**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**
**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»**

**Інститут прикладного системного аналізу**
(повна назва інституту/факультету)

**Кафедра системного проектування**
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри
_____ А.І. Петренко
(підпис) (ініціали, прізвище)
“___”_____2020 р.

# Дипломна робота

**на здобуття ступеня бакалавра**
**за освітньо-професійною програмою**
**«Інтелектуальні сервіс-орієнтовані розподілені обчислювання»**
**зі спеціальності 122 «Комп’ютерні науки»**

**на тему: «Дослідження технології Blazor для розробки інтерактивних веб-додатків»**

Виконав:
студент 4 курсу, групи ІТ-ЗП81
Бобирь Віктор Леонідович _____
(підпис)

Керівник:
доцент, кандидат технічних наук,
Булах Богдан Вікторович _____
(підпис)

Консультант економічний:
доцент, к.е.н.,
Рощина Надія Василівна _____
(підпис)

Рецензент:
доцент, к.т.н.,
Тимощук Оксана Леонідівна _____
(підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2020

**Національний технічний університет України**
**«Київський політехнічний інститут імені Ігоря Сікорського»**
**Інститут прикладного системного аналізу**
**Кафедра системного проектування**

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 122 «Комп'ютерні науки`»

Освітньо-професійна програма «Інтелектуальні сервіс-орієнтовані розподілені обчислювання»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ А.І. Петренко
«___»_____2020 р.

**ЗАВДАННЯ**
**на дипломну роботу студенту**
**Бобирю Віктору Леонідовичу**

1. Тема роботи «Дослідження технології Blazor для розробки інтерактивних веб-додатків», керівник роботи Булах Богдан Вікторович, кандидат технічних наук, доцент, затверджені наказом по університету від «01» липня 2020 р. № 1623-ф.2.

2. Термін подання студентом роботи «15» грудня 2020 р.

3. Вихідні дані до роботи

Дослідження та використання фреймворку Blazor платформи .NET.

4. Зміст роботи

Дослідження технології Blazor на прикладі проектування інтерактивного веб-додатка для продажу продуктів харчування.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо)

Рисунки, додаток з лістингом програми, презентація до захисту роботи.

6. Консультанти розділів роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| Економічний | Рощина Н.В., к.е.н., доцент | 01.09.20 | |

7. Дата видачі завдання 01.08.2020 р.

Календарний план

| № з/п | Назва етапів виконання дипломної роботи | Термін виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Отримання завдання | 01.08.2020 | |
| 2 | Аналіз предметної області | 01.09.2020 | |
| 3 | Збір інформації | 01.10.2020 | |
| 4 | Аналіз існуючих методів | 20.10.2020 | |
| 5 | Власне дослідження засобів | 20.11.2020 | |
| 6 | Функціонально-вартісний аналіз | 05.12.2020 | |
| 7 | Оформлення дипломної роботи | 12.12.2020 | |
| 8 | Отримання допуску до захисту та подача роботи в ДЕК | 15.12.2020 | |

Студент _____ Бобирь В.Л.
(підпис) (ініціали, прізвище)

Керівник роботи _____ Булах Б.В.
(підпис) (ініціали, прізвище)

# АНОТАЦІЯ

До бакалаврської дипломної роботи Бобиря Віктора Леонідовича

на тему:

«Дослідження технології Blazor для розробки інтерактивних веб-додатків»

Метою даної дипломної роботи є дослідження нової технології Blazor від компанії Microsoft, створення за її допомоги на мові програмування C# інтерактивного веб-додатка для продажу продуктів харчування, використовуючи можливості платформи .NET та фреймворку ASP.NET Core.

В роботі проведено аналіз розвитку технологій ASP.NET Core та Blazor, наведено особливості та переваги технологій Blazor Server та Blazor WebAssembly. Досліджено компоненти Blazor, їх будова, розміщення, способи виклику та механізми перенесення даних між ними. Підкреслено важливість обробки подій компонентів Blazor та зв’язування даних для створення інтерактивних веб-додатків.

Розроблено інтерактивний веб-додаток за допомогою технології Blazor Server, описано клієнтський інтерфейс та переваги веб-додатку.

Загальний об’єм роботи 73 с., 1 д., 28 рис., 8 табл., 17 джерел.

Ключові слова: Blazor, платформа .NET, фреймворк ASP.NET Core, веб-додаток.

# ANNOTATION

On Bobyr Viktor Leonidovych bachelor's degree thesis:

"Exploration Blazor technology for the development of interactive web applications"

The purpose of this thesis is to explore the new Blazor technology from Microsoft, and creation with its help and C # programming language an interactive web application for food sales, using the capabilities of the .NET platform and the ASP.NET Core framework.

The analysis of development of ASP.NET Core and Blazor technologies is carried out in the work, features and advantages of Blazor Server and Blazor WebAssembly technologies are given. Blazor components, their structure, location, methods of calling and mechanisms of data transfer between them are studied. The importance of handling Blazor component events and linking data to create interactive web applications is emphasized.

Developed an interactive web application using Blazor Server technology, describes the client interface and benefits of the web application.

The total volume of work is 73 pages, 1 appendix, 28 figures, 8 tables, 17 sources.

Keywords: Blazor, .NET platform, ASP.NET Core framework, web application.

**ЗМІСТ**

# ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

.NET – програмна технологія, запропонована компанією Microsoft як платформа для створення як звичайних програм, так і веб-застосунків.

ASP.NET Core – технологія створення веб-застосунків і веб-сервісів від компанії Microsoft

API – Application Programming Interface.

CIL – Common Intermediate Language

DOM – Document Object Model

SPA – Single Page Application

WASP – WebAssembly

# ВСТУП

В сучасному світі все поступово йде до глобальної автоматизації та комп’ютеризації. Сьогодні дуже важко знайти компанію чи приватну особу, яка не користується IT технологіями для розвитку власного бізнесу. Оскільки найбільше для цих цілей використовується Інтернет, то зовсім не дивно, що саме веб-додатки здобувають все більшу популярність.

В даній роботі розглянуто нову технологію Blazor від компанії Microsoft, яка використовується для створення інтерактивних додатків, що можуть працювати як на стороні сервера, так і на стороні клієнта, на платформі .NET.

В своєму розвитку Blazor зазнав значного впливу сучасних фреймворків для створення клієнтських додатків – Angular, React, VueJS.

Для детального ознайомлення з механізмами функціонування технології Blazor було досліджено її основні складові – компоненти, їх будову, розміщення та способи використання. Значну увагу було приділено використанню параметрів та обробці подій в компонентах. На основі аналізу сильних сторін технології Blazor було розроблено структуру інтерактивного веб-додатка, створено відповідну базу даних та розроблено клієнтський інтерфейс веб-додатка.

У даній роботі для реалізації інтерактивного веб-додатка була застосована платформа .NET, фреймворк ASP.NET Core, технологія Blazor та мова програмування C#.

# 1 ПОСТАНОВКА ЗАДАЧІ

Метою даної дипломної роботи є дослідження побудови та механізмів функціонування технології Blazor платформи .NET та аналіз її переваг та недоліків при розробці інтерактивних веб-додатків, на прикладі додатку з продажу продуктів харчування.

Веб-додаток повинен відповідати наступним критеріям:

а) використання лише мови програмування C#, HTML та CSS;

б) Blazor Server – технологія для розробки інтерактивного веб-додатка;

в) збереження даних веб-додатка в реляційній базі даних.

При розробці відповідного програмного забезпечення потрібно розв’язати наступні завдання:

а) дослідження основних механізмів технології Blazor;

б) застосування обробки подій компонентів Blazor для створення інтерактивних елементів;

в) зв’язування даних компонентів при створенні веб-додатку.

Реалізований веб-додаток має задовольняти такі вимоги:

а) мати високі показники продуктивності;

б) мати простий, але водночас зручний інтерфейс користувача;

в) користувачі повинні легко знаходити потрібний товар шляхом текстового пошуку, додавати його до кошика, змінювати вміст кошика, оформлювати замовлення;

г) адміністративним користувачі після авторизації повинні мати можливість опрацьовувати замовлення покупців, переглядати історію замовлень.

# 2 ЕВОЛЮЦІЯ СЕРЕДОВИЩА ПЛАТФОРМИ .NET І РОЗВИТОК ТЕХНОЛОГІЙ ASP.NET CORE ТА BLAZOR

## 2.1 Особливості мови програмування C# платформи .NET

Мова програмування C# (вимовляється Сі-шарп) – це мова програмування від компанії Microsoft, яка разом з Visual Basic.NET, Visual C++ та F# входить в середовище розробки Visual Studio. Одна з причин створення мови C# компанією Microsoft – це реалізація об’єктно-орієнтованої мови для платформи .NET. Багато інших мов було створено до появи платформи .NET, мова ж C# створювалася спеціально під цю платформу. Проте, звісно, для платформи .NET мова програмування C# не є єдиною.

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 що від своїх попередників – мов C++, Delphi, Smalltalk – C#, спираючись на практику їхнього використання, виключає деякі моделі, що зарекомендували себе як проблематичні при розробці програмних систем, наприклад множинне спадкування класів (на відміну від C++). [1]

Згідно стандарту ECMA-334, цілі, поставлені при розробці мови C#, були наступними:

- C# має бути простою, сучасною, об'єктно-орієнтованою мовою програмування;
- мова має підтримувати безпечні принципи програмування, такі як строга перевірка типів, перевірка меж масиву, виявлення спроб використання неініціалізованих змінних, і автоматичне прибирання сміття;
- можливість розробки програмних компонентів для розподілених систем;
- мова має підтримувати переносимість коду (портативність);

- підтримка національних мовних та інших особливостей має бути простою. [13]

Базовий синтаксис C# схожий до інших C-подібних мов, таких як C, C++ та Java.

Існує кілька середовищ програмування, які використовують мову C#, зокрема: Microsoft Visual Studio, MonoDevelop, ReSharper тощо. Найуживанішим з них є інтегроване середовище розробки Microsoft Visual Studio, яке, крім C#, підтримує і ряд інших мов програмування, а також забезпечує великий набір інструментів для розв'язування різних завдань.

## 2.2 Розвиток програмного забезпечення ASP.NET Core в екосистемі платформи .NET

Платформа ASP.NET Core – технологія від компанії Microsoft, призначена для створення різного роду веб-додатків: від невеликих веб-сайтів до великих веб-порталів і веб-сервісів.

З одного боку, ASP.NET Core є продовженням розвитку платформи ASP.NET. Але з іншого боку, це не просто черговий реліз. Вихід ASP.NET Core фактично означав революцію всієї платформи, її якісна зміна (рис. 2.1) [12].

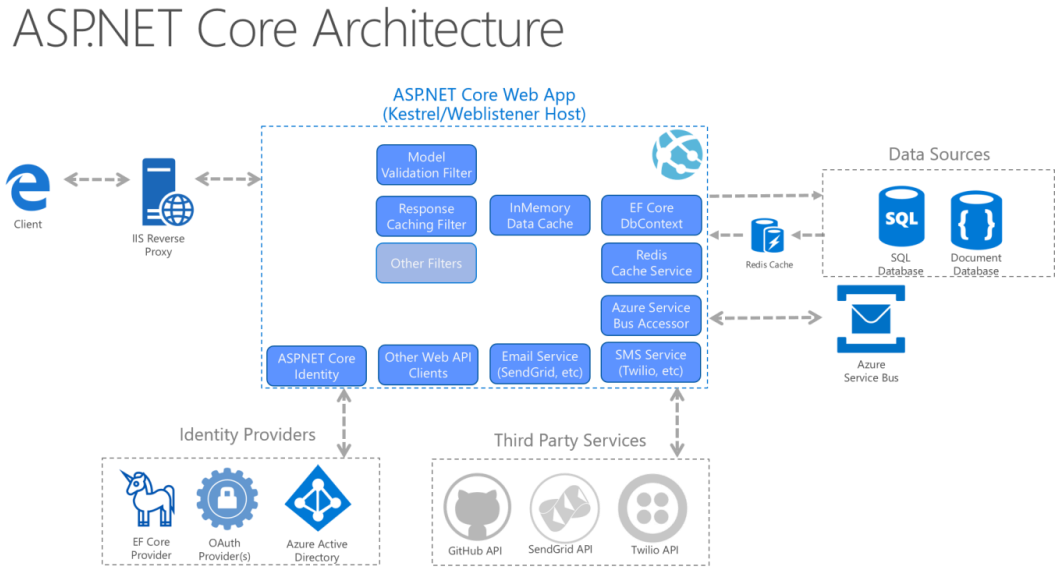


Рисунок 2.1 – Архітектура технології ASP.NET Core

Розробка над платформою почалася ще в 2014 році. Тоді платформа умовно називалася ASP.NET vNext. У червні 2016 року вийшов перший реліз платформи. А в грудні 2019 року світ побачила версія ASP.NET Core 3.1, яка власне і буде нами використовуватися в дипломній роботі.

ASP.NET Core – фреймворк з повністю з відкритим вихідним кодом. Всі вихідні файли фреймворку доступні на GitHub.

ASP.NET Core може працювати на базі крос-платформного середовища .NET Core, яке може бути розгорнуте на основних популярних операційних системах: Windows, Mac OS, Linux. Тобто, за допомогою ASP.NET Core ми можемо створювати крос-платформні додатки. Для розгортання веб-додатка можна використовувати традиційний IIS, або крос-платформний веб-сервер Kestrel.

Завдяки модульності фреймворка всі необхідні компоненти веб-додатка можуть завантажуватися як окремі модулі через пакетний менеджер Nuget. Крім того, на відміну від попередніх версій платформи немає необхідності використовувати бібліотеку System.Web.dll.

ASP.NET Core включає в себе фреймворк MVC, який об'єднує функціональність MVC, Web API і Web Pages. У попередніх версії платформи дані технології реалізувалися окремо і тому містили багато дублюючої функціональності. Зараз же вони об'єднані в одну програмну модель ASP.NET Core MVC. А Web Forms повністю пішли в минуле. [12]

Крім об'єднання вищезазначених технологій в одну модель в MVC був доданий ряд додаткових функцій. Однією з таких функцій є тег-хелпери (tag helper), які дозволяють більш органічно поєднувати синтаксис html з кодом C#.

