні користувачі очікують швидкого та легкого доступу до послуг через інтернет. Веб-застосунки надають можливість здійснювати замовлення онлайн, переглядати меню, обирати страви, здійснювати оплату та відстежувати статус замовлення. Це значно підвищує комфорт клієнтів, стимулює їх до повторних візитів та замовлень, що в свою чергу сприяє зростанню доходів закладу.

Крім того, веб-застосунки відкривають можливості для збору та аналізу даних про клієнтів та їхні вподобання. Це дозволяє закладам громадського харчування краще розуміти потреби своєї аудиторії, адаптувати меню та маркетингові стратегії відповідно до отриманих даних, що сприяє більш ефективному управлінню бізнесом. Аналіз даних допомагає виявляти популярні страви, оптимізувати запаси, планувати акції та спеціальні пропозиції.

Розробка програмного забезпечення – це комплексний процес, який охоплює кілька етапів, таких як аналіз вимог, проектування, реалізація, тестування, впровадження та підтримка. Кожен етап вимагає виконання певних завдань, що допомагають створити функціональне та якісне програмне забезпечення. Існує кілька моделей, які описують процес розробки, кожна з яких має свої особливості та підходи до управління розробкою програмного

забезпечення. Вибір архітектури та технологій має вирішальне значення для створення ефективних та масштабованих веб-рішень, які відповідають сучасним вимогам бізнесу та користувачів.

Метою роботи є автоматизація роботи закладів громадського харчування.

Методи дослідження: обробка та аналіз інформації, методи проектування та розробки веб-додатків.

У відповідності до поставленої мети в кваліфікаційній роботі поставлені наступні завдання дослідження:

1. Проаналізувати технічні засоби, що застосовуються для розробки веб-додатків.

2. Розробити архітектуру та функціональні можливості веб-додатку.

3. Розробити веб-додаток з застосуванням патерну проектування VIPER.

4. Провести тестування.

Об'єктом дослідження розробка програмного забезпечення в сфері надання ресторанних послуг.

Предметом дослідження є апаратно-програмне забезпечення для розробки веб-додатку.

Структура роботи:

- Розділ 1 Аналіз існуючих рішень.
- Розділ 2 Аналіз засобів реалізації веб-застосунку
- Розділ 3 Розробка back-end частини веб-додатку «Ресторан»

Робота складається зі вступу, 3-х розділів, висновків, списку використаних джерел з 15 найменувань. Обсяг роботи 60 сторінок кваліфікаційної роботи, 43 рисунків, 1 таблиці.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Аналіз публікацій щодо розробки веб-застосунків

Веб-додаток - це розподілений додаток, де браузер грає роль клієнта, а веб-сервер - роль сервера. Браузер, будучи складовою частиною операційної системи, відповідає за відображення веб-сторінок та отримує оновлення від постачальника операційної системи. Логіка додатку знаходиться на сервері, а браузер переважно відображає інформацію, отриману від сервера, і передає дані назад через мережу. Одна з основних переваг цього підходу - це незалежність клієнтів від операційної системи користувача, що робить веб-додатки міжплатформовими.

Стандартні функції браузера працюють незалежно від операційної системи клієнта. Таким чином, замість створення різних версій для Windows, Mac OS X або GNU/Linux, додаток розробляється один раз і розгортається на будь-якій платформі. Також якщо не знаєте URL конкретного веб-додатка можливо скористатися пошуковою системою для його знаходження.

Практично будь-який ресурс в Інтернеті є веб-додатком, включаючи пошукові системи, відеосервіси, соціальні мережі, інтернет-магазини, сервіси замовлень та бронювань. Веб-сайти класифікуються за різними категоріями, такими як блоги, бізнес-сайти, інформаційні веб-сайти, ігрові веб-сайти, довідкові ресурси, медіа-обмінні платформи, новинні веб-сайти, навчальні ресурси, соціальні мережі, веб-пошта та інші. Деякі сайти можуть одночасно належати до кількох категорій, наприклад, бути блогом і форумом водночас.

Крім того, важливо розуміти різницю між веб-сайтом і веб-сторінкою. Веб-сайт є центральним місцем, що містить одну або більше веб-сторінок. Наприклад, YouTube - це веб-сайт з мільйонами різних веб-сторінок [1].

Існує два типи веб-сторінок: статичні і динамічні.

Статичні сторінки однакові для всіх відвідувачів. Процес їх відображення виглядає так:

1) Користувач вводить запит або адресу сторінки у браузері.

2) Браузер надсилає запит на веб-сервер.

3) Веб-сервер визначає, що сторінка є статичною і відправляє її без змін назад до браузера.

Динамічні сторінки змінюються залежно від користувача чи інших факторів. Процес їх відображення виглядає так:

1) Браузер надсилає запит на веб-сервер, який може містити специфічні інструкції для динамічного відображення.

2) Веб-сервер передає запит на сервер додатків, де застосовується логіка для створення динамічної сторінки.

3) Сформована сторінка надсилається назад до браузера для відображення користувачу.

Таким чином, статичні сторінки однакові для всіх користувачів, тоді як динамічні можуть змінюватися залежно від конкретних умов або дій користувача.

Завдяки великій кількості різноманітних веб-сайтів, їх можна класифікувати за кінцевою метою створення:

1) Сайт-візитка

Використовується компаніями для представлення своїх послуг в Інтернеті. Він зазвичай містить кілька основних сторінок: «Про нас», «Послуги», «Вартість», «Контакти».

2) Корпоративний сайт

Слугує для надання інформації про компанію клієнтам, партнерам та медіапредставникам. Такий сайт зазвичай містить актуальні новини, прес-релізи, детальну інформацію про компанію, перелік послуг та клієнтів. Основною метою є ефективне представлення компанії в онлайн-просторі.

3) Інтернет-магазин

Веб-сайт, де користувачі можуть вибрати та придбати товари або послуги, а продавці можуть їх продавати. Такі магазини несуть відповідальність за своєчасне виконання замовлень покупців.

4) Інформаційний сайт

Містить новини та матеріали інформаційного характеру. Ці сайти допомагають користувачам знайти відповіді на їх питання та дізнатися останні новини світу.

5) Блог

Особистий сайт конкретної людини, яка заповнює його згідно своїх бажань і цілей. Він зазвичай складається з кількох сторінок та має простий інтерфейс.

6) Форум – місце, де люди формуються у різні чати за своїми інтересами та спілкуються між собою. Форуми мають контент, який генерують користувачі самостійно [4].

Враховуючи наші потреби, веб-додаток може бути подібним до онлайн-платформи для продажу товарів – «Інтернет-магазин».

1.2. Аналіз методів та підходів розробки веб- застосунків

Розробка програмного забезпечення – це комплексний процес, який охоплює кілька етапів, таких як аналіз вимог, проектування, реалізація, тестування, впровадження та підтримка. Кожен етап вимагає виконання певних завдань, що допомагають створити функціональне та якісне програмне забезпечення. Існує кілька моделей, які описують процес розробки, кожна з яких має свої особливості та підходи до управління розробкою програмного забезпечення.

Залежно від обраної моделі розробки використовуються різні методології, такі як водоспадна модель, ітеративна модель і спіральна модель. Водоспадна модель передбачає послідовне виконання етапів, де кожен етап розробки виконується послідовно і без можливості повернення до попередніх етапів. Ця модель підходить для проектів з чітко визначеними вимогами, але може бути неефективною для великих проектів через труднощі з адаптацією до змін і накопичення помилок [2].

Ітеративні моделі розробки, зокрема інкрементна модель, дозволяють розбивати процес на кілька ітерацій або міні-циклів, кожен з яких включає основні стадії моделі життєвого циклу і присвячений розробці окремого компонента системи. Ітеративна модель забезпечує більш гнучкий підхід до розробки програмного забезпечення, дозволяючи поступово розширювати функціональність системи, додавати нові компоненти та вносити зміни на кожній ітерації.

Що стосується архітектурних патернів популярними є Model-View-Controller (MVC), Model-View-ViewModel (MVVM) та VIPER. Патерн MVC розділяє додаток на три основні компоненти: Model (модель), View (подання) і Controller (контролер). Модель відповідає за управління даними, подання – за відображення інтерфейсу, а контролер – за взаємодію між моделлю і поданням. Чітке розділення логіки дозволяє легко підтримувати та розширювати додаток, але можливі складнощі з управлінням залежностями між компонентами [2].

Патерн MVVM розділяє додаток на три компоненти: Model (модель), View (подання) і ViewModel (модель подання). Модель подання виступає посередником між моделлю і поданням, забезпечуючи зв'язок даних і логіку поведінки. Перевагами MVVM є можливість повторного використання коду і легкість в автоматизації тестування, хоча можливість перевантаження ViewModel може ускладнити підтримку великих проектів.

Патерн VIPER розділяє додаток на п'ять компонентів: View (подання), Interactor (інтерактор), Presenter (презентер), Entity (сутність) і Router (маршрутизатор). Кожен компонент відповідає за окрему частину функціональності додатку. VIPER забезпечує чітке розділення обов'язків і високий рівень модульності, але реалізація цього патерну може бути складною і потребує написання великої кількості коду.

У розробці веб-застосунків є поділ на клієнтську сторону, серверну сторону та технологію баз даних.

Клієнтська сторона відповідає за відображення інтерфейсу та взаємодію з користувачем. Вона зазвичай реалізується за допомогою технологій HTML, CSS та JavaScript. Фреймворки, такі як Angular, React або Vue.js, забезпечують структуру та полегшують розробку складних користувацьких інтерфейсів.

Серверна сторона відповідає за обробку запитів від клієнта, управління даними та виконання бізнес-логіки. Серверні технології включають Node.js, Ruby on Rails, Django, ASP.NET та інші. Вибір серверної технології залежить від вимог проекту, обсягу даних та необхідної продуктивності.

Технології баз даних забезпечують зберігання, управління та доступ до даних. Популярними є реляційні бази даних, такі як MySQL, PostgreSQL, а також NoSQL бази даних, такі як MongoDB, Cassandra. Вибір технології баз даних залежить від характеру даних, обсягів зберігання та вимог до масштабованості.

Технологічні стеки для розробки веб-застосунків включають MEAN (MongoDB, Express.js, Angular, Node.js), LAMP (Linux, Apache, MySQL, PHP) і MERN (MongoDB, Express.js, React, Node.js). MEAN є повністю JavaScript-орієнтованим стеком, що забезпечує високу продуктивність і активну спільноту, але може викликати труднощі з масштабуванням. LAMP включає операційну систему Linux, веб-сервер Apache, базу даних MySQL і мову програмування PHP. Цей стек відомий своєю стабільністю і надійністю, але може мати проблеми з продуктивністю на великих проектах. MERN, схожий на MEAN, використовує React замість Angular для створення інтерфейсів, що забезпечує легкість у використанні, але також може стикатися з труднощами інтеграції та масштабування [3].

Розробка веб-застосунків вимагає ретельного вибору методів і підходів, враховуючи вимоги проекту, ресурси та кінцеву мету. Використання сучасних методологій, архітектурних патернів і технологічних стеків дозволяє створювати ефективні, гнучкі та масштабовані рішення. Вибір конкретного методу залежить від багатьох факторів, включаючи масштаб проекту, вимоги до продуктивності та наявні ресурси.

1.3. Висновок до першого розділу

У першому розділі було проведено детальний аналіз існуючих рішень щодо розробки веб-застосунків. Основні результати аналізу вказують на важливість правильного вибору архітектурних підходів та технологічних стеків для успішної розробки веб-застосунків.

Аналіз публікацій показав, що веб-додатки є різноманітними за своїми функціональними можливостями та можуть охоплювати широке коло сервісів, включаючи пошукові системи, соціальні мережі, інтернет-магазини тощо. При цьому основна перевага веб-додатків полягає у їх міжплатформенності та незалежності від операційної системи клієнта.

Розглянуто різні методи та підходи до розробки веб-застосунків, такі як водоспадна модель, ітеративна модель та спіральна модель, кожна з яких має свої особливості та сфери застосування. Вибір конкретної моделі залежить від масштабів проекту, вимог до продуктивності та наявних ресурсів.

Також було проаналізовано популярні архітектурні патерни, включаючи MVC (Model-View-Controller), MVVM (Model-View-ViewModel) та VIPER (View-Interactor-Presenter-Entity-Router). Кожен з цих патернів забезпечує структурованість, модульність та гнучкість коду, що полегшує процеси розробки, тестування та підтримки додатків.

Вибір архітектури та технологій має вирішальне значення для створення ефективних та масштабованих веб-рішень, які відповідають сучасним вимогам бізнесу та користувачів.

РОЗДІЛ 2. АНАЛІЗ ЗАСОБІВ РЕАЛІЗАЦІЇ ВЕБ-ЗАСТОСУНКУ

2.1. Вибір програмних засобів для реалізації проекту

Веб-застосунок кіносервіс є різновидом інтернет-магазину. Інтернет-магазин – сайт, з якого можна вибрати та замовити потрібний товар чи послугу. У даному випадку таким товаром є страви на в ресторані.

Інтернет-магазин передбачає фінансові розрахунки на відміну від сайтів, які надають послуги безкоштовно. Важливими елементами інтернет-магазину є оновлення асортименту (товари та їх кількість), можливість додавання товарів до «кошику», а також вхід для зареєстрованих користувачів. Інтернет-магазини реалізуються у вигляді вебдодатків.

Проте різні реалізації HTML, CSS, DOM та інших специфікацій у браузерах можуть створювати проблеми під час розробки та підтримки вебзастосунків. Крім того, можливість користувачів налаштовувати багато параметрів браузера (наприклад, розмір шрифту, кольори, відключення сценаріїв) може заважати коректній роботі додатка.

Веб-додаток обробляє запити від клієнта, виконує необхідні операції, після чого формує веб-сторінку і відправляє її назад клієнту через мережу за допомогою HTTP. Такий додаток також може виступати у ролі клієнта для інших служб, таких як бази даних або інші веб-додатки [5].

При виборі технологій для створення веб-додатку "ресторан" слід зважати на багато чинників, включно з функціональними вимогами проекту, потребами користувачів, технічними можливостями, масштабованістю, безпекою та ефективністю. Ось декілька важливих аспектів, які варто розглянути:

1) Веб-фреймворк

Використання веб-фреймворка дозволяє прискорити розробку, забезпечити стандартизовану структуру проекту та підтримку різних

функціональних можливостей. Популярні веб-фреймворки включають Spring, Django, Flask, Ruby on Rails, Laravel, Yii, ASP.NET та Angular.

2) Мова програмування

Вибір мови програмування залежить від веб-фреймворка та його підтримки. Наприклад, Django та Flask використовують Python, ASP.NET – C#. Важливо вибрати мову з якою команда розробки має досвід і комфортність роботи.

3) База даних

Вибір бази даних залежить від вимог до обробки даних, масштабованості та швидкодії. Популярні варіанти включають MySQL, PostgreSQL, MongoDB, а також хмарні бази даних, такі як Amazon RDS або Microsoft Azure SQL Database.

4) Хмарні платформи

Використання хмарних платформ забезпечує інфраструктуру та послуги для хостингу та масштабування веб-застосунків. Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform (GCP) надають широкий набір сервісів для веб-розробки.

5) Інструменти розробки

Застосування різних інструментів розробки полегшує роботу з кодом, версіонуванням, тестуванням та розгортанням веб-застосунків. Наприклад, Git для версіонування коду, Docker для контейнеризації, фреймворки для тестування застосунків такі як Selenium або Jest [8].

Провівши детальний аналіз було обрано такі технології реалізації серверної частини для веб-додатку: .Net, ASP.Net, Visual Studio Community 2022, NuGet та Json.

2.2. Програмні засоби для розробки серверної частини

Для реалізації проекту потрібна мова програмування, що підтримує об'єктно-орієнтовану парадигму. Серед найпоширеніших мов на ринку є Java,

C++, C# та Python. Оскільки наш проект орієнтований на веб-програмування, ми можемо виключити C++, оскільки вона має високий рівень складності для розробки веб-додатків. Python є гарним вибором завдяки простоті та швидкості розробки на веб-платформі, але вона може бути менш гнучкою для наших потреб. Отже, наш вибір зводиться до Java та C#.

Було обрано C# з кількох важливих причин:

1) Розширені можливості розробки

C# підтримує сучасні об'єктно-орієнтовані концепції, такі як інтерфейси, абстрактні класи, наслідування та поліморфізм. Це робить його ідеальним для створення складних та масштабованих веб-додатків.

2) Інтеграція з хмарними сервісами

.NET Core добре інтегрується з хмарними платформами, такими як Microsoft Azure, що забезпечує гнучкість та масштабованість нашого додатка.

3) Безпека

C# забезпечує високий рівень безпеки за рахунок вбудованих механізмів типізації та управління пам'яттю, включаючи автоматичний збирач сміття, що знижує ризик витоку пам'яті та інших помилок пов'язаних з управлінням пам'яттю.

4) Спільнота та підтримка, документація

C# має велику та активну спільноту розробників, а також підтримується Microsoft, що забезпечує наявність великої кількості ресурсів, документації та інструментів для вирішення будь-яких проблем, які можуть виникнути під час розробки.

5) Visual Studio

Розробка на C# здійснюється у Visual Studio, яка є одним з найпотужніших та найзручніших середовищ розробки з широким спектром інструментів для дебагінгу, тестування та розгортання додатків.

Основною особливістю мови C# є її використання CIL (Common Intermediate Language) як проміжної мови. Всі компільовані класи перетворюються на CIL, який потім обробляється віртуальною машиною CLR

(Common Language Runtime). CLR є частиною .NET Framework чи .NET Core, що дозволяє запускати C#-програми на різних операційних системах.

Основні переваги мови програмування C# включають:

1) Безпека

C# забезпечує високий рівень безпеки типів та управління пам'яттю завдяки механізмам типобезпеки та автоматичному збирачу сміття (garbage collector).

2) Ефективність

Завдяки JIT-компіляції (Just-In-Time), програми на C# можуть виконуватися швидко та ефективно на різних платформах.

3) Об’єктно-орієнтована спрямованість

C# підтримує всі основні концепції ООП, включаючи інтерфейси, абстракції, наслідування та поліморфізм.

4) Стійкість до помилок

Завдяки статичній типізації та механізмам обробки винятків, C# допомагає уникати багатьох типових помилок програмування.

5) Підтримка багатозадачності

C# має потужні засоби для реалізації багатозадачності, включаючи асинхронне програмування за допомогою ключових слів async та await.

6) Незалежність від архітектури

Завдяки використанню проміжної мови (CIL) та віртуальної машини CLR, програми на C# можуть виконуватися на різних апаратних архітектурах.

7) Доступ до інструментів та матеріалів

C# має широкий спектр інструментів розробки та обширну документацію.

8) Ефективність розробки

Завдяки розвиненій екосистемі та інструментам для налагодження, тестування і розгортання, розробка на C# є швидкою та ефективною.

9) Велика спільнота

C# має велику та активну спільноту розробників, що забезпечує підтримку та обмін знаннями.

Ключова мета C# полягає у забезпеченні безпеки від несанкціонованого доступу, обмежуючи можливість виклику глобальних методів чи доступу до системних ресурсів, що значно підвищує рівень безпеки програм. C# також підтримується численними безкоштовними ресурсами для навчання і вирішення проблем, доступними в інтернеті [11].

.NET – це потужна платформа для розробки програмного забезпечення, створена компанією Microsoft. Вона надає розробникам широкий спектр інструментів і бібліотек для створення різноманітних додатків, включаючи веб-додатки, десктопні програми, мобільні додатки, хмарні сервіси, ігри та багато іншого. Основні компоненти .NET включають .NET Framework, .NET Core, а також новітні версії платформи, починаючи з .NET 5, які об'єднують можливості обох попередніх версій в єдину крос-платформенну платформу.

.NET Framework – це оригінальна реалізація .NET, яка призначена для створення та виконання додатків на Windows. Вона включає в себе багатий набір бібліотек, інструментів та служб, що спрощують розробку програмного забезпечення. .NET Core, з іншого боку, є крос-платформенною реалізацією .NET, яка дозволяє розробляти додатки для Windows, macOS і Linux. Ця версія забезпечує високу продуктивність та підтримку сучасних технологій, що робить її популярною серед розробників, які створюють додатки для різних операційних систем.

З виходом .NET 5, Microsoft об'єднала можливості .NET Framework та .NET Core в єдину крос-платформенну платформу. Це спрощує розробку, оскільки розробникам тепер не потрібно обирати між двома версіями платформи. .NET 5 і новіші версії забезпечують високий рівень продуктивності, масштабованість і крос-платформенність, дозволяючи створювати додатки, що працюють на різних операційних системах.

ASP.NET є фреймворком для створення динамічних веб-додатків і веб-сервісів. ASP.NET Core, крос-платформенна версія ASP.NET, забезпечує

високу продуктивність і масштабованість, що робить її ідеальним вибором для створення сучасних веб-додатків. Xamarin, ще один компонент .NET, дозволяє розробляти мобільні додатки під iOS і Android, використовуючи C# і спільний код. Це дозволяє розробникам писати код один раз і використовувати його на різних мобільних платформах, що значно знижує витрати на розробку та підтримку додатків [9].

Хмарна платформа Azure, інтегрована з .NET, надає широкий спектр послуг для розробки, розгортання та керування додатками в хмарі. Це включає в себе можливості для створення масштабованих хмарних додатків, управління базами даних, аналітики та багато іншого. Завдяки глибокій інтеграції з .NET, розробники можуть легко використовувати ці сервіси для створення потужних та масштабованих хмарних рішень.

Основні характеристики .NET включають крос-платформеність, високу продуктивність, багатий набір бібліотек, об'єктно-орієнтований підхід і високу безпеку. Платформа підтримує об'єктно-орієнтоване програмування, що сприяє створенню масштабованого та модульного коду. Вбудовані механізми безпеки, включаючи управління пам'яттю та типобезпеку, забезпечують захист від виконання шкідливого коду. Інструменти розробки, такі як IDE Visual Studio та Visual Studio Code, надають розширені можливості для написання, налагодження та розгортання коду, що робить процес розробки більш ефективним і зручним.

Завдяки своїй гнучкості, масштабованості та широкому набору інструментів, .NET є популярною платформою для розробки програмного забезпечення у різних галузях. Вона забезпечує розробників всіма необхідними засобами для створення потужних, ефективних та безпечних додатків, що відповідають сучасним вимогам [7].

JSON (JavaScript Object Notation) є текстовим форматом для представлення та обміну структурованими даними, який вирізняється своєю легкістю для читання та запису як людиною, так і машиною. JSON заснований на підмножині синтаксису об'єктів JavaScript, але є незалежним від конкретної

мови програмування, що робить його універсальним засобом для передачі даних між різнорідними системами.

Основні характеристики JSON включають:

1. Простота та зрозумілість

JSON є текстовим форматом, що складається з пар "ключ-значення" та впорядкованих списків значень. Це робить його легко зрозумілим як для людей, так і для машинного аналізу.

2. Легкість та компактність

JSON має компактний синтаксис, що зменшує обсяг даних, які передаються через мережу, підвищуючи ефективність комунікаційних процесів.

3. Універсальність

Може використовуватися в будь-якій мові програмування, яка має бібліотеки для парсингу JSON наприклад JavaScript, Python.

4. Структурованість

JSON підтримує вкладені структури, що дозволяє представляти складні дані у вигляді вкладених об'єктів та масивів [6].

Visual Studio Community 2022 є безкоштовною, повнофункціональною та розширюваною інтегрованою середовищем розробки (IDE) від Microsoft, призначеною для індивідуальних розробників, студентів, відкритих проектів і навчальних закладів. Вона надає широкий спектр інструментів і можливостей для створення веб-додатків, десктопних програм, мобільних додатків, ігор та хмарних сервісів.

Visual Studio Community 2022 вирізняється інтуїтивно зрозумілим та настроюваним інтерфейсом користувача. Панелі, вікна та інструменти можна налаштовувати відповідно до потреб розробника, що забезпечує ефективну роботу. Доступні теми оформлення (темна, світла і синя) дозволяють адаптувати зовнішній вигляд середовища для комфортної роботи.

Ця IDE підтримує широкий спектр мов програмування, включаючи C#, C++, Python, JavaScript, TypeScript, HTML, CSS, SQL та багато інших. Це

дозволяє розробляти різноманітні додатки в одному середовищі, що підвищує продуктивність і знижує потребу у використанні додаткових інструментів.

Редактор коду у Visual Studio Community 2022 забезпечує інтелектуальну підтримку через систему IntelliSense, яка включає підказки та автозаповнення коду. Це спрощує написання коду, знижуючи кількість помилок та підвищуючи продуктивність. Підсвічування синтаксису, рефакторинг коду, навігація по коду та відладка роблять процес розробки більш ефективним.