ASP.NET Core характеризується розширюваністю. Фреймворк побудований з набору щодо незалежних компонентів. І ми можемо або використовувати вбудовану реалізацію цих компонентів, або розширити їх за допомогою механізму спадкування, або зовсім створити і застосовувати свої компоненти зі своїм функціоналом.

Підсумовуючи, можна виділити наступні ключові відмінності ASP.NET Core від попередніх версій ASP.NET:

- новий легкий і модульний конвеєр HTTP-запитів;
- можливість розгортати додаток як на IIS, так і в рамках свого власного процесу;
- використання платформи .NET Core і її функціональності;
- поширення пакетів платформи через NuGet;
- інтегрована підтримка для створення та використання пакетів NuGet;
- єдиний стек веб-розробки, що поєднує Web UI і Web API;
- конфігурація для спрощеного використання в хмарі;
- вбудована підтримка для впровадження залежностей;
- можливість розширення;
- кросплатформеність: можливість розробки і розгортання додатків ASP.NET на Windows, Mac і Linux;
- розвиток як open source, відкритість до змін.

Ці та інші особливості і можливості стали основою для нової моделі програмування.

## 2.3 Дослідження технології Blazor та її переваг

Технологія Blazor – фреймворк інтерфейсу користувача (UI-фреймворк), який використовується для створення інтерактивних додатків, що можуть працювати як на стороні сервера, так і на стороні клієнта, на платформі .NET. В своєму розвитку фреймворк Blazor зазнав значного впливу сучасних фреймворків для створення клієнтських додатків – Angular, React, VueJS. Зокрема, це проявляється в ролі компонентів при побудові інтерфейсу користувача.

В той же час і на стороні клієнта, і на стороні сервера при визначенні коду в якості мови програмування застосовується C#, замість JavaScript. А для опису візуального інтерфейсу використовуються стандартні HTML і CSS. [10]

Фреймворк Blazor розвивається як проект з відкритим вихідним кодом, розташованим в репозиторії ASP.NET Core на онлайн ресурсі github.com: https://github.com/dotnet/aspnetcore/.

Blazor надає розробникам наступні переваги:

- написання коду веб-додатків за допомогою C# замість JavaScript;
- використання можливостей екосистеми .NET, зокрема, бібліотек .NET при створенні додатків, безпеки і продуктивності платформи .NET;
- клієнтська і серверна частини програми можуть використовувати загальну логіку;
- використання Visual Studio в якості інструменту для розробки, який має вбудовані шаблони для спрощення створення додатка. [15]

Функціонально на поточний момент Blazor ділиться на дві підсистеми:

- Blazor Server: дозволяє створювати серверні додатки і підтримується ASP.NET Core;
- Blazor WebAssembly: дозволяє створювати односторінкові інтерактивні додатки клієнтської сторони, які запускаються в браузері користувача і працюють за допомогою технології WebAssembly (Wasm).

Blazor Server вийшов в реліз у вересні 2019 року, а Blazor WebAssembly – в травні 2020 року, і обидві ці платформи включені в .NET і повноцінно можуть використовуватися для створення серверних додатків і клієнтських додатків. Тобто фактично Blazor покриває потреби в веб-додатках як на стороні сервера, так і на стороні клієнта. Проте у компанії Microsoft є далекосяжні плани з розвитку Blazor. В майбутнього передбачається, що на Blazor можна буде створювати також нативні мобільні і десктопні додатки для різних платформ (рис. 2.2) . [14]

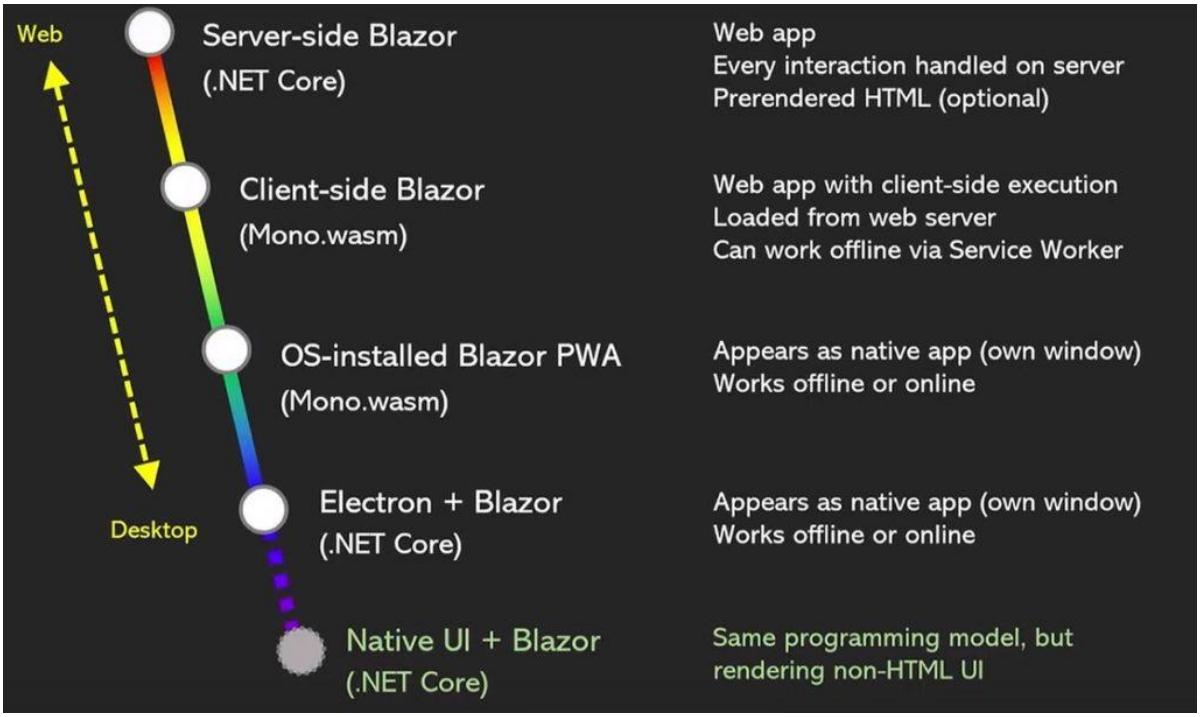


Рисунок 2.2 – Перспективи розвитку технології Blazor

Blazor WebAssembly дозволяє створювати інтерактивні односторінкові додатки (SPA), які запускаються в браузері користувача за допомогою технології WebAssembly. При побудові і запуску програми Blazor WebAssembly файли з кодом C# і Razor компілюються в збірки .NET. Потім Blazor WebAssembly завантажує середовище виконання .NET, збірки і їх залежності і налаштовує середовище виконання .NET для виконання збірок.

За допомогою взаємодії з JavaScript фреймворк Blazor WebAssembly може звертатися до DOM і API браузера (рис. 2.3) . [4]

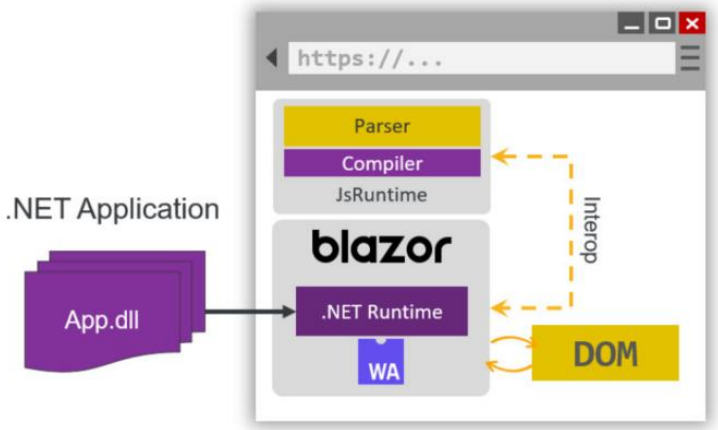


Рисунок 2.3 – Взаємодія Blazor з DOM браузера

Одним з переваг Blazor WebAssembly є те, що він може оптимізувати завантаження збірки. Зокрема, при публікації додатка невикористаний код забирається лінкером (компоновщиком) IL. Крім того, всі необхідні файли середовища виконання .NET і збірок, що завантажуються, кешуються в браузері.

При цьому Blazor WebAssembly не залежить від сервера. За великим рахунком нам може бути достатньо статичного сервера, на якому розміщені всі файли програми. Всі необхідні файли завантажуються браузером, і після завантаження файлів додаток працює повністю на стороні браузера і абсолютно не залежить від сервера.

У той же час Blazor WebAssembly має ряд обмежень. Наприклад, браузер повинен підтримувати технологію WebAssembly – на даний момент останні версії поширених браузерів (Google Chrome, Mozilla Firefox, Opera, Microsoft Edge) підтримують цю технологію. Однак більш старі версії, або Internet Explorer не мають подібної підтримки. Також браузеру необхідно завантажити файли великого розміру, так як додаток повністю відпрацьовує на стороні клієнта, що збільшує навантаження на мережу і час завантаження. [15]

При використанні технології Blazor Server додаток відпрацьовує на стороні сервера. Оновлення елементів призначеного для користувача інтерфейсу, обробка подій, виклики JavaScript на стороні клієнта здійснюються за допомогою взаємодії сервера і клієнта через SignalR.

Тобто коли користувач взаємодіє з додатком в браузері, викликає події призначеного для користувача інтерфейсу (наприклад, натискає на кнопку), то клієнтська сторона посилає на сервер інформацію про подію, сервер обробляє отриману інформацію і посилає клієнтові у відповідь інструкції, як необхідно оновити елементи інтерфейсу. В якійсь мірі це схоже на підхід, що застосовувався в ASP.NET WebForms.

Оскільки велика частина логіки додатка зосереджена на стороні сервера, то всі файли, що завантажуються клієнтом, набагато менший розмір порівняно з Blazor WebAssembly. Веб-додаток не обмежений браузером і може скористатися

можливостями серверної обробки. Крім того, додаток може працювати з застарілими браузерами, які не підтримують WebAssembly. У той же час для роботи програми необхідна постійна підтримка мережевого підключення .

## 2.4 Огляд технології Blazor Server

При використанні технології Blazor Server основна логіка додатка розташовується на стороні сервера. Якщо на боці клієнта відбуваються якісь події, то за допомогою SignalR клієнт посилає серверу інформацію про проведені дії. Сервер отримує цю інформацію, обробляє її і посилає клієнтові відповідь. Оновлення елементів інтерфейсу користувача, обробка подій, виклики JavaScript на стороні клієнта здійснюються за допомогою взаємодії сервера і клієнта через SignalR.

Середовище розробки програмного забезпечення Visual Studio 2019 дозволяє доволі просто створити веб-додаток з використанням технології Blazor Server. Дане середовище розробки має чудово підтримку технології Blazor, зокрема, вже за замовчуванням існує шаблон для створення відповідної програми (рис. 2.4) . [6]

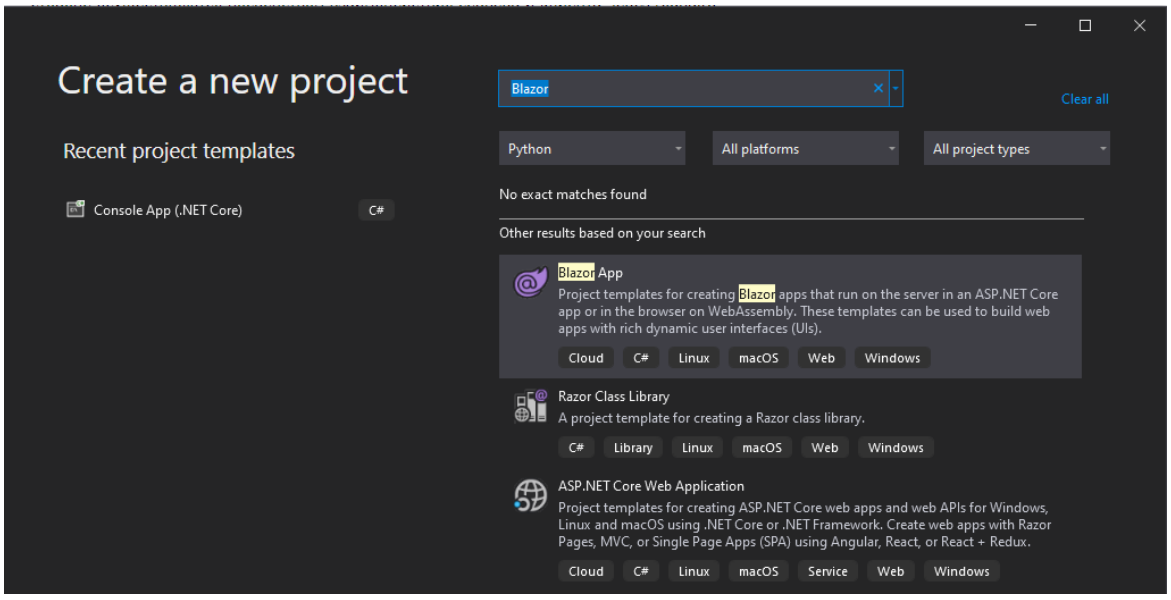


Рисунок 2.4 – Створення нового проекту Blazor App в середовиці розробки програмного забезпечення Visual Studio 2019

Варто відзначити, що структура проекту Blazor схожа на проекти ASP.NET Core, зокрема, проект для Razor Page.

Основні елементи проекту (рис. 2.5):

- Тека wwwroot для зберігання статичних файлів, за замовчуванням зберігає файли css, що використовуються, зокрема, файли фреймворка bootstrap.
- Тека Data зберігає класи C#, які описують дані, що використовуються (клас WeatherForecast), і сервіси (клас WeatherForecastService).
- Тека Pages містить сторінки Razor Pages, що визначають візуальну частину програми та його логіку, а також компоненти Razor (розташовуються в файлах з розширенням * .razor ), які слугують для представлення основного змісту сторінки.