Visual Studio Community 2022 включає потужні інструменти для відладки та тестування додатків. Відладчик дозволяє виконувати покрокову відладку, аналізувати значення змінних у реальному часі, встановлювати точки зупинки та вивчати стек викликів. IDE підтримує модульне тестування, інтеграцію з популярними фреймворками для тестування, такими як NUnit, MSTest, xUnit, та надає інструменти для автоматизованого тестування, що підвищує якість програмних продуктів.

Інтеграція з Git та GitHub забезпечує зручну роботу з системами контролю версій безпосередньо з Visual Studio. Це дозволяє легко створювати, клонувати, комітити та пушити репозиторії, вирішувати конфлікти та працювати з гілками коду. Visual Studio Community 2022 також інтегрована з хмарними платформами, такими як Microsoft Azure, що дозволяє легко створювати, розгортати та керувати хмарними додатками.

Visual Studio підтримує велику кількість розширень, що дозволяють додавати нові функціональні можливості та налаштовувати IDE відповідно до потреб розробника. Marketplace Visual Studio пропонує тисячі розширень для різних мов програмування, фреймворків та інструментів. Завдяки інтеграції з Xamarin, Visual Studio Community 2022 дозволяє розробляти крос-платформенні мобільні додатки для iOS та Android з використанням єдиного коду на C#. Це спрощує процес розробки та підтримки мобільних додатків.

Visual Studio Community 2022 підтримує розробку ігор з використанням Unity та Unreal Engine, що забезпечує можливість створення високоякісних 2D

та 3D ігор. Крім того, Visual Studio Community 2022 підтримує сучасні технології та фреймворки, такі як .NET 6, C# 10, ASP.NET Core, Blazor та інші, що дозволяє розробникам використовувати найновіші інструменти та методики для створення інноваційних додатків.

Visual Studio Community 2022 є безкоштовною для індивідуальних розробників, студентів, відкритих проектів та навчальних закладів, що робить її доступною для широкого кола користувачів. Інтеграція з іншими продуктами Microsoft, такими як Azure, Office 365 та GitHub, дозволяє використовувати широкий спектр можливостей та сервісів у єдиному середовищі розробки, що підвищує продуктивність та спрощує процес розробки. Велика та активна спільнота розробників забезпечує доступ до численних ресурсів, включаючи документацію, форуми, відеоуроки та блоги, що полегшує процес навчання та вирішення проблем.

Visual Studio Community 2022 є потужною та універсальною середовищем розробки, яка надає розробникам всі необхідні засоби для створення високоякісних додатків для різних платформ. Її безкоштовність, багатий набір можливостей, підтримка сучасних технологій та інтеграція з хмарними сервісами роблять її ідеальним вибором як для початківців, так і для досвідчених розробників.

2.3. Патерни проектування

Патерни проектування відіграють ключову роль у розробці веб-додатків, забезпечуючи структурованість, зрозумілість та масштабованість коду. Вони дозволяють розробникам застосовувати перевірені рішення для типових проблем, що виникають під час створення програмного забезпечення. Серед найпопулярніших патернів, які використовуються у веб-розробці, можна виділити Model-View-Controller (MVC), Model-View-ViewModel (MVVM), та VIPER. Кожен з цих патернів має свої переваги та недоліки, а також сферу

застосування, що робить їх незамінними інструментами в арсеналі сучасного розробника. Розглянемо більш детально кожен з них.

MVC (Model-View-Controller) - це патерн проектування, що використовується для створення організованих і структурованих додатків, особливо в контексті веб-розробки. Він забезпечує розділення логіки додатка на три взаємодіючі компоненти: Model, View, і Controller. Основна мета MVC - розділити внутрішнє представлення інформації від способу її подання та взаємодії з користувачем.

1) Model - відповідає за управління даними додатка.

2) View - відповідає за відображення даних користувачу.

3) Controller - виступає як посередник між Model і View.

Переваги MVC

1) Розділення обов'язків

MVC забезпечує чітке розділення обов'язків між різними компонентами додатка, що полегшує його розробку, тестування та підтримку.

2) Модульність

Кожен компонент (Model, View, Controller) може бути розроблений, протестований і підтримуваний незалежно, що сприяє більшій масштабованості та зрозумілості коду.

3) Гнучкість

Завдяки розділенню логіки додатка, MVC дозволяє легко змінювати та оновлювати окремі частини додатка без впливу на інші компоненти.

4) Повторне використання коду

Компоненти Model і View можуть бути повторно використані у різних частинах додатка або навіть у різних проектах [12].

MVVM (Model-View-ViewModel) - це патерн проектування, що використовується для створення структурованих і масштабованих додатків, особливо в контексті розробки з використанням сучасних фреймворків та бібліотек, таких як WPF, Angular, та React. Основна мета MVVM - розділення

представлення інтерфейсу користувача та бізнес-логіки, що полегшує розробку, тестування та підтримку додатка.

1) Model - відповідає за управління даними додатка. Це компоненти, які взаємодіють з базою даних, виконують бізнес-логіку і надають дані для ViewModel. Model містить чисту бізнес-логіку і не залежить від інших компонентів MVVM.

2) View - відповідає за відображення даних користувачу. Це інтерфейс користувача, який може бути реалізований за допомогою HTML, XAML, або іншого мови розмітки. View отримує дані з ViewModel і відображає їх користувачу. Крім того, View може містити анімації, стилі та інші елементи UI, але не містить бізнес-логіки.

3) ViewModel - виступає як посередник між View та Model. Він отримує дані з Model, обробляє їх і надає View у зручному для відображення форматі. ViewModel також обробляє події від View (наприклад, кліки на кнопки) і викликає відповідні методи Model. Основною перевагою ViewModel є те, що він дозволяє двосторонню прив'язку даних, що автоматично оновлює View при зміні даних у ViewModel і навпаки.

Переваги MVVM

- Розділення обов'язків:

MVVM забезпечує чітке розділення обов'язків між даними, логікою та інтерфейсом користувача, що полегшує розробку та підтримку додатка.

- Модульність

Кожен компонент (Model, View, ViewModel) може бути розроблений, протестований і підтримуваний незалежно, що сприяє більшій масштабованості та зрозумілості коду.

- Двостороння прив'язка даних

Однією з основних переваг MVVM є можливість двосторонньої прив'язки даних між View і ViewModel, що знижує необхідність вручну оновлювати інтерфейс при зміні даних.

- Повторне використання коду

ViewModel і Model можуть бути повторно використані у різних частинах додатка або навіть у різних проектах [13].

VIPER (View, Interactor, Presenter, Entity, Router) - це патерн проектування, що застосовується для розробки модульних і масштабованих додатків, особливо в контексті мобільної розробки, але також може бути адаптований для веб-додатків. Основна мета VIPER - розділення відповідальностей між різними компонентами для досягнення високої ступені модульності та тестованості коду.

1) View - відповідає за відображення даних і взаємодію з користувачем. Він отримує команди від Presenter і відображає відповідний контент.

2) Interactor - містить бізнес-логіку додатка. Він отримує запити від Presenter, обробляє їх, і повертає результат назад до Presenter.

3) Presenter - виступає як посередник між View та Interactor. Він отримує дані від Interactor і форматує їх для відображення у View. Також обробляє події з View і передає їх до Interactor.

4) Entity - містить модель даних. Це можуть бути об'єкти, що представляють дані додатка, такі як користувачі, продукти тощо.

5) Router - відповідає за навігацію між екранами додатка. Містить логіку переходів та навігаційних маршрутів [14].

Результатом є таблиця 2.1, що надає порівняння основних характеристик патернів MVC, MVVM та VIPER, показуючи їх переваги та недоліки, а також підказуючи, який патерн може бути найкращим вибором для конкретного проекту залежно від його вимог.

Таблиця 2.1

Порівня патернів проектування

Параметр	MVC (Model-View-Controller)	MVVM (Model-View-ViewModel)	VIPER (View-Interactor-Presenter-Entity-Router)
Основна мета	Розділення представлення даних та бізнес-логіки	Розділення представлення інтерфейсу користувача та бізнес-логіки з підтримкою двосторонньої прив'язки даних	Розділення відповідальностей для досягнення модульності та тестованості
Компоненти	Model, View, Controller	Model, View, ViewModel	View, Interactor, Presenter, Entity, Router
Відповідальність Model	Управління даними та бізнес-логікою	Управління даними та бізнес-логікою	Управління даними та бізнес-логікою
Відповідальність View	Відображення даних користувачу	Відображення даних користувачу	Відображення даних користувачу
Відповідальність Controller/Presenter / ViewModel	Controller обробляє запити від View і взаємодіє з Model	ViewModel обробляє події з View і взаємодіє з Model	Presenter обробляє запити від View, взаємодіє з Interactor і оновлює View
Двостороння	Ні	Так	Ні

прив'язка даних			
Модульність	Середня	Висока	Дуже висока
Тестованість	Середня	Висока	Дуже висока
Масштабованість	Середня	Висока	Дуже висока
Придатність для великих проектів	Середня	Висока	Дуже висока
Навігація між екранами	Обробляється в Controller	Обробляється окремо від ViewModel	Виконується Router
Складність реалізації	Помірна	Середня	Висока

Для реалізації веб-додатку було обрано VIPER завдяки його чіткому розділенню обов'язків між компонентами (View, Interactor, Presenter, Entity, Router), що підвищує модульність, тестованість і масштабованість додатків. Це забезпечує легку підтримку та розширення функціональності, оскільки кожен компонент має свою чітку роль. VIPER також сприяє покращеній читабельності коду та ефективній командній роботі, дозволяючи розробникам працювати над окремими частинами додатка без конфліктів. Центральне управління навігацією через Router робить навігацію більш передбачуваною і керованою, що є критичним для великих і складних проектів.

2.3. Висновок до другого розділу

У цьому розділі було проведено аналіз та обрано оптимальні засоби для реалізації веб-застосунку «Ресторан». Для розробки проекту було обрано мову програмування C# та платформу .NET, що забезпечили потужні можливості

для створення надійного та масштабованого додатку. JSON використовувався для зручного обміну даними між компонентами системи.

Середовище розробки Visual Studio Community 2022 надало широкий спектр інструментів для налагодження, тестування та розгортання програмного забезпечення, що значно спростило процес розробки.

Впровадження патерну проектування VIPER дозволило досягти високої модульності, тестованості та масштабованості додатка, забезпечуючи чітке розділення обов'язків між його компонентами. Таким чином, обрані технології та підходи дозволили створити сучасний веб-застосунок, що відповідає потребам користувачів та вимогам бізнесу у сфері надання кінематографічних послуг.

РОЗДІЛ 3. РОЗРОБКА ВЕБ-ЗАСТОСУНКУ «КІНОСЕРВІС»

3.1. Концепція розробки програмного додатку на VIPER

У сучасному світі розробка програмного забезпечення вимагає все більшої структурованості та модульності для забезпечення масштабованості, легкості підтримки та можливості повторного використання коду. VIPER (View, Interactor, Presenter, Entity, Router) – це архітектурний шаблон, який дозволяє розробникам програмних додатків досягати цих цілей, забезпечуючи чіткий розподіл обов'язків між компонентами. Використання VIPER сприяє підвищенню якості коду та зменшенню складності підтримки додатку, що є особливо важливим в умовах швидко змінюваних вимог ринку.

VIPER сприяє підвищенню безпеки додатку. Завдяки чітко визначеним межам між компонентами, зловмисникам стає складніше експлуатувати вразливості, що виникають через помилки у взаємодії між різними частинами додатку. Таким чином, вибір VIPER як архітектурного шаблону для розробки додатку є виправданим рішенням, яке забезпечує надійність, зручність підтримки та гнучкість розробки.

VIPER здатен забезпечити високу масштабованість проектів. Оскільки кожен компонент VIPER відповідає лише за свою частину функціональності, команди розробників можуть працювати незалежно одна від одної, зменшуючи ризик конфліктів та прискорюючи процес розробки. Це особливо важливо для великих проектів з численними функціями та складною логікою.

3.2 Структура проекту

Для реалізації даного проекту потрібно сформувати цілі розробки на основі архітектури Viper:

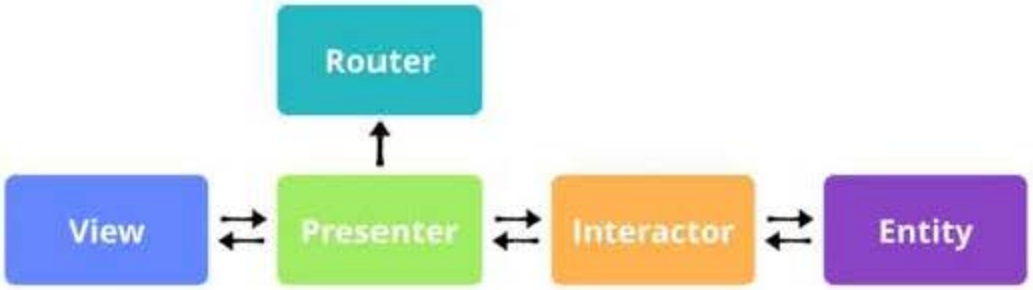


Рисунок 3.1 – Структура архітектури VIPER

VIPER поділяється на 5 чітко визначених компонентів View, Interactor, Presenter, Entity та Router. Кожен з цих компонентів має власні функції які повинен виконувати (рис. 3.1).