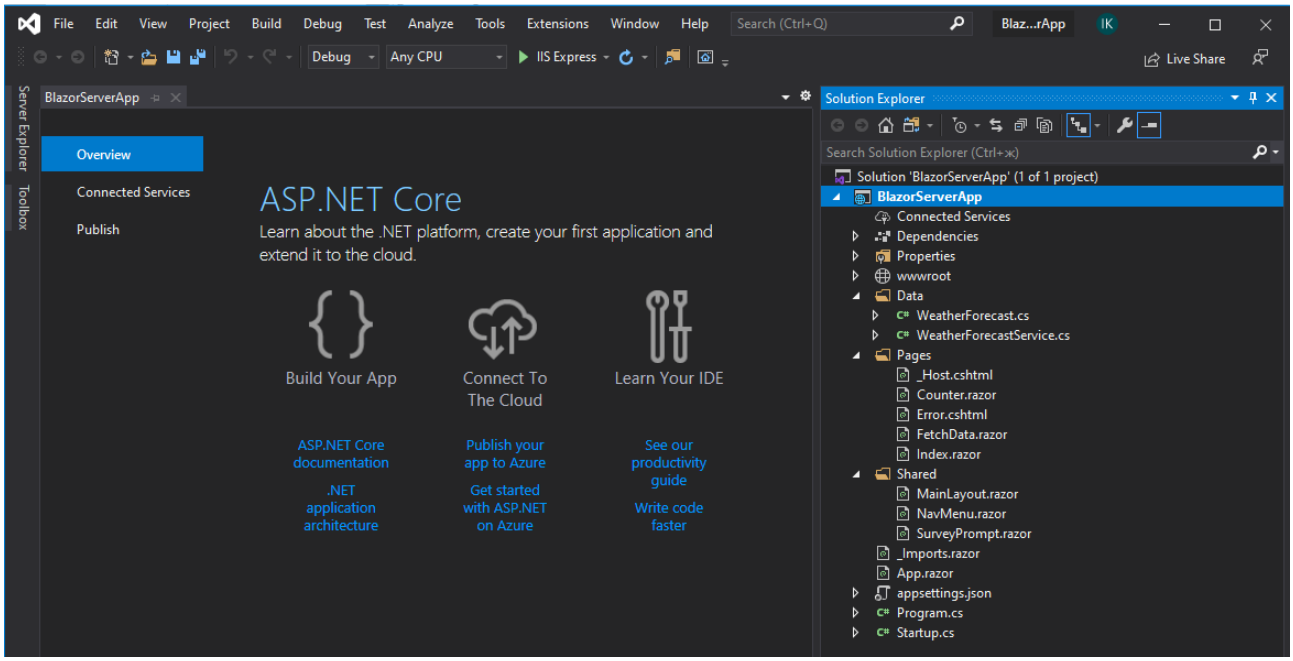


Рисунок 2.5 – Структура проекту Blazor Server

Проект Blazor Server вже містить деяку базову типову функціональність, що дозволяє нам запустити проект і оцінити роботу даного фреймворка.

Наприклад, натиснення на кнопку «Fetch data» (ліворуч) запускає компонент FetchData, який за допомогою сервісу WeatherForecastService отримує деякі дані та виводить їх на сторінку (рис. 2.6).

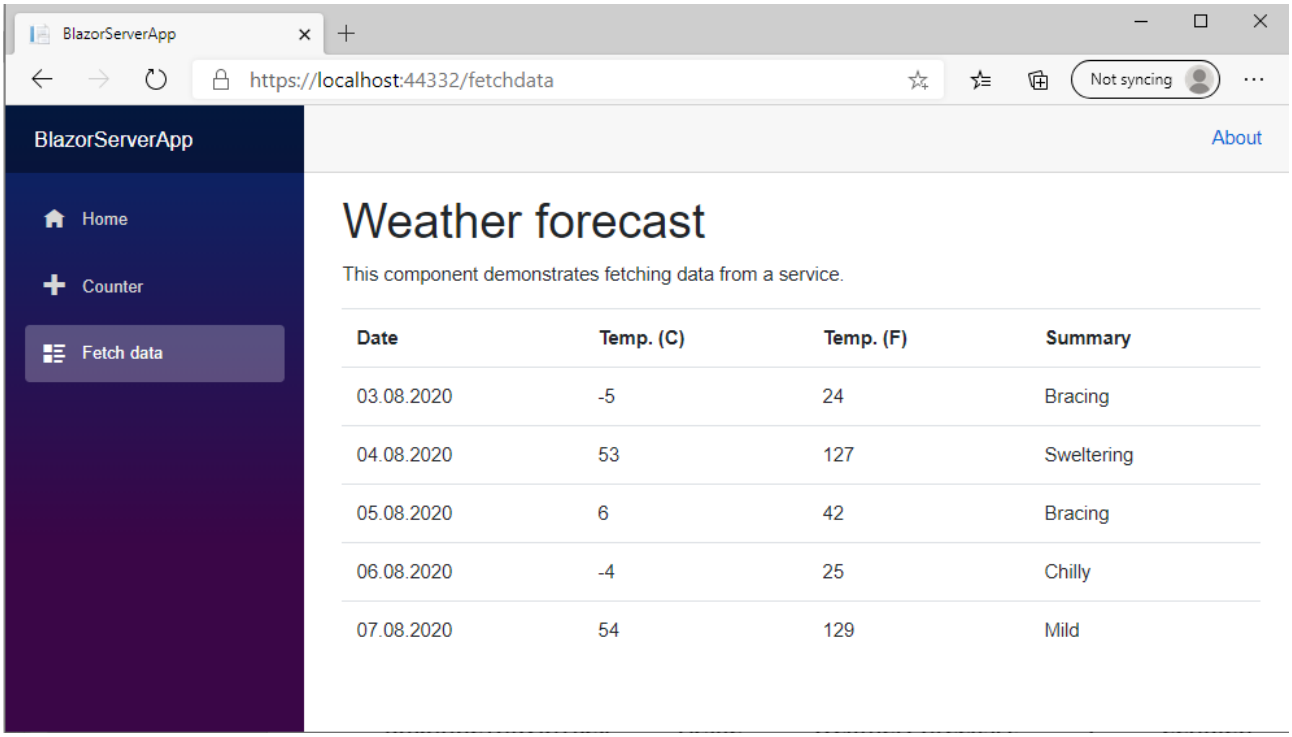


Рисунок 2.6 – Сторінка «Weather forecast» шаблонного проекту веб-додатку Blazor Server

Приклад коду компоненти «FetchData», що отримує за допомогою впровадженого сервісу WeatherForecastService дані і виводить їх в код html, наведено в Додатку А.

Отже, так виглядає веб-додаток, побудований на технології Blazor Server, використовуючи середовище розробки програмного забезпечення Visual Studio 2019. Наступним розглянемо побудову веб-додатку на технології Blazor WebAssembly.

## 2.5 Огляд технології Blazor WebAssembly

Blazor WebAssembly дозволяє створювати додатки клієнтської сторони, які повністю працюють на стороні клієнта за допомогою технології WebAssembly. Під час запуску програми файли програми, всі її залежності та середовище

виконання .NET завантажуються браузером. Додаток виконується повністю на стороні клієнта в браузері.

Для створення проекту Blazor WebAssembly в середовищі розробки програмного забезпечення Visual Studio 2019 також існує відповідний пункт меню.

Структура проекту Blazor WebAssembly частково схожа на структуру проекту Blazor Server, однак в деяких деталях все-таки відрізняється:

- Тека wwwroot містить статичні файли програми.
- Тека Pages містить компоненти Razor.
- Тека Shared зберігає додаткові компоненти Razor.
- _Imports.razor містить підключення просторів імен за допомогою директиви using, які будуть підключатися в компоненти Razor.
- App.razor містить визначення кореневої компоненти додатка, яка дозволяє встановити маршрутизацію між вкладеними компонентами за допомогою іншої вбудованої компоненти Router.
- Файл Program.cs містить клас Program, який представляє точку входу в додаток. В даному випадку це стандартний для додатка ASP.NET Core клас Program, який запускає і конфігурує хост, в рамках якого розгортається веб-додаток Blazor.

Проект Blazor WebAssembly також містить базову типову функціональність, що дозволяє нам запустити проект і оцінити його (рис. 2.7).

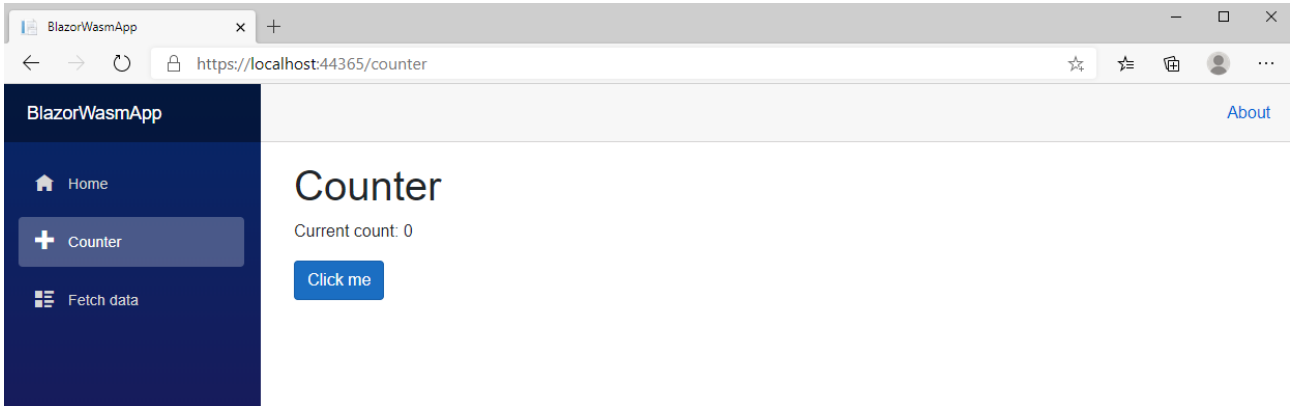


Рисунок 2.7 – Сторінка «Counter» веб-додатка Blazor WebAssembly

Розглянемо базові принципи роботи програми Blazor WebAssembly.

Робота програми починається з класу Program:

```
public class Program
{
    public static async Task Main(string[] args)
    {
        var builder = WebAssemblyHostBuilder.CreateDefault(args);
        builder.RootComponents.Add<App>("app");

        builder.Services.AddTransient(sp => new HttpClient {
            BaseAddress = new Uri(builder.HostEnvironment.BaseAddress)
            });
        await builder.Build().RunAsync();
    }
}
```

Основне завдання класу Program – налаштувати і запустити хост, який представлений класом WebAssemblyHost . Для цього в класу створювача хоста – WebAssemblyHostBuilder викликається метод Build(). А для запуску хоста викликається метод RunAsync(). Крім цього, за допомогою властивості RootComponents і її властивості Add() додається клас кореневої компоненти і її селектор ( builder.RootComponents.Add<App>("app"); ).

В даному випадку клас компоненти називається App, а для її рендерингу на веб-сторінці використовується елемент <app>.

Ну і крім того, тут в додаток впроваджується в якості сервісу HttpClient, який використовується в компонентах Blazor для відправки http-запитів:

```
builder.Services.AddTransient(sp => new HttpClient { BaseAddress = new Uri(builder.HostEnvironment.BaseAddress) });
```

При зверненні до програми браузер завантажить сторінку index.html з теки wwwroot:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width" />
    <title>HelloBlazorWasmApp</title>
    <base href="/" />
    <link href="css/bootstrap/bootstrap.min.css" rel="stylesheet" />
```

```
    <link href="css/site.css" rel="stylesheet" />
</head>

<body>
    <app>Loading...</app>
    <div id="blazor-error-ui">
        An unhandled error has occurred.
        <a href="" class="reload">Reload</a>
        <a class="dismiss">x</a>
    </div>
    <script src="_framework/blazor.webassembly.js"></script>
</body>
</html>
```

Це звичайна веб-сторінка, в коді якої можна виділити дві особливості. Перш за все це підключення скрипту blazor.webassembly.js . Цей скрипт завантажує середовище виконання .NET, додаток і його залежності та ініціалізує середовище виконання для завантаження програми. [7]

## 2.6 Висновки до розділу

В даному розділі було проаналізовано особливості платформи ASP.NET Core, яка стала революційною зміною платформи ASP.NET.

Також розглянуто та досліджено технологію Blazor – фреймворк інтерфейсу користувача (UI-фреймворк), який використовується для створення інтерактивних додатків, що можуть працювати як на стороні сервера, так і на стороні клієнта, на платформі .NET. Особливістю якого є можливість використання мови програмування C# як на стороні клієнта, так і на стороні сервера. При цьому для опису візуального інтерфейсу використовуються стандартні HTML і CSS.

# 3 ДОСЛІДЖЕННЯ КОМПОНЕНТІВ ЯК ОСНОВИ ТЕХНОЛОГІЇ BLAZOR

## 3.1 Побудова, розміщення та виклик компонентів фреймворку Blazor

Ключовим елементом технології Blazor є компоненти. Компоненти Blazor застосовується подібно до концепцій Angular, React, VueJS. Компонент – елемент інтерфейсу, наприклад, якийсь певний зміст, меню, діалогове вікно, форма введення даних. Компоненти визначають логіку рендерингу елементів інтерфейсу, а також логіку обробки дій користувача. Компоненти можуть бути вкладеними в інші компоненти. Їх можна повторно використовувати в інших проектах і переносити в вигляді бібліотеки класів Razor. Зазвичай клас компонента розташовується в файлі з розширенням .razor, а для їх визначення застосовується синтаксис Razor, який дозволяє об'єднати розмітку HTML з кодом на мові програмування C#. [3]

Razor – технологія, альтернативна системі Model-View-Controller, що дозволяє створювати сторінки з кодом Razor, характерною особливістю яких є здатність обробляти запити. В деякій мірі ця функціональність нагадує роботу веб-форм (сторінок з розширенням aspx), які містять файл логіки, написаний на C#, пов'язаний з даною сторінкою. Інакше кажучи, Razor представляє альтернативу стандартній моделі MVC для побудови програми.

Синтаксис Razor складається з розмітки Razor, C# та HTML. Файли, що містять Razor, зазвичай мають розширення .cshtml. Razor також міститься у файлах компонентів Razor (.razor). Рендеринг HTML із розмітки Razor не відрізняється від рендерингу HTML з файлу HTML.

Особливої уваги заслуговують такі елементи синтаксису Razor як:

- директиви – зарезервовані ключові слова з префіксом @, які зазвичай змінюють спосіб розбору розмітки компонентів;

- атрибути директив – зарезервовані ключові слова з префіксом @, які зазвичай змінюють спосіб розбору розмітки елементів компонентів.

Назва компонента має починатися з великої літери. Наприклад, MyBlazorComponent.razor є прийнятним для використання, проте myBlazorComponent.razor – ні.

Компонент, як і звичайні C# класи, може визначити змінні, які зберігають стан компонента, і методи, які визначають логіку компонента (наприклад, обробка подій візуального інтерфейсу). Всі методи, змінні та властивості компонента визначаються в межах блоку @code. [14]

Щоб звернутися до змінної компонента, використовується символ @:

```
<h1>Привіт, @name</h1>
<h2>@CurrentDate()</h2>
@code{
   string name = "друже";
   string CurrentDate()
   {
      return $"Сьогодні {DateTime.Now.ToString("dd.MM.yyyy")}";
   }}
```

Результат відпрацювання коду програми зображено на рисунку 3.1.

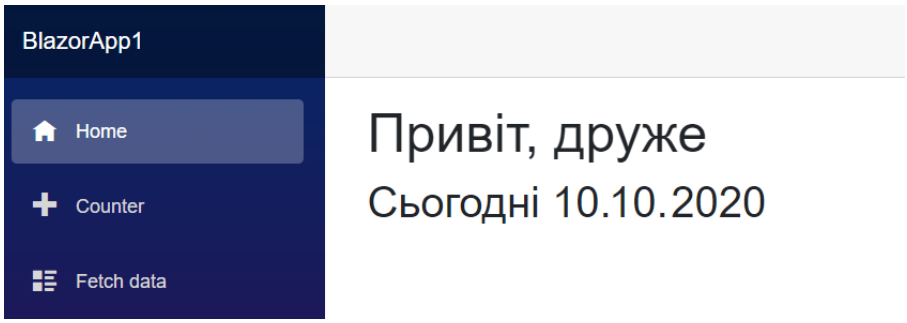


Рисунок 3.1 – Приклад відпрацювання коду компонента

Компонент Razor може використовувати стилі CSS, скрипти JavaScript, файли зображень та інший статичний вміст. У цьому випадку Blazor дотримується стандартних конвенцій, яких дотримуються в ASP.NET Core, а саме: статичний контекст розміщується в папці web root або кореневій папці для веб-контенту, яка за замовчуванням називається wwwroot. Щоб отримати доступ

до файлів з цієї теки, використовується відносний шлях, який починається із символу «/», що вказує на папку wwwroot.

Приклад розміщення файлів в папці wwwroot відображено на рисунку 3.2.

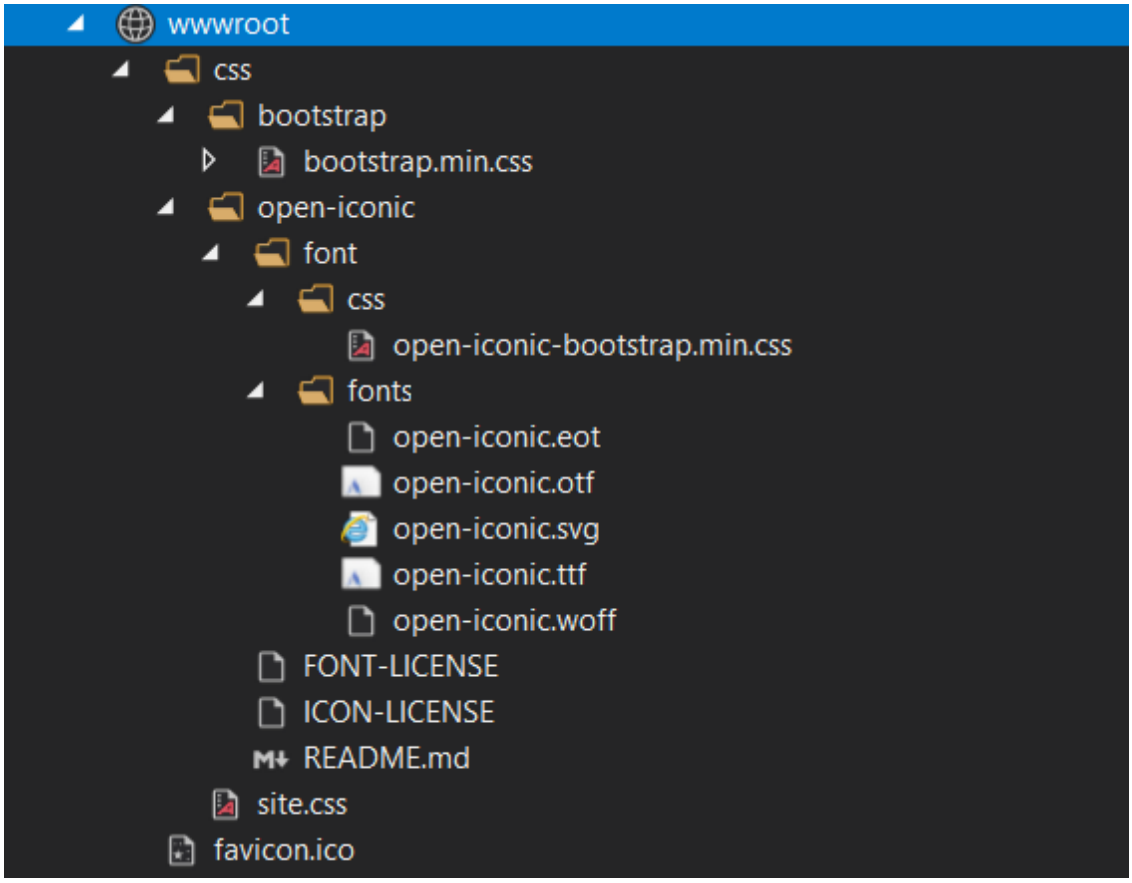


Рисунок 3.2 – Використання папки wwwroot для розміщення веб-контенту

Таким чином можна використовувати в проекті і інший контент, наприклад, скрипти JavaScript.

Компоненти можуть бути розташовані в будь-якому місці проекту. Єдине, що потрібно враховувати при виклику компонентів, це їх простір імен. Наприклад, якщо основний компонент Main.razor знаходиться в папці Components, то його простір імен – «[Project_name].Components». Для підключення простору імен в компоненті використовується директива @using:

```
@page  "/"
@namespace BlazorApp1.Pages
@*підключаємо простір імен компонента Main*@
@using BlazorApp1.Components
```

Власне застосування технології Blazor починається з виклику основного компонента веб-додатку, в межах якого розгортаються всі інші компоненти. Виклик головного компоненту Blazor Server відбувається або в коді сторінки Razor, або в поданні MVC. Для цього існує два способи – використання tag-helper <component> або HTML-helper Html.RenderComponentAsync().

## 3.2 Використання параметрів Blazor для перенесення даних між компонентами

Компоненти існують незалежно один від одного. Змінні та методи одного компонента застосовуються лише в межах окремого компонента. Наприклад, кожний компонент може містити власну перемінну заголовка тощо. Проте, часто виникає необхідність перенесення даних від головного компонента (батьківського) до підлеглого (дочірнього) чи навпаки. Така можливість забезпечується параметрами компонента – загальнодоступними властивостями компонента, до яких застосовується атрибут Parameter.

Розглянемо як працює використання параметрів на прикладі компонента MainPage.razor:

```
<h2>@header</h2>
@code{
string header = "Головна сторінка"; }
```

та модифікованого компонента Index.razor:

```
@page "/"
@using BlazorApp1.Components
<h1>@header</h1>
<MainPage></MainPage>
@code{
   string header = "веб-додаток Blazor";}
```

Відповідно до наведеного коду компонент головної сторінки MainPage.razor знаходиться в папці Components проекту BlazorApp1 (простір імен BlazorApp1.Components), тому для його використання в компоненті

Index.razor було використано директиву @using. Щоб контент компонента MainPage відображався в компоненті Index застосовано відповідну конструкцію тегів – <MainPage></MainPage>.

Результат відпрацювання коду програми зображено на рисунку 3.3:

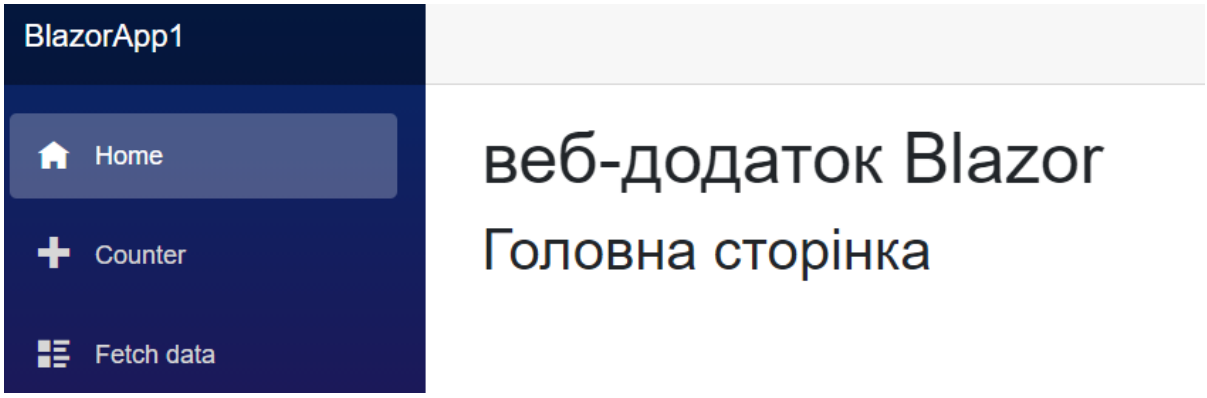


Рисунок 3.3 – Приклад відпрацювання коду компонента Index.razor

Так як компоненти існують незалежно один від одного, то змінні і методи одного компонента застосовуються тільки в рамках даного компонента. Проте існує можливість передачі даних з батьківського компонента до дочірнього. Цю можливість надають параметри компонента - публічні властивості компонента, до яких застосовується атрибут Parameter. Наприклад, змінюємо компонент MainPage.razor, додавши відповідні атрибути:

```
<h2>@Title</h2>
<h3>@Summary</h3>
@code{
   [Parameter]
   public string Title { get; set; }
   [Parameter]
   public string Summary { get; set; }}
```

Тепер компонент містить дві властивості, до яких застосовується атрибут [Parameter], тому ці властивості можна назвати параметрами компонента. Через них компонент зможе отримувати дані ззовні.

Змінимо виклик компонента MainPage в батьківському компоненті:

```
@page "/"
```

```
@using BlazorApp1.Components
<h1>@header</h1>
<MainPage Title="Головна сторінка"
Summary="@summary"></MainPage>
@code{
   string header = "веб-додаток Blazor";
   string summary = "тестування параметрів компонента";}
```

Через атрибути, назви яких збігаються з назвами параметрів компонента, можна передати цим параметрам необхідні дані. При цьому існує можливість передачі параметру компонента значення деякого виразу, наприклад, змінної (в нашому випадку – @summary) (рис. 3.4).

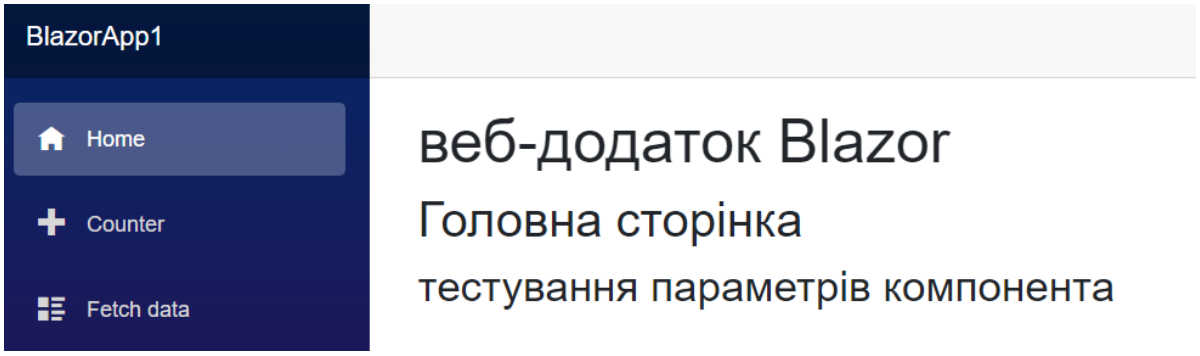


Рисунок 3.4 – Приклад відпрацювання коду модифікованого компонента Index.razor

Важливо зазначити, що є можливість передавати більш складний вміст, ніж звичайні текстові рядки. Наприклад, можна модифікувати компонент *MainPage*, щоб він приймав список:

```
<h2>Кількість студентів групи ІТ-зп81: @Count</h2>
<br />
<h3>Повний перелік:</h3>
<ul>
   @foreach (var student in Students)
   {  <li>@student</li> }
</ul>
@code{
   [Parameter]
   public List<string> Students { get; set; }
```

```
[Parameter]
public int Count { get; set; }}
```

Компонент тепер отримує список і відображає його в HTML-коді. Для відображення даного списку у батьківському компоненті модифікуємо відповідний код компонента Index.razor:

```
@page "/"
@using BlazorApp1.Components
<h1>@header</h1>
<MainPage Count="@students.Count" Summary="@users"></MainPage>
@code{
   string header = "веб-додаток Blazor";
   List<string> students = new List<string> { "Андрій", "Віктор", "Володимир", "Катаріна", "Кристина", "Юлія" };
}
```

Результат відпрацювання коду програми зображено на рисунку 3.5:

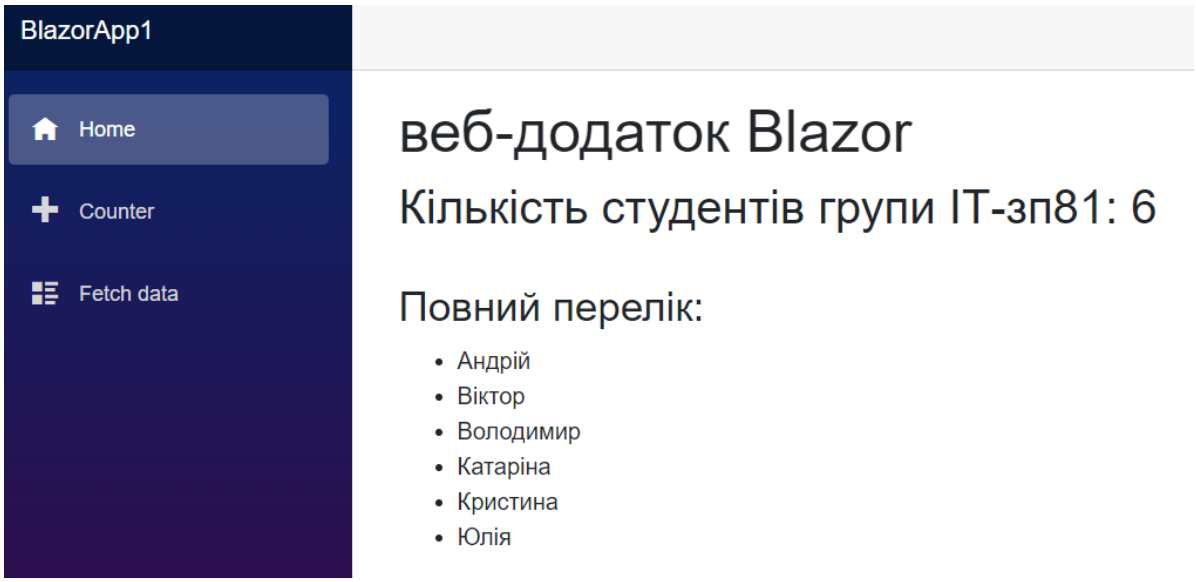


Рисунок 3.5 – Відображення результату перенесення списку між компонентами

## 3.3 Обробка подій компонентів Blazor для створення інтерактивних веб-додатків

Важливим елементом технології Blazor є підтримка компонентами обробки подій. Якщо елемент HTML має такі атрибути як on{ПОДІЯ}, які

дозволяють пов'язати подію з певною функцією JavaScript (наприклад, атрибут onclick), то технологія Blazor має свої аналоги - атрибути типу @on{ПОДІЯ}, які дозволяють прикріпити до події в якості обробника метод компонента. [5]

Прикладом використання атрибуту обробки подій може бути наступний компонент:

```
@using Microsoft.AspNetCore.Components.Web
<h1>Blazor Counter</h1>
<button @onclick="IncrementCount">+</button>
<span>@currentCount</span>
<button @onclick="DecrementCount">-</button>
@code {
  private int currentCount = 0;
   private void IncrementCount()
  {
     currentCount++;
  }
  private void DecrementCount()
  {
     currentCount--;
  }}
```

Перш за все, варто зазначити, що підтримка прив'язки подій зосереджена в просторі імен Microsoft.AspNetCore.Components.Web. Зокрема, в ньому визначено клас EventHandlers, який настроює порівняння між назвами подій і типами аргументів подій. Власне, у визначенні класу можна знайти і переглянути всі типи подій, які ним підтримуються.

HTML-елемент <button>, підтримує атрибут onclick, який дозволяє викликати функції JavaScript при натисканні кнопки. Blazor передбачає для цього атрибута свій аналог – @onclick. В якості значення цьому атрибуту можна передати один із методів компонента.

В наведеному прикладі визначено дві кнопки, які пов'язані з методами IncrementCount і DecrementCount. Метод IncrementCount призводить до

збільшення поточної змінної currentCount, а метод DecrementCount – до зменшення (рис. 3.6).


Рисунок 3.6 – Відображення результату використання атрибуту обробки подій

В таблиці 3.1 наведено повний перелік подій, що обробляються технологією Blazor. [12]

Таблиця 3.1 – Перелік подій (EventArgs), що підтримуються для обробки технологією Blazor

| **Подія** | **Клас** | **DOM події** |
|---|---|---|
| Clipboard | ClipboardEventArgs | oncut, oncopy, onpaste |
| Drag | DragEventArgs | ondrag, ondragstart, ondragenter, ondragleave, ondragover, ondrop, ondragend |
| Error | ErrorEventArgs | onerror |
| Event | EventArgs | *General*<br>onactivate, onbeforeactivate, onbeforedeactivate, ondeactivate, onfullscreenchange, onfullscreenerror, onloadeddata, onloadedmetadata, onpointerlockchange, onpointerlockerror, onreadystatechange, onscroll<br>*Clipboard*<br>onbeforecut, onbeforecopy, onbeforepaste<br>*Input*<br>oninvalid, onreset, onselect, onselectionchange, onselectstart, onsubmit<br>*Media*<br>oncanplay, oncanplaythrough, oncuechange, ondurationchange, onemptied, onended, onpause, onplay, onplaying, onratechange, onseeked, onseeking, onstalled, onstop, onsuspend, ontimeupdate, ontoggle, onvolumechange, onwaiting |
| Focus | FocusEventArgs | onfocus, onblur, onfocusin, onfocusout |
| Input | ChangeEventArgs | onchange, oninput |
| Keyboard | KeyboardEventArgs | onkeydown, onkeypress, onkeyup |

| Подія | Клас | DOM події |
|---|---|---|
| Mouse | MouseEventArgs | onclick, oncontextmenu, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout |
| Mouse pointer | PointerEventArgs | onpointerdown, onpointerup, onpointercancel, onpointermove, onpointerover, onpointerout, onpointerenter, onpointerleave, ongotpointercapture, onlostpointercapture |
| Mouse wheel | WheelEventArgs | onwheel, onmousewheel |
| Progress | ProgressEventArgs | onabort, onload, onloadend, onloadstart, onprogress, ontimeout |
| Touch | TouchEventArgs | ontouchstart, ontouchend, ontouchmove, ontouchenter, ontouchleave, ontouchcancel |

## 3.4 Зв’язування даних при створенні веб-додатків Blazor

Blazor дає можливість зв’язувати значення полів і властивостей компонентів з атрибутами елементів HTML. Для цього використовується атрибут @bind-. При цьому дані можна прив'язати до будь-якого атрибута елементу HTML. Наприклад, у елемента введення даних input є атрибут value, який представляє значення, введене в елемент. Щоб зв’язати певне значення з цим атрибутом використовується Blazor-атрибут @bind-value (загальна формула – @bind-[назва_атрибуту]). [6]

Зв’язування даних в Blazor можна розглянути в наступному прикладі:

```
@using Microsoft.AspNetCore.Components.Web
<div>
   <input @bind-value="@phrase" />
   <p>Значення змінної: @phrase</p>
</div>
@code {
   private string phrase = "Перемога";
}
```

Текстове поле input зв'язане зі значенням змінної phrase. В процесі рендерингу компонента елемент input отримує значення змінної phrase, а після того, як користувач введе нове значення в текстове поле і перемістить фокус на інший елемент веб-сторінки, спрацює подія onchange, і значення змінної phrase буде оновлено (рис. 3.7).

Тобто фактично цей код буде еквівалентний наступному:

```
@using Microsoft.AspNetCore.Components.Web
 <div>
   <input value="@phrase" @onchange="@((ChangeEventArgs e) =>
                                        phrase = e.Value.ToString())" />
   <p>Значення змінної: @phrase </p>
</div>
 @code {
    private string phrase = "Перемога";
}
```

Тобто, в обробнику події onchange змінній phrase присвоюється нове введене значення. Проте, завдяки атрибуту @bind розробникам не потрібно прописувати додаткову логіку, адже технологія Blazor зробить самостійно.

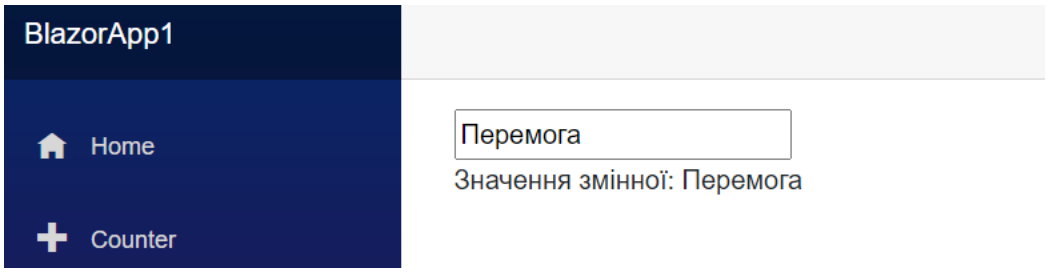


Рисунок 3.7 – Відображення результату рендерингу прикладу зв’язування даних в Blazor

Ми розглянули приклад зв’язування атрибута value та поля input, проте подібним чином можна реалізувати зв’язування з іншими атрибутами.

## 3.5 Порівняльний аналіз технології Blazor та інших сучасних фреймворків для створення клієнтських додатків

До основних переваг використання технології Blazor при створенні веб-додатків можна віднести:

- швидке та ефективне розгортання;

- відсутність необхідності використання плагінів;
- використання найновіших веб-функцій;
- впровадження залежностей;
- можливість роботи на старих браузерах;
- IntelliSense та багатий інструментарій Visual Studio;
- повне .NET налагодження;
- можливість рендерингу на стороні сервера;
- швидке оновлення безпосередньо в браузері під час розробки додатків.

Технологію Blazor доцільно розглянути в порівнянні з найбільш популярними фреймворками для створення клієнтських додатків, а саме – Angular, React, Vue.js. [17]

Технологія Angular – це фреймворк на основі JavaScript, розроблений і випущений в 2016 році Google. Даний фреймворк спрямований на спрощення розробки та тестування односторінкових додатків. Angular, як і Blazor, є фреймворком з відкритим кодом. Їх основна відмінність полягає в тому, що Angular базується на JavaScript, тоді як Blazor використовує розробку за допомогою C#. Також, на відміну від Angular, технологія Blazor все ще розробляється та змінюється, тобто є менш зрілою.

Технологія React – це бібліотека компонентів інтерфейсу на основі JavaScript, розроблена та випущена в 2013 році Facebook. Дана бібліотека вважається найкращим веб-фреймворком для розробки інтерактивних інтерфейсів користувачів. React має функцію швидкої візуалізації, що надає йому невелику перевагу над Angular. Він також включає кілька підходів до зменшення обсягу операцій DOM, що пришвидшує процес оновлення, роблячи його ще більш ефективним. React є надзвичайно популярним та поширеним, тоді як Blazor на дану мить є мало відомим.

Технологія Vue.js була розроблена Еваном Ю, колишнім співробітником Google, у 2014 році. Подібно до Angular і React, Vue – це платформа з відкритим кодом для побудови інтерфейсів користувачів та односторінкових додатків, яка

використовує JavaScript. Vue.js є другим за популярністю фреймворком після React.

Деталі порівняння фреймворків наведено в таблиці 3.2. [17]

Таблиця 3.2 – Порівняння Blazor та інших сучасних фреймворків для створення клієнтських додатків

| **Критерій** | **Blazor** | **React** | **Angular** | **Vue** |
|---|---|---|---|---|
| **Тип** | Фреймворк | Фронт-енд бібліотека | Фреймворк | Бібліотека |
| **Дата першого релізу** | 2018 | 2013 | 2016 | 2014 |
| **Розробник** | Microsoft | Facebook | Google | Evan You |
| **Ліцензія** | Apache | MIT | MIT | MIT |
| **Мова** | C#, WebAssembly | JavaScript | TypeScript | JavaScript |
| **DOM** | Інкрементний DOM | Віртуальний DOM | Звичайний DOM | Віртуальний DOM |
| **Тестування та налагодження** | Blazor Testing PlaceHolder | Jest | Jasmine | Vue DevTools |
| **Зв'язування даних** | Одностороннє | Одностороннє | Двостороннє | Двостороннє |
| **Складність вивчення** | Відносно легко | Відносно легко | Складно | Відносно легко |
| **Популярність та підтримка спільноти** | Microsoft, а також майбутня потужна спільнота учасників | Підтримка спільноти розробників Facebook та Uber | За підтримки великої кількості розробників та Google | За підтримки проекту з відкритим кодом |

Як нова платформа, Blazor забезпечує щільну інтеграцію з .NET, та створює можливість спробувати за допомогою C# досягти того, що пропонують сучасні js-фреймворки.

## 3.6 Висновки до розділу

В третьому розділі було досліджено ключові елементи технології Blazor – компоненти. Компоненти Blazor застосовується подібно до концепцій Angular,

React, VueJS. Компоненти визначають логіку рендерингу елементів інтерфейсу, а також логіку обробки дій користувача. Компоненти можуть бути вкладеними в інші компоненти. Їх можна повторно використовувати в інших проектах і переносити в вигляді бібліотеки класів Razor. Завдяки механізмам реалізації компонентів існує можливість перенесення даних між ними, обробки подій та зв’язуванні даних при створенні веб-додатків.

# 4 РЕАЛІЗАЦІЯ ІНТЕРАКТИВНОГО ВЕБ-ДОДАТКА ЗА ДОПОМОГОЮ ТЕХНОЛОГІЇ BLAZOR

## 4.1 Структура інтерактивного веб-додатка

Для написання веб-додатку була вибрана технологія Blazor Server, яка буда детально описана в попередніх розділах. Дана технологія дозволяє створювати прогресивні веб-додатки, які можуть слугувати як веб-сайтом, так і мобільним та настільним додатком. [16]

Веб-додаток для розв’язання поставленої задачі спроектовано з дотримання парадигм об’єктно-орієнтованого програмування. На рисунку 4.1 наведена діаграма класів програми:

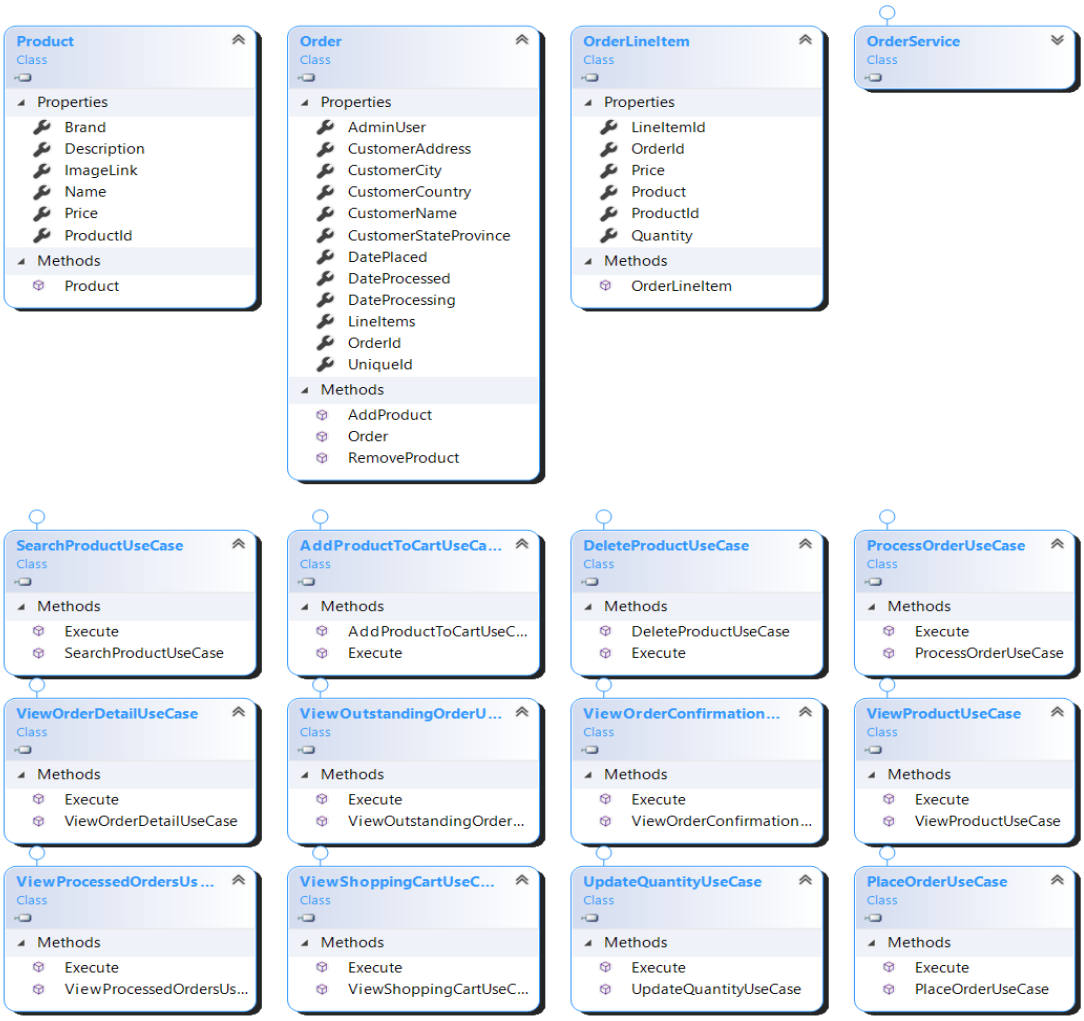


Рисунок 4.1 – Діаграма класів

Інтерактивний веб-додаток для придбання продуктів харчування буде складатись з веб-сторінки, де користувачеві буде доступний для вибору перелік товарів з можливістю пошуку по ключовому слову. З даної сторінки покупець зможе додавати потрібні товари до кошика, переходити до кошика для підтвердження замовлення або зміни кількості товару в замовленні чи видаленні зайвих товарів. Після підтвердження замовлення покупцеві відкриється форма для заповнення адреси доставки, що містить обов’язкові до заповнення поля.