Спочатку потрібно почати з моделей, вони в структурі представлені як Entity. Це структури даних які використовуються виключно для зберігання даних та в подальшому додавання в базу даних.

Далі йде Interactor. Використовується компонент для обробки бізнес логіки та взаємодією з моделями даних Entity.

Наступним є Presenter. Створений для того, щоб виводити дані які надав право Interactor, тобто виведення колекцій та списків, або навпаки бути проміжним етапом при отриманні нових даних на запис.

Дані отримані з Presenter вже використовує Router. Це маршрутизатор між різними веб сторінками View, по іншому їх можна назвати контролерами. Також завдяки таким контролерам і отримуються дані які заніс користувач, або навпаки виводяться з Presenter [10].

Отже VIPER можна уявити як ланцюг компонентів, через які дані проходять та покроково обробляються. Головним завданням розробника це реалізувати архітектуру таким чином, щоб доступ до бізнес логіки не можливо було отримати безпосередньо просто взявши з Interactor, а побудувати систему безпеки де контролер може отримувати виключно ті функції які надає йому саме Presenter.

3.3 Інструменти розробки

Перш за все потрібно обрати зручне середовище розробки для веб додатку. Загалом, найкращим застосунком є Microsoft Visual Studio. Visual Studio є потужним інтегрованим середовищем розробки (IDE), яке підтримує розробку додатків на ASP.NET Core. Воно забезпечує розробників численними інструментами для написання, налагодження та тестування коду також він інтегрований разом з Git.

Рисунок 3.1 – Microsoft Visual Studio

Почати розробку веб-додатку за допомогою Visual Studio, є доволі легко. Відразу після встановлення користувач може до завантажити потрібні для роботи технології (рис. 3.2):

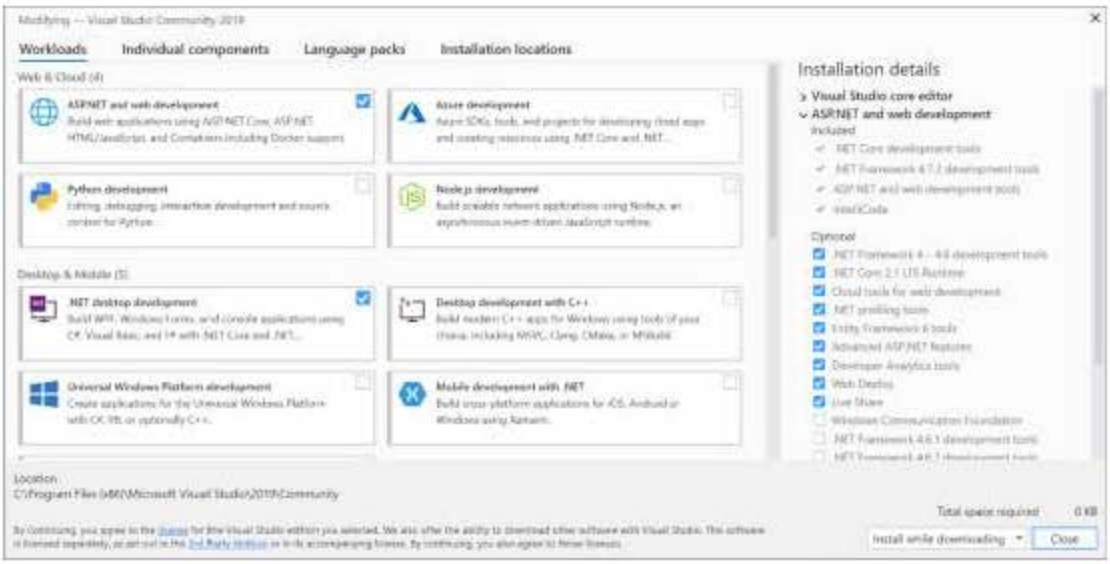


Рисунок 3.2 – До завантаження потрібних технологій

Visual Studio, також дозволяє встановлювати вузько направлені пакети для завдяки менеджеру пакетів NuGet (рис. 3.3).

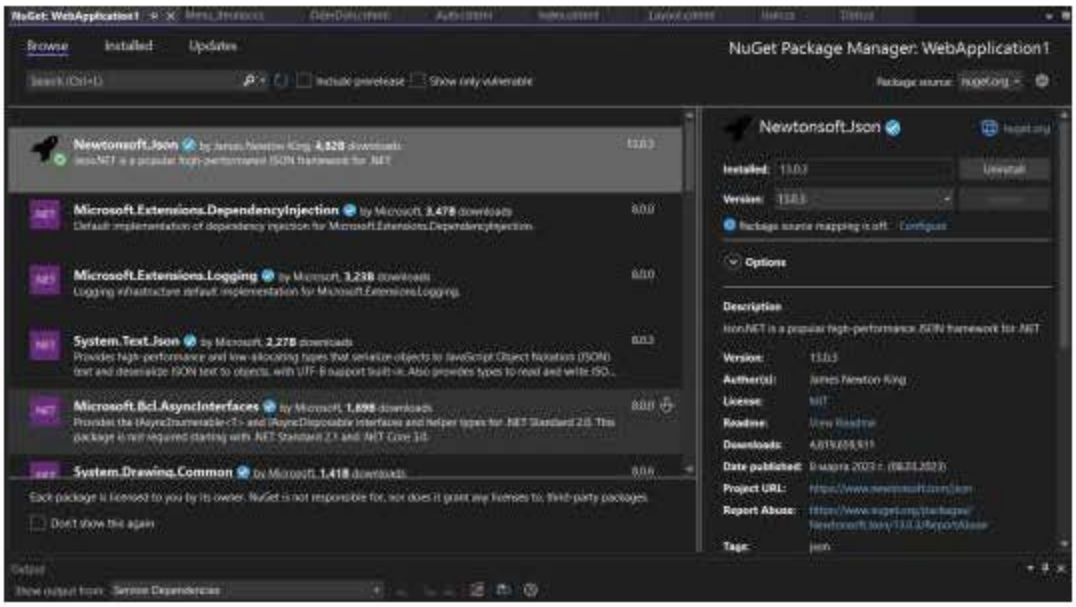


Рисунок 3.3 – Менеджер пакетів NuGet

NuGet – це менеджер пакетів для .NET, який дозволяє розробникам легко додавати, оновлювати та керувати бібліотеками та залежностями у своїх проектах. Використовуючи NuGet, розробники можуть швидко інтегрувати популярні бібліотеки, такі як Newtonsoft.Json для роботи з JSON, AutoMapper

для мапування об'єктів та багато інших, що значно прискорює процес розробки.

Варто зазначити й синтаксис для написання веб-сторінок на ASP.NET Core, який поєднує C# і HTML, дозволяючи створювати динамічні веб-сторінки з мінімальним обсягом коду. Дані сторінки створюватимуться з розширенням chtml замість html та дозволятимуть додавати функції чи конструкції з мови програмування C#.

Звичайно, що також варто зазначити мову програмування, яку не одноразово згадували - C#. C# це об’єктно-орієнтована мова програмування яка надає можливість створювати окремі об’єкти – класи, які мають свої поля та функції. Це важлива складова на якій буде ґрунтуватися розробка проекту. Крім створення класів, використовуватиметься інкапсуляція, за допомогою якого, будуть обмежуватися деякі функції для виведення даних, та покращить в цілому безпеку даних проекту (рис. 3.4).

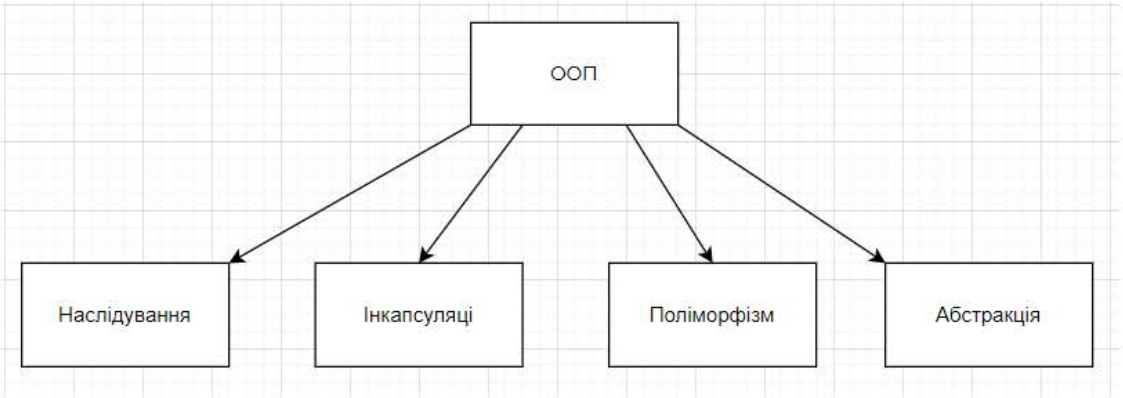


Рисунок 3.4 - ООП

3.4 Розробка проекту

Для розробки потрібно визначити класи які відповідають за свою роль в VIPER. Щоб структурувати проект потрібно створити відповіді папки під кожен компонент патерну в середині проекту:

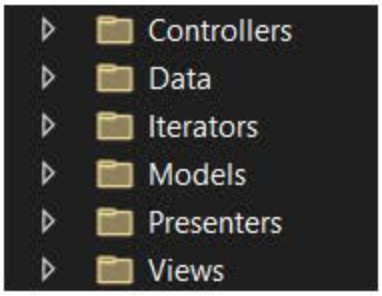


Рисунок 3.5 – Загальна структура проекту в Visual Studio

Загальна структура проекту, якщо його структурувати відповідно до патерну відповідатиме наступним чином (рис. 3.6):

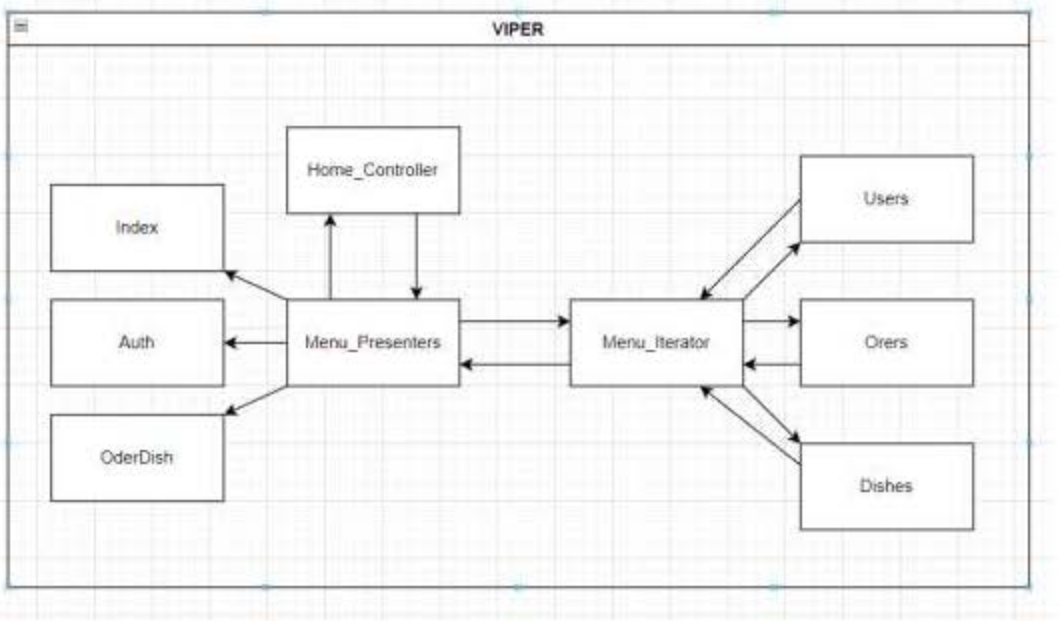


Рисунок 3.6 – Структура проекту

На ілюстрації можна спостерігати назву компонентів та їх кількість. Отже в проекті присутні наступні компоненти: 3 представлення, 3 моделі та по одному класу контролера, виведення та бізнес логіки [15]. В Visual Studio, проект має наступну структуру класів:

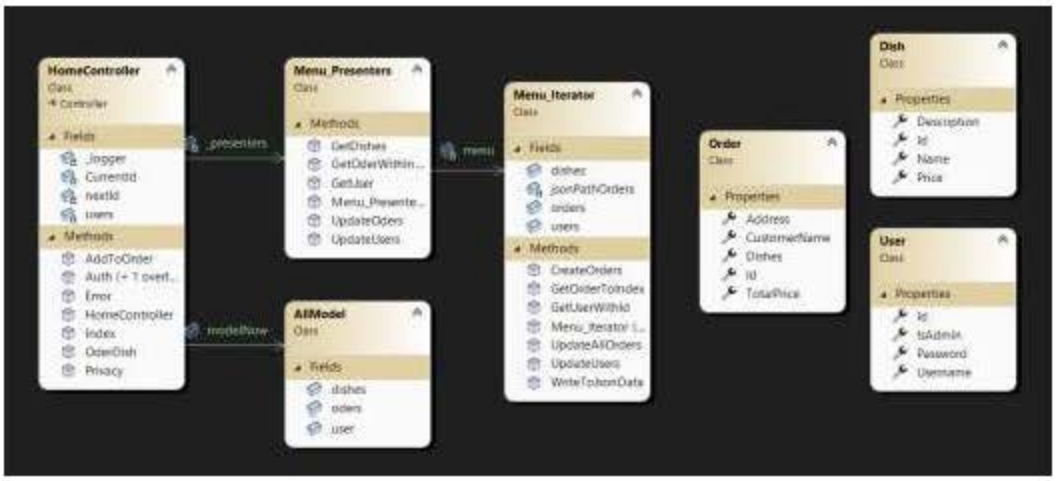


Рисунок 3.7 – Діаграма класів проекту

Крім даних класів в проект входять системний клас, який потребує налаштування, від нього залежить робота проекту та початкове представлення:

```
public class Program
{
    public static void Main(string[] args)
    {
        var builder = WebApplication.CreateBuilder(args);

        // Add services to the container.
        builder.Services.AddControllersWithViews();

        var app = builder.Build();

        // 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
            app.UseHsts();
        }

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

        app.UseRouting();

        app.UseAuthorization();

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

        app.Run();
    }
}
```

Рисунок 3.8 – клас Program.cs

Даний клас визначає конфігурації, сервіси та інші налаштування. Додає підтримку контролерів з представленнями (Views) до контейнера служб: builder.Services.AddControllersWithViews(). Наступні функції app:

- UseHttpsRedirection: Перенаправляє всі HTTP-запити на HTTPS.
- UseStaticFiles: Дозволяє обробляти статичні файли, такі як HTML, CSS, JavaScript тощо.
- UseRouting: Додає підтримку маршрутизації.
- UseAuthorization: Додає підтримку авторизації.

Рядок app.MapControllerRoute(name: "default", pattern: "{controller=Home}/{action=Index}/{id?}"), визначає з якого представлення буде завантажуватися проект на початку.

Далі потрібно почати розгляд з класів, що відповідають за зберігання даних - Entity. Загалом їх кількість 3:

Рисунок 3.9 – Класи для зберігання даних

Клас Users, використовується для авторизації або реєстрації користувача, має наступні параметри:

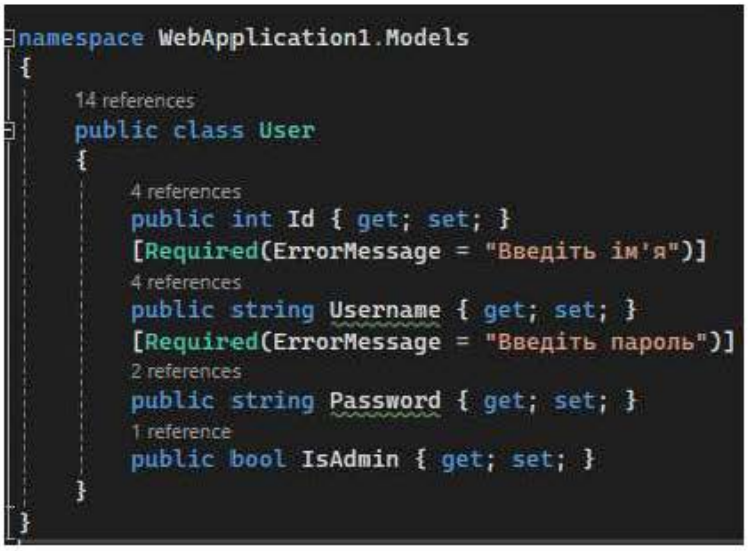


Рисунок 3.10 – Клас Users

- public int Id { get; set; } - Ідентифікатор користувача. Це унікальне число, яке використовується для ідентифікації кожного користувача в базі даних.
- [Required(ErrorMessage = "Введіть ім'я")] public string Username {get;set;} - Ім'я користувача. Ця властивість є обов'язковою, що вказано атрибутом [Required]. Якщо значення не буде введено, користувач побачить повідомлення "Введіть ім'я".
- [Required(ErrorMessage = "Введіть пароль")] public string Password { get; set;} - Пароль користувача. Також є обов'язковою властивістю з атрибутом [Required].
- public bool IsAdmin { get; set; } - Вказує, чи є користувач адміністратором.

Використання атрибутів валідації, таких як [Required], дозволяє автоматично перевіряти введені дані і відображати повідомлення про помилки, якщо дані не відповідають вимогам. Це допомагає забезпечити цілісність та правильність даних, що вводяться користувачами.

Наступний клас - Dish. Даний клас представляє модель страви, яка буде використовуватися в додатку для ресторанних послуг (рис. 3.11).

```
namespace WebApplication1.Models
{
    18 references
    public class Dish
    {
        6 references
        public int Id { get; set; }
        6 references
        public string Name { get; set; }
        8 references
        public decimal Price { get; set; }
        4 references
        public string Description { get; set; }
    }
}
```

Рисунок 3.11 – Клас Dish

Він містить наступні властивості:

- public int Id { get; set; } - Ідентифікатор страви.
- public string Name { get; set; } - Назва страви.
- public decimal Price { get; set; } - Ціна страви.
- public string Description { get; set; } - Опис страви

Останній клас Entity - Order. Даний клас використовується для зберігання замовлення.

```
namespace WebApplication1.Models
{
    14 references
    public class Order
    {
        5 references
        public int Id { get; set; }
        10 references
        public List<Dish> Dishes { get; set; }
        3 references
        public string CustomerName { get; set; }
        3 references
        public string Address { get; set; }
        7 references
        public decimal TotalPrice { get; set; }
    }
}
```

Рисунок 3.12 – Клас Order

Містить наступні поля:

- public int Id { get; set; } - Ідентифікатор замовлення
- public List<Dish> Dishes { get; set; } - Список страв, включених до замовлення.

- public string CustomerName { get; set; } – ім'я клієнта який зробив замолення
- public string Address { get; set; } – адреса замовлення
- public decimal TotalPrice { get; set; } – загальна вартість замовлення

Після класів Entity, наступним класом є Menu_Iterator. Даний клас відповідає за бізнес логіку додатку та має доступ до всіх класів Entity.

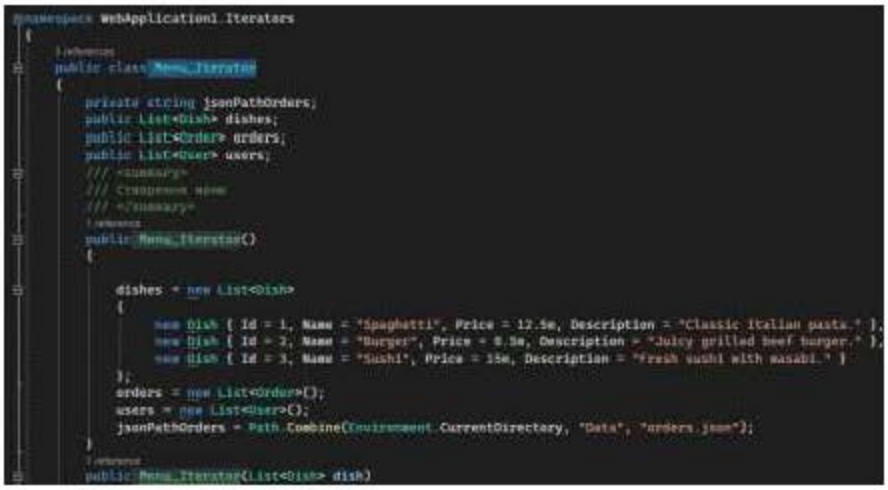


Рисунок 3.13 – клас Menu_Iterator

Розглянемо важливі функції даного класу. Перш за все потрібно звернути увагу на конструктор класу:

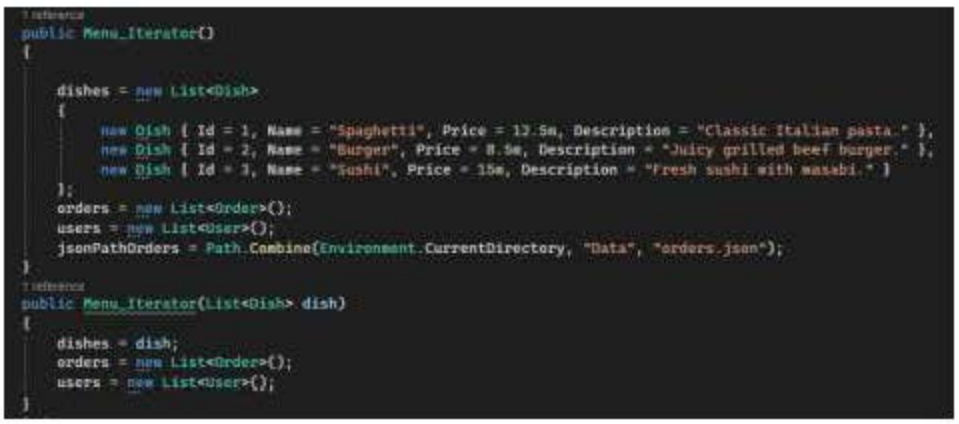


Рисунок 3.14 – Конструктор класу Menu_Iterator

Клас Menu_Iterator має два конструктори. Перший конструктор ініціалізує меню з попередньо визначеним набором страв, а його перезавантаження дозволяє ініціалізувати меню з власним набором страв, що будуть в наявності в ресторані.

Наступні 2 функції CreateOrders та UpdateUsers, використовуються для створення нового замовлення та оновлення списку користувачів (рис. 3.15):

```
0 references
public void CreateOrders(Order order)
{
    orders.Add(order);
}
1 reference
public void UpdateUsers(User user)
{
    users.Add(user);
}
1 reference
```

Рисунок 3.15 – Функції CreateOrders та UpdateUsers

Наступна функція GetUserWithId(int id). Використовуються дана функція для отримання об’єкту конкретного коримтувача:

```
1 reference
public User GetUserWithId(int id)
{
    for (int i = 0; i< users.Count; i++)
    {
        if (users[i].Id == id)
        {
            return users[i];
        }
    }
    return null;
}
```

Рисунок 3.16 – Функція GetUserWithId

Функція GetUserWithId приймає ціле число id як параметр і шукає користувача з цим ідентифікатором у списку users. Вона перебирає кожен

елемент у списку users за допомогою циклу for. Якщо вона знаходить користувача, ідентифікатор якого співпадає з переданим параметром id, функція повертає цього користувача. Якщо жоден користувач з таким ідентифікатором не знайдений, функція повертає null. Ця функція дозволяє швидко знайти конкретного користувача у списку за його унікальним ідентифікатором.

Далі функція GetOrderToIndex(int indexUser), також виконує пошук конкретного замолвення:

```
public Order GetOrderToIndex(int indexUser)
{
    string file = File.ReadAllText(jsonPathOrders);
    if(file != "")
    {
        orders = JsonConvert.DeserializeObject<List<Order>>(File.ReadAllText(jsonPathOrders));
    }

    for (int i = 0; i < orders.Count; i++)
    {
        if (orders[i].Id == indexUser)
        {
            return orders[i];
        }
    }

    return null;
```

Рисунок 3.17 – Функція GetOrderToIndex

Функція GetOrderToIndex приймає ціле число indexUser як параметр і шукає замовлення з цим ідентифікатором у списку orders. Спочатку функція читає вміст файлу за шляхом jsonPathOrders у рядок file. Якщо цей рядок не порожній, вміст файлу десеріалізується у список замовлень (orders) за допомогою JsonConvert.DeserializeObject<List<Order>>. Після цього функція перебирає кожен елемент у списку orders за допомогою циклу for. Якщо вона знаходить замовлення, ідентифікатор якого співпадає з переданим параметром indexUser, функція повертає це замовлення. Якщо жодне замовлення з таким ідентифікатором не знайдене, функція повертає null. Таким чином, функція дозволяє завантажувати замовлення з файлу та шукати конкретне замовлення за його ідентифікатором

Крім цього в класі присутня функція для запису файлів json:

```
public void WriteToJsonData()
{
    File.WriteAllText(jsonPathOrders, JsonConvert.SerializeObject(orders, Formatting.Indented));
}
```

Рисунок 3.18 – Функція WriteToJsonData

Остання функція даного класу це UpdateAllOrders. Дана функція приймає параметри int IndexUser, int IndexDish. Вони використовуються для подальшої ідентифікації страви та замовлення. Перша частина функції виконує завантаження даних з файлу orders.json та отримання конкретної страви, що замовив користувач через цикл for, завдяки якому перебираються елементи зі списку dishes:

```
public void UpdateAllOrders(int IndexUser, int IndexDish)
{
    string file = File.ReadAllText(jsonPathOrders);
    if (file != "")
    {
        orders = JsonConvert.DeserializeObject<List<Order>>(File.ReadAllText(jsonPathOrders));
    }
    //Отримати страву яку замовив користувач
    Dish dish = new Dish();
    for(int i = 0; i< dishes.Count; i++)
    {
        if(IndexDish == dishes[i].Id)
        {
            dish = dishes[i];
        }
    }
```

Рисунок 3.19 – Перша частина функції UpdateAllOrders

Друга частина функції виконується в тому випадку, якщо відсутні будь-які замовлення. В даному випадку створюється повністю нове замовлення, та заповнюються відповідні параметри: адреса замовлення, новий список страв, та загальну ціну всього замовлення (рис. 3.20):

```
if(orders.Count == 0)
{
    //Коли замовлень нема то створюємо нове замовлення
    Order order = new Order();
    order.Id = IndexUser;
    order.Dishes = new List<Dish>();
    order.Dishes.Add(dish);
    // встановлюємо ім'я
    for(int i = 0; i< users.Count; i++)
    {
        if (users[i].Id == IndexUser)
        {
            order.CustomerName = users[i].Username;
        }
    }
    order.Address = "Адреса ресторану";
    order.TotalPrice = 0;
    //Розраховуємо ціну
    foreach (Dish d in order.Dishes)
    {
        order.TotalPrice += d.Price;
    }
    //Додавання нового замолвення
    orders.Add(order);
}
else
```

Рисунок 3.20 – Друга частина функції UpdateAllOrders

Остання частина функції виконується в тому випадку коли вже створені деякі замовлення. Підчас роботи даної частини додається параметр bool OrderToUserHave. Він визначає, чи є замовлення, що відповідає параметру IndexUser. Якщо знайдено таке замовлення то функції змінює вже існуюче замовлення, додаючи нову страву (рис. 3.21):

```
bool OrderToUserHave = false; // Вказує на наявність конкр
//Якщо декілька замолвень вже є
for(int i = 0; i< orders.Count; i++)
{
    if (orders[i].Id == IndexUser)
    {
        OrderToUserHave = true;
        orders[i].Dishes.Add(dish);
        orders[i].TotalPrice = 0;
        //Перераховуємо загальну ціну знову
        foreach ( Dish d in orders[i].Dishes)
        {
            orders[i].TotalPrice += d.Price;
        }
    }
    else
    {
        //Тобто замолвення для цього користувача немає
        OrderToUserHave = false;
    }
}
```

Рисунок 3.21 – 3 частина функції UpdateAllOrders

В іншому випадку якщо подібне замовлення не знайдене, то створюється повністю нове замовлення з усіма відповідним параметрами:

```
if(!OrderToUserHave) //Якщо замовлення для цього користувача немає
{
    //Створюємо нове замолення для цього користувача
    Order order = new Order();
    order.Id = IndexUser;
    order.Dishes = new List<Dish>();
    order.Dishes.Add(dish);
    // встановлюємо ім'я
    for (int i = 0; i < users.Count; i++)
    {
        if (users[i].Id == IndexUser)
        {
            order.CustomerName = users[i].Username;
        }
    }
    order.Address = "Адреса ресторану";
    order.TotalPrice = 0;
    //Розраховуємо ціну
    foreach (Dish d in order.Dishes)
    {
        order.TotalPrice += d.Price;
    }
    //Додавання нового замолвення
    orders.Add(order);
}
```

Рисунок 3.22 – Остання частина функції UpdateAllOrders

Далі наступний клас Menu_Presenters. Клас Menu_Presenters є частиною простору імен WebApplication1.Presenters і слугує як "Presenter" у архітектурі VIPER, що взаємодіє з класом Menu_Iterator для управління даними меню, користувачів і замовлень. Цей клас інкапсулює логіку доступу до даних і виступає посередником між інтерфейсом користувача та бізнес-логікою (рис. 3.23).

```
using WebApplication1.Models;

namespace WebApplication1.Presenters
{
    4 references
    public class Menu_Presenters
    {
        private Menu_Iterator menu;

        0 references
        public Menu_Presenters(List<Dish> dishes)
        {
            menu = new Menu_Iterator(dishes);
        }
        1 reference
        public Menu_Presenters()
        {
            menu = new Menu_Iterator();
        }
        1 reference
        public void UpdateUsers(User user)
        {
            menu.UpdateUsers(user);
        }
        2 references
        public User GetUser(int id)
        {
            return menu.GetUserWithId(id);
        }
        1 reference
        public List<Dish> GetDishes()
        {
            return menu.dishes;
        }
        3 references
        public Order GetOderWithIndex(int indexID)
        {
            return menu.GetOrderToIndex(indexID);
        }

        1 reference
        public void UpdateOders(int indexUser, int indexDish)
        {
            menu.UpdateAllOrders(indexUser, indexDish );
        }
    }
```

Рисунок 3.23 – Клас Menu_Presenters

Єдиним полем класу є private Menu_Iterator menu, використовується для взаємодії з даними меню, користувачів та замовлень.

Далі створюються 2 конструктори класу Menu_Presenters, що відрізняються наявністю списку страв в якості параметра функції (рис. 3.24):

```
private Menu_Iterator menu;

0 references
public Menu_Presenters(List<Dish> dishes)
{
    menu = new Menu_Iterator(dishes);
}
1 reference
public Menu_Presenters()
{
    menu = new Menu_Iterator();
}
1 reference
```

Рисунок 3.24 – Конструктори класу

Після конструкторів йдуть створені функції, які просто перевизначають вже існуючі функції з класу Menu_Iterator:

- UpdateUsers – для оновлення інформації про користувача.
- GetUser – знаходить користувача за його ідентифікатором.
- GetDishes – використовуються для отримання списку страв
- GetOderWithIndex – повертає замовлення за іденпифікатором
- UpdateOders – оновлює весь список замовлень

```
public void UpdateUsers(User user)
{
    menu.UpdateUsers(user);
}
2 references
public User GetUser(int id)
{
    return menu.GetUserWithId(id);
}
1 reference
public List<Dish> GetDishes()
{
    return menu.dishes;
}
3 references
public Order GetOderWithIndex(int indexID)
{
    return menu.GetOrderToIndex(indexID);
}

1 reference
public void UpdateOders(int indexUser, int indexDish)
{
    menu.UpdateAllOrders(indexUser, indexDish );
}
```

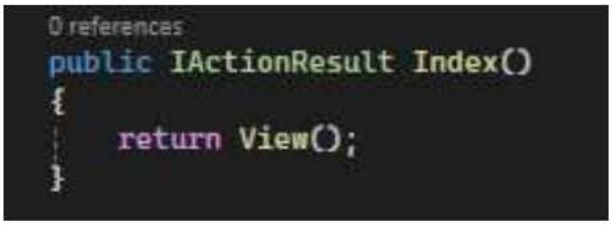
Рисунок 3.25 – Перевизначені функції Menu_Iterator

Останній клас, це контроллер HomeController, відповідає за обробку запитів користувачів у веб-додатку. Він взаємодіє з моделями та презентерами для надання відповідей на запити та передачу даних між представленями.

```
3 references
public class HomeController : Controller
{
    public AllModel modelNow;//Тільки для передачі даних між представленями
    private static List<User> users = new List<User>();
    private static int nextId = 0;
    private int CurrentId = 0;
    Menu_Presenters _presenters;
    private readonly ILogger<HomeController> _logger;

    0 references
    public HomeController(ILogger<HomeController> logger)
    {
        modelNow = new AllModel();
        modelNow.oders = new Order();
        modelNow.oders.Dishes = new List<Dish>();
        _logger = logger;
        _presenters = new Menu_Presenters();
    }

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

    0 references
    public IActionResult Privacy()
    {
        return View();
    }
    [HttpGet]
    0 references
    public IActionResult Auth()
    {
        return View();
    }
    [HttpPost]
    0 references
    public IActionResult Auth(User user)
    {
```

Рисунок 3.26 – Клас HomeController

Звичайна функції маршрутизації має назву функції = назва представлення, та зазвичай повертає return View(), щоб завантажити дану сторінку.



```
0 references
public IActionResult Index()
{
    return View();
}
```

Рисунок 3.27 – Звичайна функція для відкриття представлення

Однак в даному класі є функції які мають HttpGet та HttpPost є атрибути. Атрибут HttpGet вказує, що метод дії обробляє HTTP-запити типу GET. HTTP GET-запити використовуються для запиту ресурсів з сервера, таких як веб-сторінки або дані. Ці запити не повинні змінювати стан сервера (ніяких змін даних, оновлень або видалень).

Атрибут HttpPost вказує, що метод дії обробляє HTTP-запити типу POST. HTTP POST-запити використовуються для надсилання даних на сервер з метою створення або оновлення ресурсів.

Для прикладу є HttpGet та HttpPost запит на авторизацію користувача, в якому HttpGet завантажує сторінку з формою, а HttpPost запит вже приймає дані та записує їх (рис. 3.28):

```
public IActionResult Auth()
{
    return View();
}
[HttpPost]
0 references
public IActionResult Auth(User user)
{
    if (ModelState.IsValid)
    {
        user.Id = nextId;
        CurrentId = nextId;
        _presenters.UpdateUsers(user);
        Console.WriteLine(users.Count);
        nextId++;
        return RedirectToAction("OderDish");
    }
    return View(user);
}
```

Рисунок 3.28 - HttpGet та HttpPost запити

Аналогічно й працюють запити на створення замовлень при виборі страви, при натисканні на кнопку (рис. 3.29):

```
0 references
public IActionResult OderDish()
{
    modelNow.dishes = _presenters.GetDishes();
    modelNow.user = _presenters.GetUser(CurrentId);
    if(_presenters.GetOderWithIndex(CurrentId) != null)
    {
        modelNow.oders = _presenters.GetOderWithIndex(CurrentId);
    }
    return View(modelNow);
}
[HttpPost]
0 references
public IActionResult AddToOrder(int dishId)
{
    _presenters.UpdateOders(CurrentId, dishId);
    modelNow.user = _presenters.GetUser(CurrentId);
    modelNow.oders = _presenters.GetOderWithIndex(CurrentId);
    return RedirectToAction("OderDish");
}
```

Рисунок 3.29 - HttpGet та HttpPost на замовлення

Для коректності роботи таких запитів, потрібно правильно визначити методи вже в представлені вказуючи на який запит працюватиме кнопка:

```
<form method="get" action="/Home/Auth">
    <button type="submit">Go to Authorization</button>
</form>
```

Рисунок 3.30 – створення форми із запитом post

При передачі даних між представленнями зручно користуватися проміжним класом, який може мати декілька об'єктів:

```
{
    5 references
    public class AllModel
    {
        public User user;
        public List<Dish> dishes;
        public Order oders;
    }
}
```

Рисунок 3.31 – Клас AllModel

Даний клас використовуються для передачі даних під час роботи представлення з меню. Для цього використовуються ініціалізація типу: @model AllModel. Далі вже в середині представлення можна використовувати цю модель для виведення всіх страв в меню (рис. 3.32):

```
@foreach (var dish in Model.dishes)
{
    <div class="col-md-4">
        <div class="card">
            <img src="@Url.Content("~/images/" + dish.Id + ".jpg")" class="card-img-top" alt="@dish.Name">
            <div class="card-body">
                <h5 class="card-title">@dish.Name</h5>
                <p class="card-text">@dish.Description</p>
                <p class="card-text"><strong>Price:</strong> $@dish.Price</p>
                <form asp-action="AddToOrder" method="post">
                    <input type="hidden" name="dishId" value="@dish.Id" />
                    <button type="submit" class="btn btn-primary">Add to Order</button>
                </form>
            </div>
        </div>
    </div>
}
```

Рисунок 3.32 – Виведення страв через цикл @foreach

За допомогою синтаксису @, можна додавати певні конструкції з мови програмування, в даному випадку використовувалися цикли для перебору елементів списку страв.