Адміністративний користувач зможе авторизуватися в системі ввівши свій логін та пароль. Після авторизації він матиме доступ до переліку невиконаних та виконаних замовлень, зможе переглядати деталі замовлення та змінювати їх статус на «виконане».

На рисунку 4.2 представлена діаграма прецедентів, яка описує функції та дії акторів у веб-додатку.

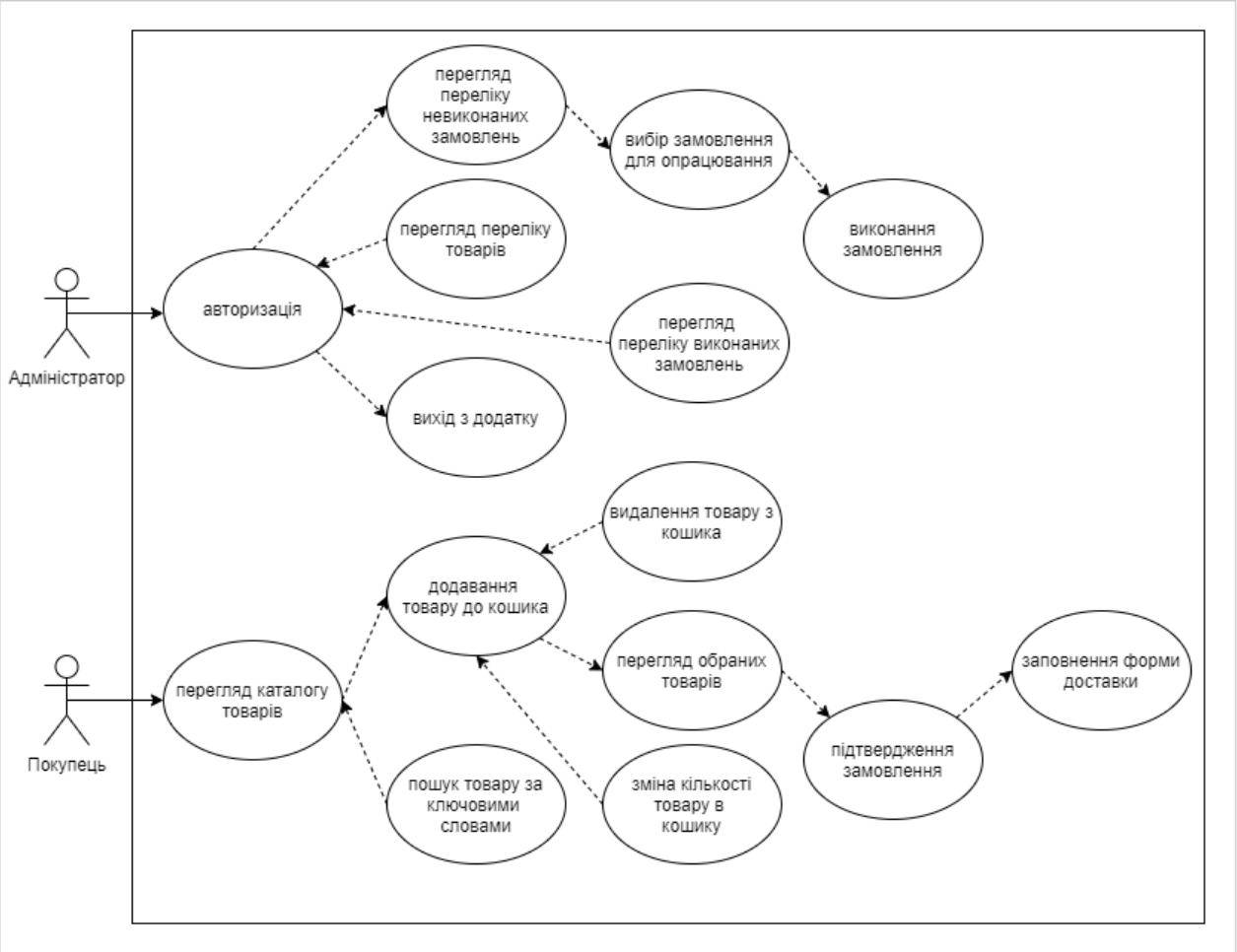


Рисунок 4.2 – Діаграма прецедентів веб-додатка Blazor

## 4.2 База даних веб-додатка Blazor

База даних інтерактивного додатка Blazor складається з трьох таблиць реляційної бази даних, які створюють єдиний інформаційний простір для зберігання та отримання доступу до даних.

Таблиця «Продукт» (Product) містить в собі всі дані щодо продуктів харчування, які доступні у веб-додатку, зокрема опис, ціну та посилання на зображення.

Таблиця «Замовлення» (Order) містить інформацію щодо кожного окремого замовлення розміщеного у веб-додатку, зокрема інформацію щодо статусу замовлення та адреси доставки.

Таблиця «Товари у замовленні» (OrderLineItem) відображає інформацію щодо кожного окремого товару в замовленні, його кількості та загальної вартості.

На рисунку 4.3 наведено схему бази даних веб-додатку.

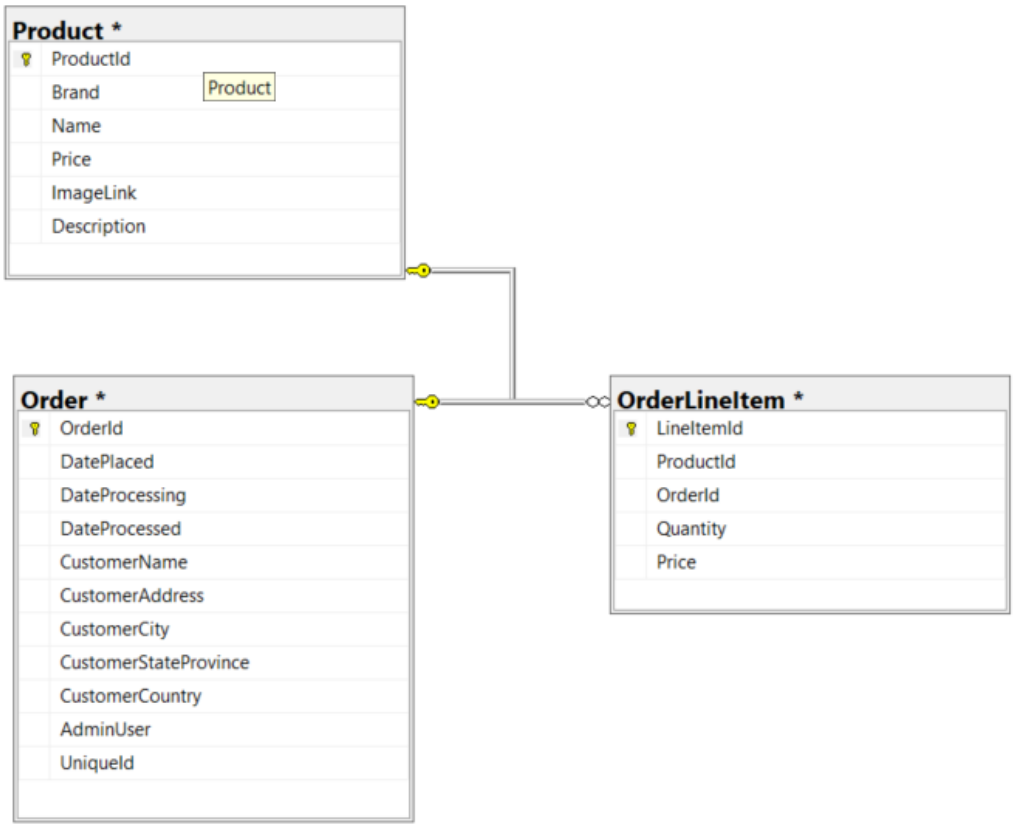


Рисунок 4.3 – Схема бази даних

## 4.3 Опис розробки клієнтського інтерфейсу та переваг веб-додатку

На головній сторінці веб-додатку Blazor розміщено зображення та опис доступних для придбання продуктів харчування, інформація про які надходить з реляційної бази даних. В правому верхньому кутку сторінки знаходиться лічильник кількості товарів, що наразі знаходяться в кошику. У верхній частині сторінки – панель пошуку (рис. 4.4).

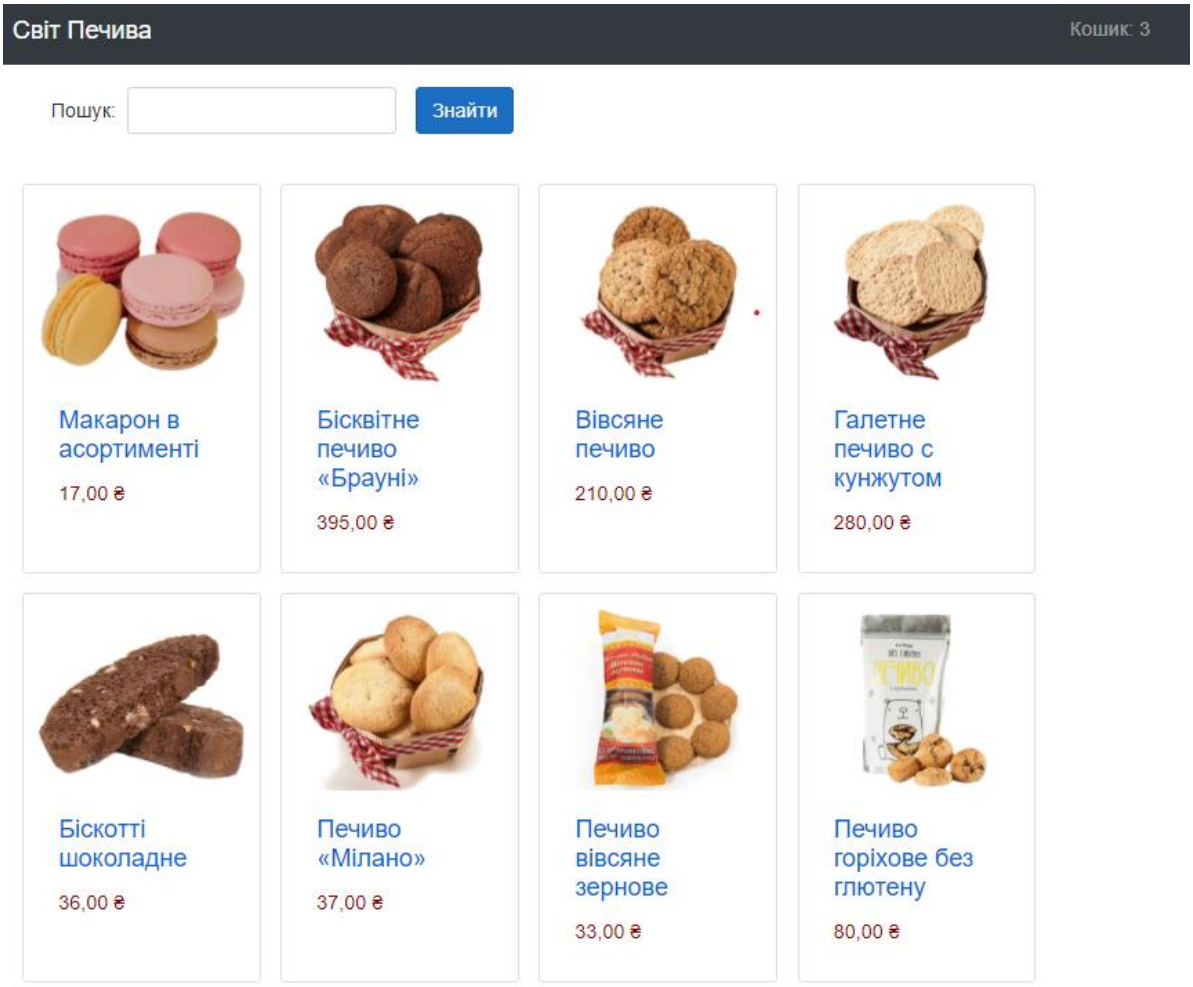


Рисунок 4.4 – Головна сторінка веб-додатка Blazor

Панель пошуку дозволяє фільтрувати відображення товарів у відповідності до введених пошукових слів або їх частин. При цьому сторінка миттєво оновлює перелік товарів, що відповідають критеріям пошуку, не перезавантажуючи інші елементи веб-додатка (рис. 4.5).

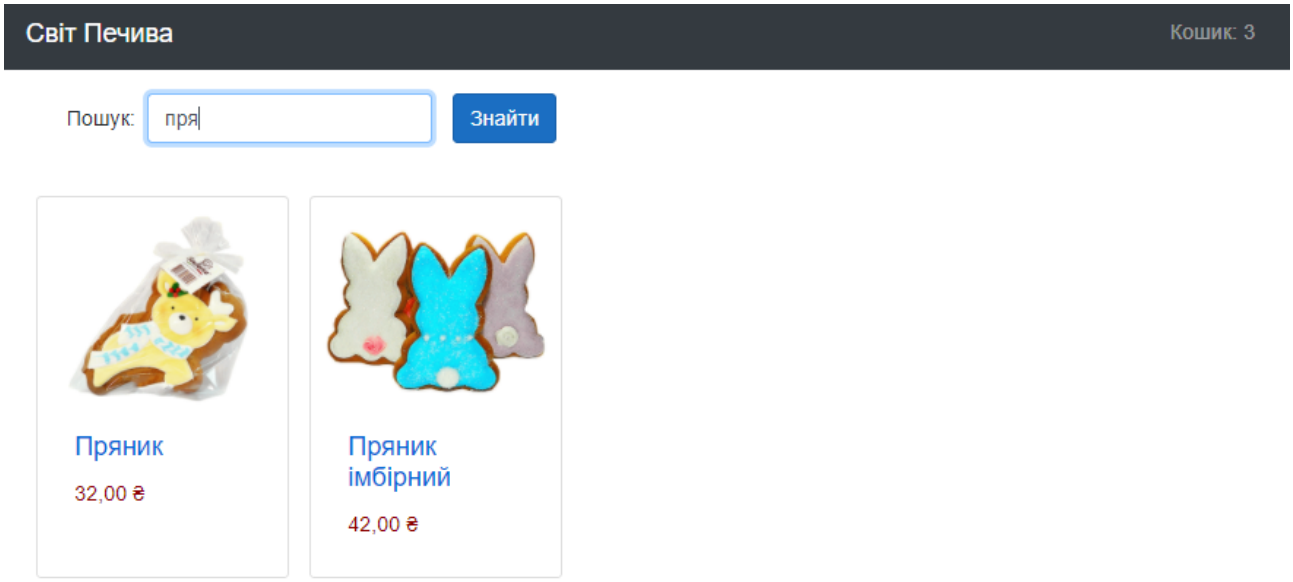


Рисунок 4.5 – Фільтрування вмісту головної сторінки веб-додатка Blazor

Натиснувши на товар, що зацікавив покупця, відкривається сторінка перегляду деталей щодо товару. Кнопка «Додати до кошика» додає товар до загального кошика, після перегляду якого користувач приймає рішення щодо замовлення (рис. 4.6).

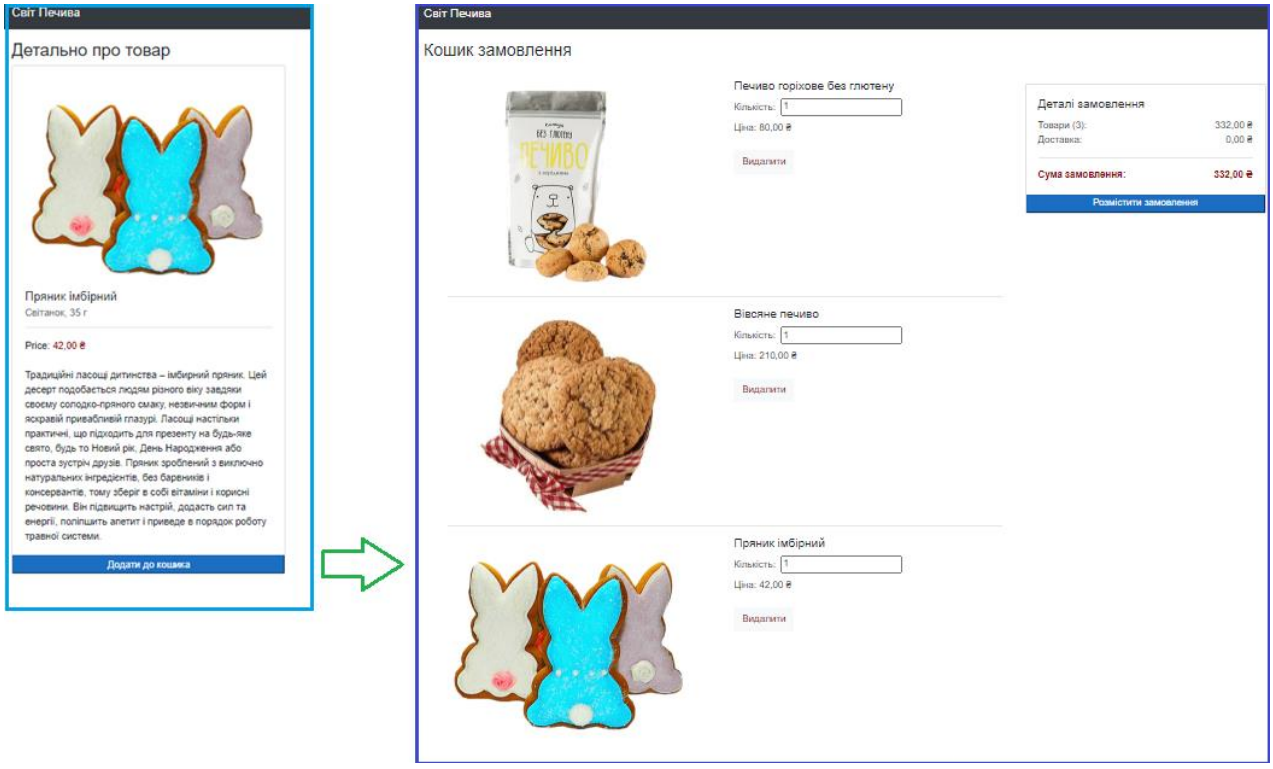


Рисунок 4.6 – Додавання товару до кошика та перегляд його вмісту

Товарами в кошику зручно керувати за допомогою інтерактивного поля зміни кількості товару в замовленні та кнопок, що дозволяють видалити непотрібний товар із замовлення. При цьому автоматично перераховується та оновлюється сума замовлення в деталях замовлення (рис. 4.7).

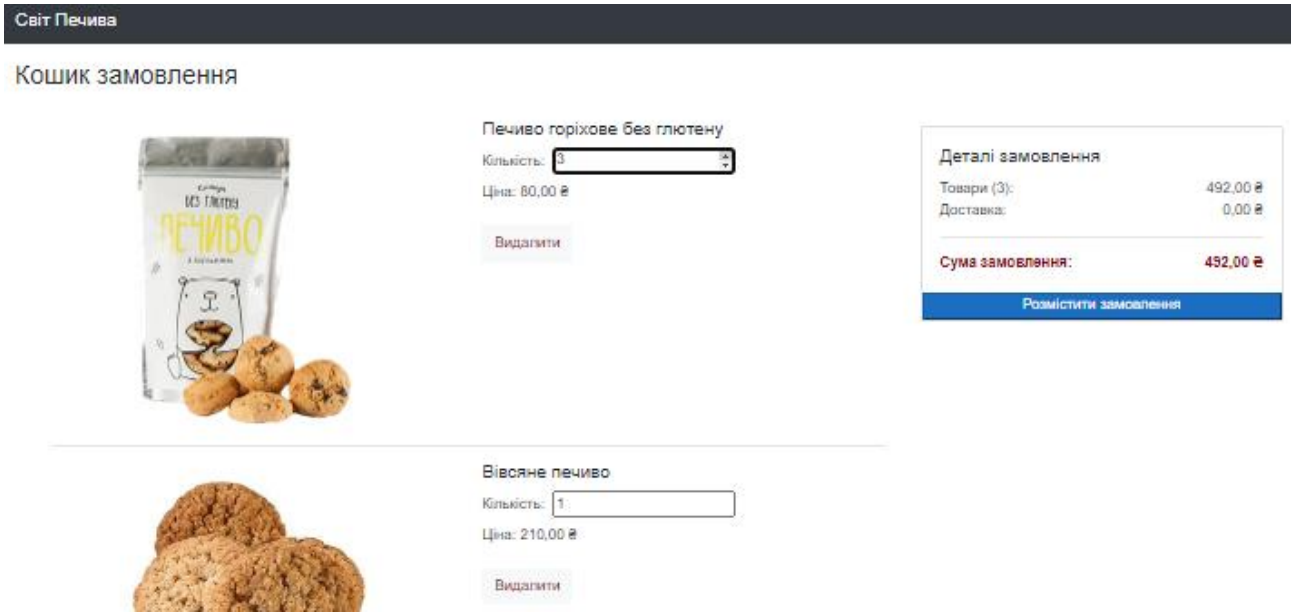


Рисунок 4.7 – Зміна вмісту кошика

Підтвердивши замовлення, користувач перенаправляється на форму оформлення замовлення, де потрібно ввести адресу доставки. Обов’язкові до заповнення поля після внесення до них даних виділяються зеленою рамкою, а не заповнені – червоною (рис. 4.8).

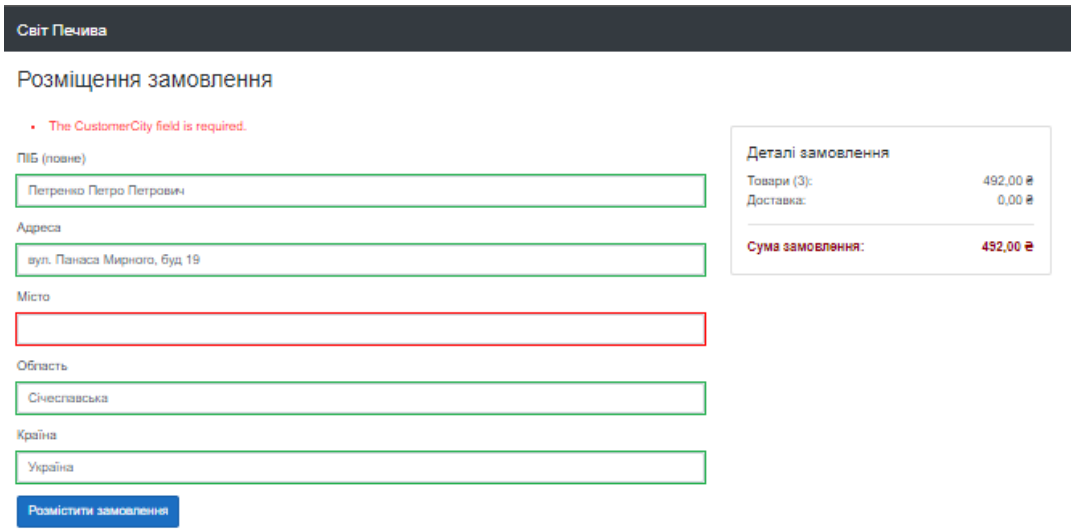


Рисунок 4.8 – Заповнення обов’язкових полів та розміщення замовлення

Після здійснення авторизації адміністративний користувач, перейшовши на вкладку «Невиконані замовлення», отримує перелік замовлень, які потрібно переглянути та опрацювати. Опрацьовані замовленні за потреби можна знайти на вкладці «Опрацьовані замовлення» (рис. 4.9)

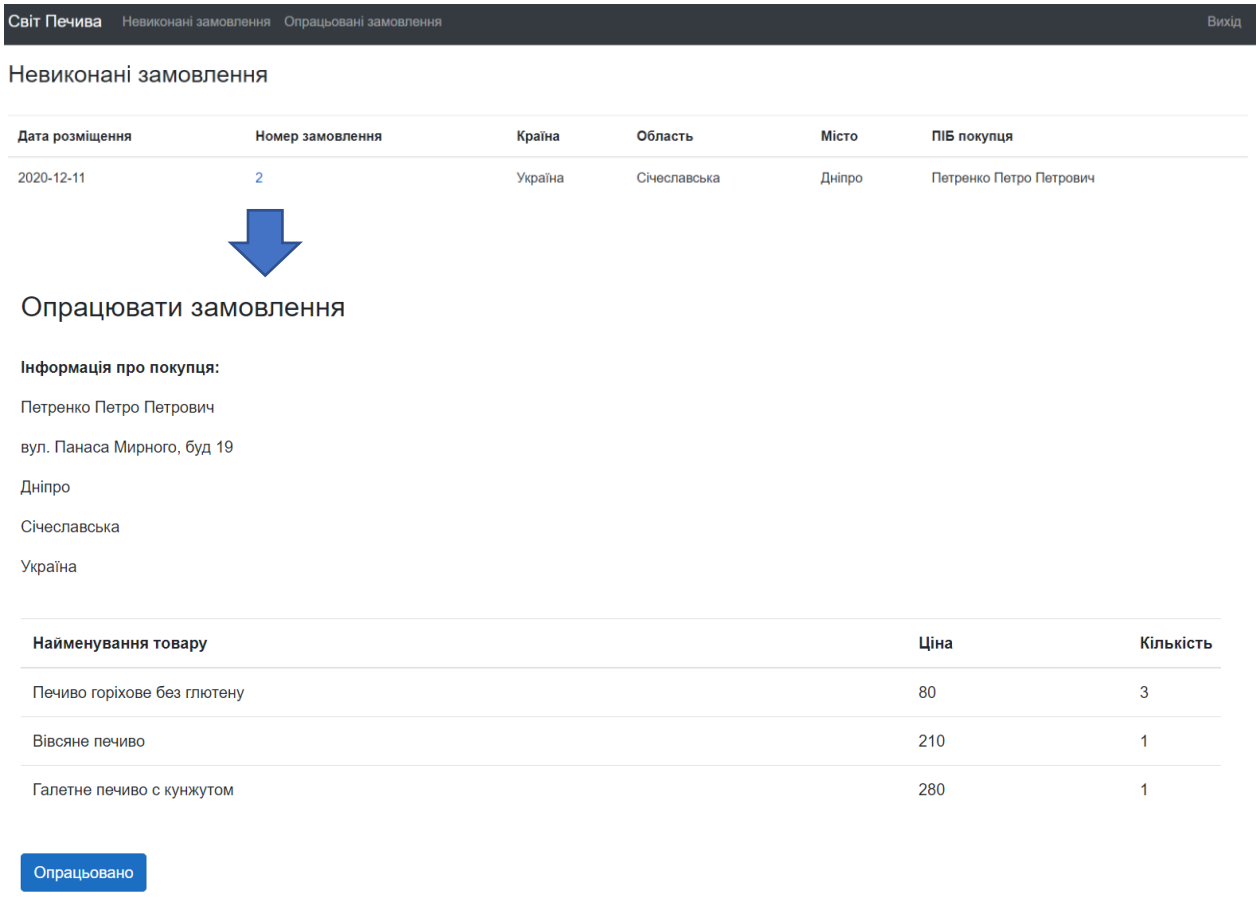


Рисунок 4.9 – Опрацювання замовлення адміністратором

Після завершення роботи адміністративний користувач натискає на кнопку «Вихід» та завершує роботу з веб-додатком.

Лістинг коду основних компонентів веб-додатку Blazor наведено в Додатку А.

## 4.4 Висновки до розділу

У четвертому розділі було розглянуто структуру побудови інтерактивного веб-додатка, описано функції та дії користувачів в ньому, наведено схему бази

даних. Наведено приклади та елементи клієнтського інтерфейсу при виконанні різних дій користувачами, зазначено сильні сторони інтерактивного веб-додатку, створеного за допомогою технології Blazor на платформі ASP.NET Core.

# 5 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

У даному розділі проводиться оцінка основних характеристик інтерактивного веб-додатка, призначеного для продажу продуктів харчування. Інтерфейс користувача був розроблений за допомогою фреймворку ASP.NET Core на мові програмування C# в середовищі розробки Microsoft Visual Studio 2019. Інтерфейс користувача створений за допомогою технології Blazor.

Веб-додаток призначено для використання на персональних комп'ютерах під управлінням операційних систем Windows, Mac OS, Linux.

Нижче наведено аналіз різних варіантів реалізації програмного продукту з метою вибору оптимальної, з огляду як на економічні фактори, так і на характеристики продукту, що впливають на продуктивність роботи і на його сумісність з апаратним забезпеченням. Для цього було використано апарат функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) – це технологія, яка дозволяє оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії. Як прямі, так і побічні витрати розподіляються по продуктам та послугам у залежності від потрібних на кожному етапі виробництва обсягів ресурсів. Виконані на цих етапах дії у контексті метода ФВА називаються функціями.

Мета ФВА полягає у забезпеченні правильного розподілу ресурсів, виділених на виробництво продукції або надання послуг, на прямі та непрямі витрати. У даному випадку – аналізу функцій програмного продукту й виявлення усіх витрат на реалізацію цих функцій.

Фактично цей метод працює за таким алгоритмом:

– визначається послідовність функцій, необхідних для виробництва продукту. Спочатку – всі можливі, потім вони розподіляються по двом групам: ті, що впливають на вартість продукту і ті, що не впливають. На цьому ж етапі