Для роботи веб додатку було створено 3 представлення. Перше представлення це проста форма в якій є лише кнопка для переходу на наступні сторінку, та має декілька рядків тексту.

```
    </style>
</head>
<body>
    <div class="card">
        <div class="menu-header">
            <h1>Welcome to Our Restaurant</h1>
            <p>Experience the best cuisine with a delightful ambiance.</p>
        </div>
        <div class="info-section">
            <p><strong>Operating Hours:</strong> Mon - Fri: 9am - 10pm | Sat - Sun: 10am - 11pm</p>
            <p>
                <a href="https://www.facebook.com" target="_blank" class="fa fa-facebook"></a>
                <a href="https://www.twitter.com" target="_blank" class="fa fa-twitter"></a>
                <a href="https://www.instagram.com" target="_blank" class="fa fa-instagram"></a>
            </p>
            <p><strong>Contact Us:</strong> +1 (234) 567-890 | info@restaurant.com</p>
        </div>
        <div class="auth-button">
            <form method="get" action="/Home/Auth">
                <button type="submit">Go to Autherization</button>
            </form>
        </div>
        <div class="footer">
            <p>&copy; 2024 Our Restaurant. All rights reserved.</p>
        </div>
    </div>
    <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
    <script src="https://cdn.jsdelivr.net/npm/popperjs/core@2.5.4/dist/umd/popper.min.js"></script>
    <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
</body>
</html>
```

Рисунок 3.33 – Розмітка початкової сторінки

Наступне представлення, це форма авторизації користувача, в які потрібно занести дані, щоб ввійти в систему:

```
</head>
<body>
    <div class="card">
        <h2>Authorization</h2>
        <form asp-action="Auth" method="post">
            <div asp-validation-summary="All" class="text-danger"></div>
            <div class="form-group">
                <label for="Username">Username:</label>
                <input type="text" id="Username" asp-for="Username" class="form-control" />
                <span asp-validation-for="Username" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label for="Password">Password:</label>
                <input type="password" id="Password" asp-for="Password" class="form-control" />
                <span asp-validation-for="Password" class="text-danger"></span>
            </div>
            <div class="form-group form-check">
                <input type="checkbox" id="IsAdmin" asp-for="IsAdmin" class="form-check-input" />
                <label class="form-check-label" for="IsAdmin">Is Admin</label>
            </div>
            <button type="submit" class="btn btn-primary">Submit</button>
        </form>
    </div>
    <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
    <script src="https://cdn.jsdelivr.net/npm/popperjs/core@2.5.4/dist/umd/popper.min.js"></script>
    <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
</body>
</html>
```

Рисунок 3.34 – Розмітка сторінки авторизації

Останнє представлення – це сторінка замовлень, в якій користувач може обрати страву та додати її до власного замовлення:

```
<body>
    <div class="container">
        <div class="menu-header">
            <h1>Our Menu</h1>
            <p>Select your favorite dishes and add them to your order.</p>
        </div>
        <div class="row">
            @foreach (var dish in Model.dishes)
            {
                <div class="col-md-4">
                    <div class="card">
                        <img src="@Url.Content("~/images/" + dish.Id + ".jpg")" class="card-img-top" alt="@dish.Name">
                        <div class="card-body">
                            <h5 class="card-title">@dish.Name</h5>
                            <p class="card-text">@dish.Description</p>
                            <p class="card-text"><strong>Price:</strong> $@dish.Price</p>
                            <form asp-action="AddToOrder" method="post">
                                <input type="hidden" name="dishId" value="@dish.Id" />
                                <button type="submit" class="btn btn-primary">Add to Order</button>
                            </form>
                        </div>
                    </div>
                </div>
            }
            <h2>Замовлення</h2>
            <table class="table table-striped">
                <thead>
                    <tr>
                        <th>Order ID</th>
                        <th>Customer Name</th>
                        <th>Address</th>
                        <th>Total Price</th>
                        <th>Dishes</th>
                    </tr>
                </thead>
                <tbody>
                    <tr>
                        <td>@Model.oders.Id</td>
                        <td>@Model.oders.CustomerName</td>
                        <td>@Model.oders.Address</td>
                        <td>@Model.oders.TotalPrice</td>
                        <td>
                            <ul>
                                @foreach (var dish in Model.oders.Dishes)
                                {
                                    <li>@dish.Name - @dish.Price $</li>
                                }
                            </ul>
                        </td>
                    </tr>
                </tbody>
            </table>
        </div>
    </div>
```

Рисунок 3.35 – Розмітка сторінки з замовленням

3.5 Тестування проекту

Потрібно перевірити весь функціонал та роботу патерну VIPER, шляхом перегляду всіх сторінок та введенням відповідних даних, які потребує додаток.

Завантажимо проект у веб-браузері. Користувач повинен бачити стартовий екран, в якому є по центру є заголовок «Welcome to Our Restaurant», час роботи закладу та кнопка на яку потрібно натисну, щоб пройти авторизацію (рис. 3.36):

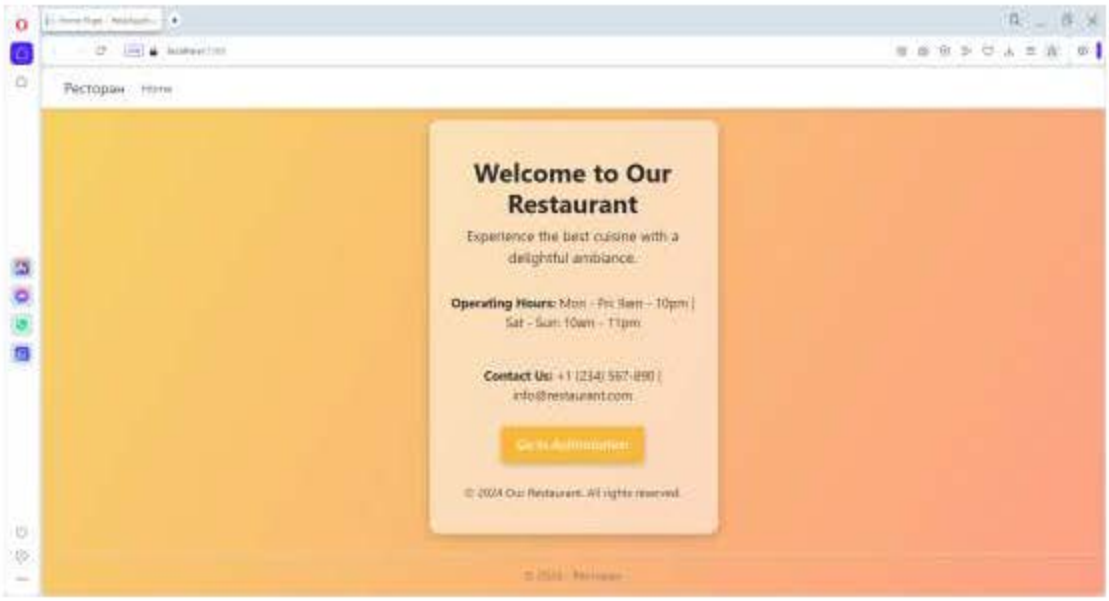


Рисунок 3.36 – Стартове вікно додатку

Далі натискаючи на кнопку користувач може перейти на наступні сторінку авторизації:

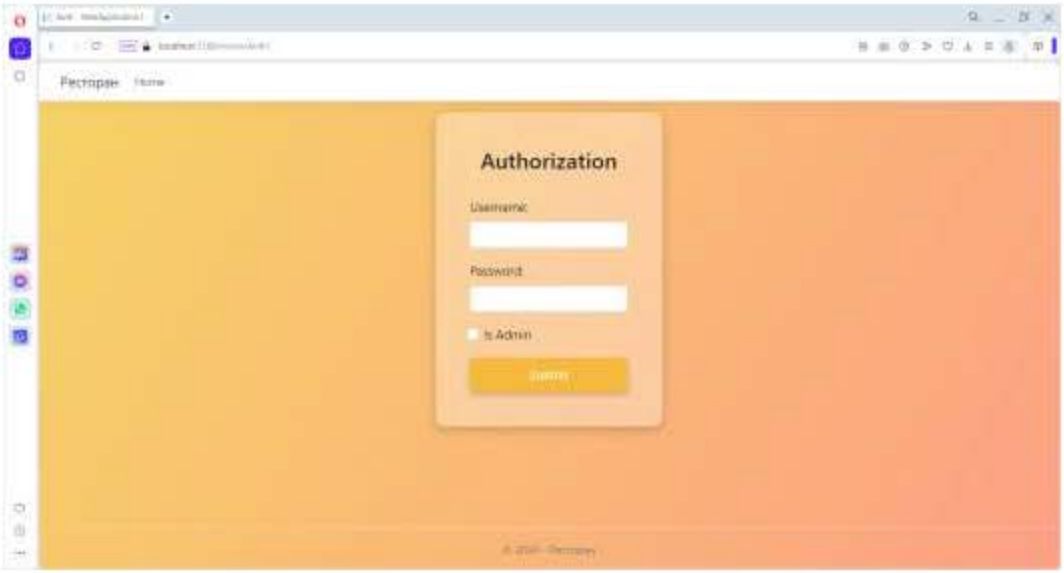


Рисунок 3.37 – Сторінка авторизації

Якщо користувач спробує нічого не ввести, то додаток відреагує на це відповідним повідомленням (рис. 3.38):

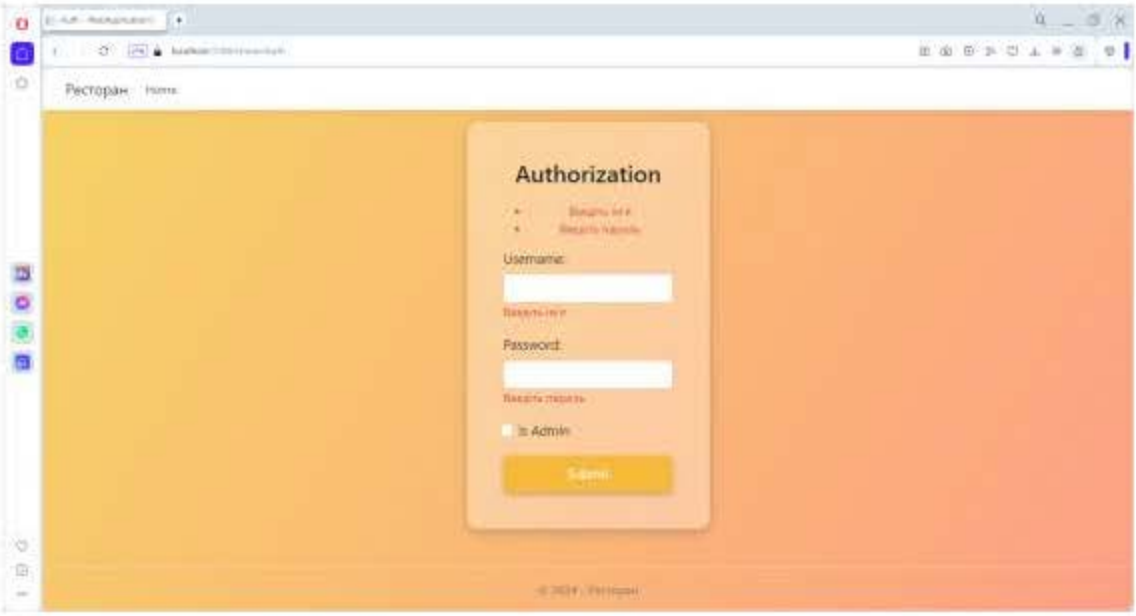


Рисунок 3.38 – Повідомлення про не вірний формат введення

Якщо ввести дані правильно, то користувач повинен потрапити в меню ресторану (рис. 3.39):

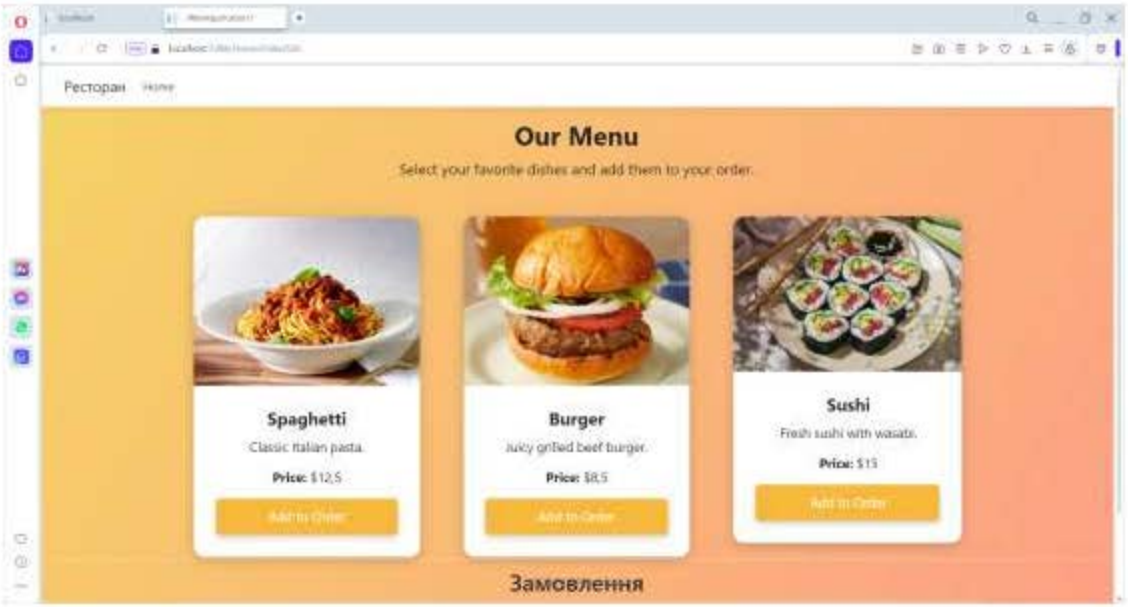


Рисунок 3.39 – Меню ресторану

При наведенні на певну страву з'являється анімація збільшення картки зі стравою:

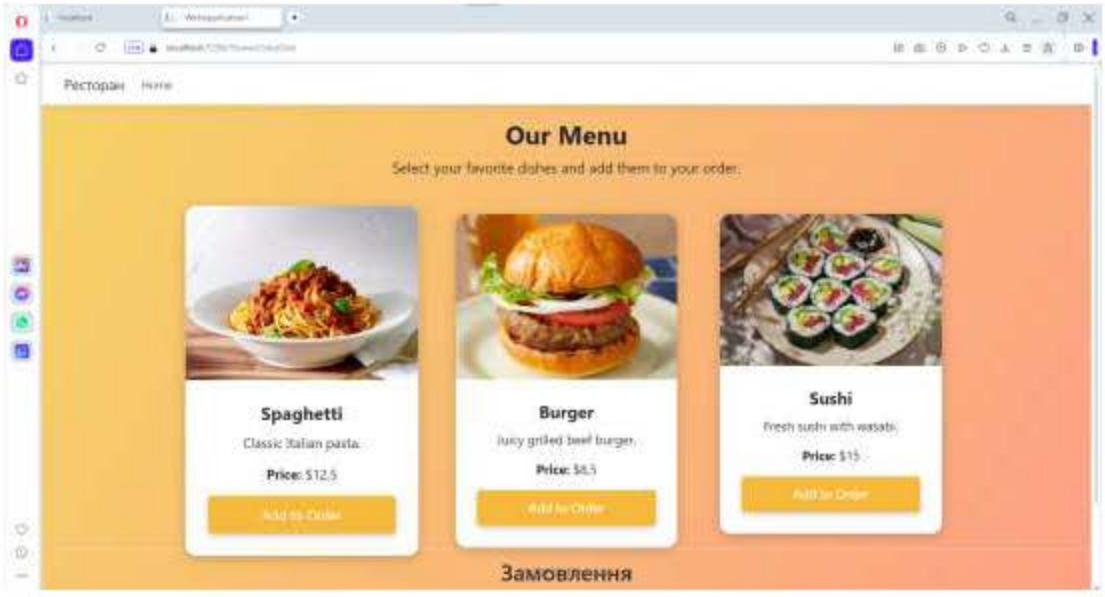


Рисунок 3.40 – Анімація збільшення

Щоб оформити замовлення потрібно натиснути кнопку «Add to Order». Після цього повинно з'явитися нижче таблиця з замовленням:

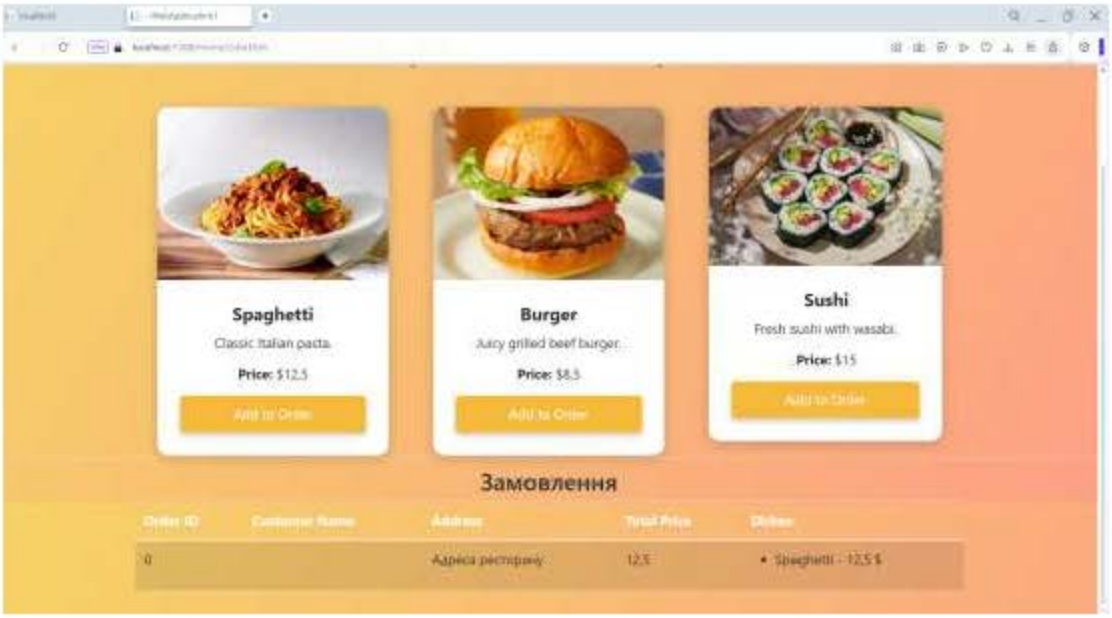


Рисунок 3.41 – Додавання замовлення

Додаємо ще декілька замовлень, щоб отримати загальну суму замовлення на декілька страв:

Рисунок 3.42 – Обробка декількох страв

Після додавання декількох страв, можна відслідкувати загальну суму замовлення з декількох страв. Результати роботи додатку зберігаються в orders.json (рис. 3.43).

Рисунок 3.43 – Зберігання даних про замовлення

3.6. Висновок до третього розділу

Отже, результатом розробки є успішна реалізація додатку, що побудована на архітектурі VIPER. Дана архітектура дозволяє розробникам чітко визначити межі відповідальності кожного компонента, що сприяє більшій структурованості коду і полегшує процеси налагодження та розширення функціональності.

У третьому розділі було детально розглянуто компоненти VIPER такі як View, Interactor, Presenter, Entity та Router. Кожен з цих компонентів має власні функції які повинен виконувати. середовище розробки для веб додатку. У якості середовища розробки було обрано Microsoft Visual Studio. Було надано структуру проекту, діаграму класів, та наведено приклади програмної реалізації класів та функцій. Робота розробленого програмного забезпечення та його основні функції проілюстровані екранними формами.

В результаті виконання мети кваліфікаційної роботи у третьому розділі було проілюстровано розроблений додаток, що може безпечно обробляти перевіряти введені дані користувачем, за рахунок розподіленого функціоналу між компонентами додатку.

ВИСНОВОК

Кваліфікаційна робота присвячена розробці веб-застосунку для закладів громадського харчування, що є актуальним завданням у контексті сучасної цифрової трансформації бізнесу. Метою роботи було створення ефективного інструменту для автоматизації процесів замовлення, оплати та доставки, що сприятиме підвищенню ефективності роботи закладів та покращенню якості обслуговування клієнтів.

У першому розділі роботи було проведено детальний аналіз існуючих рішень та підходів до розробки веб-застосунків. Особливу увагу було приділено сучасним архітектурним патернам, таким як MVC, MVVM та VIPER. Встановлено, що вибір правильного патерну та технологічного стека є критичним для успішної реалізації проекту, оскільки це забезпечує структурування, модульність та гнучкість коду.

У другому розділі роботи було проаналізовано та обрано оптимальні технології для реалізації проекту. Для розробки серверної частини було обрано мову програмування C# та платформу .NET, що забезпечили потужні можливості для створення надійного та масштабованого додатку. Використання середовища розробки Visual Studio Community 2022 значно спростило процес написання, налагодження та тестування коду. Для забезпечення високої модульності, тестованості та масштабованості додатку було обрано архітектурний патерн VIPER.

У третьому розділі було детально описано процес розробки веб-застосунку. Реалізовано ключові компоненти VIPER: View, Interactor, Presenter, Entity та Router, що дозволило забезпечити чітке розділення обов'язків між компонентами. Було створено три основні представлення для користувачів: головна сторінка, сторінка авторизації та сторінка замовлень, що дозволяє користувачам зручно здійснювати замовлення онлайн.

Проведене тестування веб-застосунку підтвердило його функціональність та відповідність вимогам. Результати тестування показали,

що застосунок ефективно обробляє замовлення, забезпечує надійне зберігання даних та пропонує користувачам зручний інтерфейс для взаємодії.

Розроблений веб-застосунок демонструє високу ефективність та відповідає сучасним вимогам бізнесу в сфері громадського харчування. Використання архітектурного патерну VIPER забезпечило високу модульність та гнучкість коду, що полегшує його подальшу підтримку та розвиток. Результати роботи можуть бути корисними для інших розробників та для закладів громадського харчування, які прагнуть підвищити свою конкурентоспроможність через автоматизацію процесів.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Андерсон М. Squarespace. Website planet. – [Електронний ресурс] – Режим доступу: https://www.websiteplanet.com/uk/website-builders/squarespace/.

2. NAKANO, Yuusuke, et al. Web performance acceleration by caching rendering results. In: 2015 17th Asia-Pacific Network Operations and Management Symposium (APNOMS). IEEE, 2015. p. 244-249

3. Мадрі Д. Weebly. Website planet. – [Електронний ресурс] – Режим доступу: https://www.websiteplanet.com/uk/website-builders/weebly/

4. Односторінкові сайти: види, застосування, приклади.– [Електронний ресурс] – Режим доступу: https://vmb.net.ua/552-odnostorinkovi-sajty-vydy-zastosuvannya-pryklady/.

5. Склоттмен А. GoDaddy Website Builder. Website planet. – [Електронний ресурс] – Режим доступу: https://www.websiteplanet.com/uk/website-builders/godaddy-website-builder/.

6. JavaScript JSON. W3Schools. URL: https://www.w3schools.com/js/js_json_intro.asp. (Дата звернення: 18.05.23)

7. C# і .NET | Властивості. METANIT.COM - Сайт про програмування. URL: https://metanit.com/sharp/tutorial/3.4.php (дата звернення: 14.02.2023).

8. Morales-Vargas, A., Pedraza-Jiménez, R., & Codina, L. (2020). Website quality: An analysis of scientific production. Profesional De La Información.

9. ASP.NET documentation. Microsoft. – [Електронний ресурс] – Режим доступу: https://docs.microsoft.com/en-us/aspnet/overview

10. Client-Server Architecture. Sciencedirect. – [Електронний ресурс] – Режим доступу: https://www.sciencedirect.com/topics/computer-science/client-server-architecture

11. C# Programming Language. Geekforgeeks. – [Електронний ресурс] – Режим доступу: https://www.geeksforgeeks.org/csharp-programming-language/

12. MVC Framework Introduction – [Електронний ресурс] – Режим доступу: https://www.geeksforgeeks.org/mvc-framework-introduction/

13. Introduction to MVVM Architecture – [Електронний ресурс] – Режим доступу: https://medium.com/@onurcem.isik/introduction-to-mvvm-architecture-5c5558c3679

14. Understanding VIPER Architecture – [Електронний ресурс] – Режим доступу: https://medium.com/@pinarkocak/understanding-viper-pattern-619fa9a0b1f1

15. Що таке UML-діаграми? – [Електронний ресурс] – Режим доступу: https://evergreens.com.ua/ua/articles/uml-diagrams.html