оптимізується сама послідовність скороченням кроків, що не впливають на цінність і відповідно витрат.

– для кожної функції визначаються повні річні витрати й кількість робочих часів.

– для кожної функції на основі оцінок попереднього пункту визначається кількісна характеристика джерел витрат.

– після того, як для кожної функції будуть визначені їх джерела витрат, проводиться кінцевий розрахунок витрат на виробництво продукту.

## 5.1 Постановка задачі техніко-економічного аналізу

У роботі застосовується метод ФВА для проведення техніко-економічного аналізу розробки інтерактивного веб-додатку, побудованого за допомогою технології Blazor. Оскільки основні проектні рішення стосуються всього програмного продукту, кожна окрема підсистема має їм задовольняти. Тому фактичний аналіз представляє собою аналіз функцій інтерактивного веб-додатку, призначеного для продажу продуктів харчування.

Відповідно цьому варто обирати і систему показників якості програмного продукту.

Технічні вимоги до продукту наступні:

– програмний продукт повинен функціонувати на персональних комп’ютерах із стандартним набором компонент;

– забезпечувати високу швидкість обробки великої кількості тразакцій у реальному часі;

– забезпечувати зручність і простоту взаємодії з користувачем, зокрема з адміністраторами ресурсу;

– передбачати мінімальні витрати на впровадження програмного продукту.

### 5.1.1 Обґрунтування функцій програмного продукту

Головна функція $F_0$ – розробка програмного продукту (ПП), який дозволяє користувачам зручно та швидко знаходити та купувати продукти харчування. Виходячи з конкретної мети, можна виділити наступні основні функції ПП:

$F_1$ – вибір мови програмування;

$F_2$ – робота з базою даних товарів та замовлень;

$F_3$ – вибір моделі хостингу.

Кожна з основних функцій може мати декілька варіантів реалізації.

Функція $F_1$:

а) мова програмування C#;

б) мова програмування HTML;

Функція $F_2$:

а) робота з СУБД MS SQL;

б) робота з СУБД PostgeSQL.

Функція $F_3$:

а) веб-додаток, створений за технологією Blazor Server;

б) веб-додаток, створений за технологією Blazor WebAssembly

### 5.1.2 Варіанти реалізації основних функцій

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 5.1). На основі цієї карти побудовано позитивно-негативну матрицю варіантів основних функцій (таблиця 5.1).

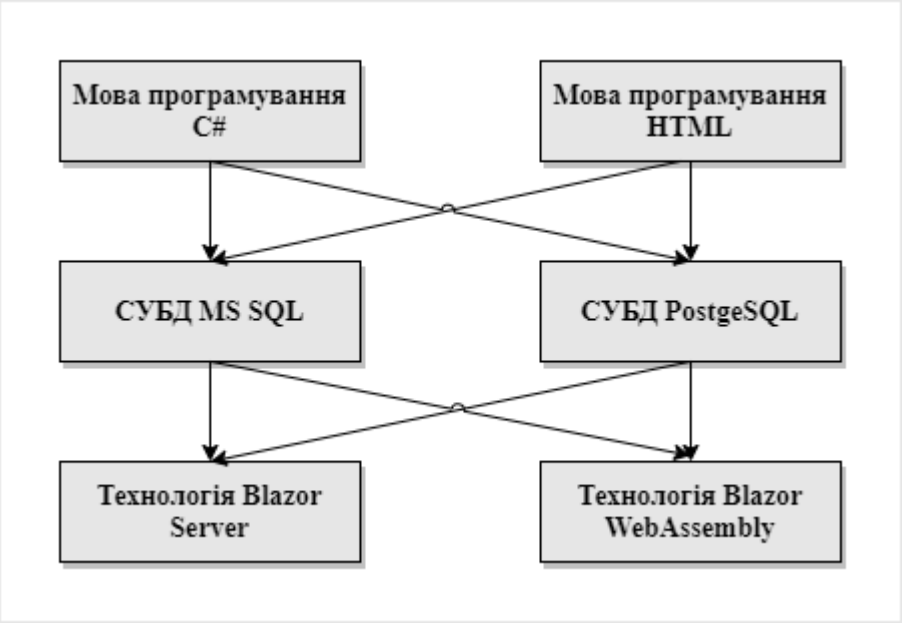


Рисунок 5.1 – Морфологічна карта

Морфологічна карта відображує всі можливі комбінації варіантів реалізації функцій, які складають повну множину варіантів ПП.

Таблиця 5.1 – Позитивно-негативна матриця

| **Основні функції** | **Варіанти реалізації** | **Переваги** | **Недоліки** |
|---|---|---|---|
| *F1* | *А* | Займає менше часу при написанні коду, кросплатформений | Кросплатформеність наразі лише для розробки на ASP.NET Core |
| | *Б* | Код швидко виконується, кросплатформений | Займає більше часу при написанні коду |
| *F2* | А | Високоефективна сучасна СУБД | Безкоштовна версія має обмеження |
| | Б | Сучасна безкоштовна СУБД | Неприйнятна для деяких великих корпоративних клієнтів |
| *F3* | *А* | Швидкість розрахунків та інших операцій | Загальна ефективність залежить від каналів зв’язку |
| | *Б* | Канали зв’язку слабко впливають на роботу веб-додатка після його завантаження | Швидкість розрахунків та інших операцій залежить від потужності ПК користувача |

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому, що вони не відповідають поставленим перед програмним продуктом задачам. Ці варіанти відзначені у морфологічній карті.

Функція *F1:*

Оскільки технологія Blazor розроблена та оптимізована для мови програмування C#, то реалізація на чистому HTML буде мало ефективною, тому варіант б) має бути відкинутий.

Функція *F2:*

Оскільки планується розглядати широкий спектр різноманітних варіантів розгортання СУБД для клієнтів різного рівня, то перший варіант є більш зручним в даному випадку, а отже то варіант б) має бути відкинутий

Функція *F3:*

Обидві технології забезпечують повну функціональність програмного продукту, тому вважаємо варіанти а) та б) гідними розгляду.

Таким чином, будемо розглядати такі варіанти реалізації ПП:

1. F1а – F2а – F3а
2. F1а – F2а – F3б

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

## 5.2 Обґрунтування системи параметрів ПП

### 5.2.1 Опис параметрів

На підставі даних про основні функції, що повинен реалізувати програмний продукт, вимог до нього, визначаються основні параметри виробу, що будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

– *X1* – швидкодія мови програмування;

– *X2* – об’єм пам’яті для збереження даних;

– *X3* – час обробки даних;

– *X4* – потенційний об’єм програмного коду.

*X1*: Відображає швидкодію операцій залежно від обраної мови програмування.

*X2:* Відображає об’єм пам’яті в оперативній пам’яті персонального комп’ютера, необхідний для збереження та обробки даних під час виконання програми.

*X3:* Відображає час, який витрачається на дії.

*X4:* Показує розмір програмного коду який необхідно створити безпосередньо розробнику.

### 5.2.2 Кількісна оцінка параметрів

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію ПП як показано у табл. 5.2.

Таблиця 5.2 – Основні параметри ПП

| **Назва Параметра** | **Умовні позначення** | **Одиниці виміру** | **Значення параметра** | | |
|---|---|---|---|---|---|
| | | | **гірші** | **середні** | **кращі** |
| Швидкодія мови програмування | X1 | Оп/мс | 14000 | 8000 | 1000 |
| Об’єм пам’яті для збереження даних | X2 | Мб | 64 | 32 | 16 |
| Час обробки даних сторінки | X3 | мс | 1000 | 500 | 80 |
| Потенційний об’єм програмного коду | X4 | кількість строк коду | 3000 | 2000 | 1000 |

За даними таблиці 4.2 будуються графічні характеристики параметрів – рис. 5.2 – рис. 5.5:

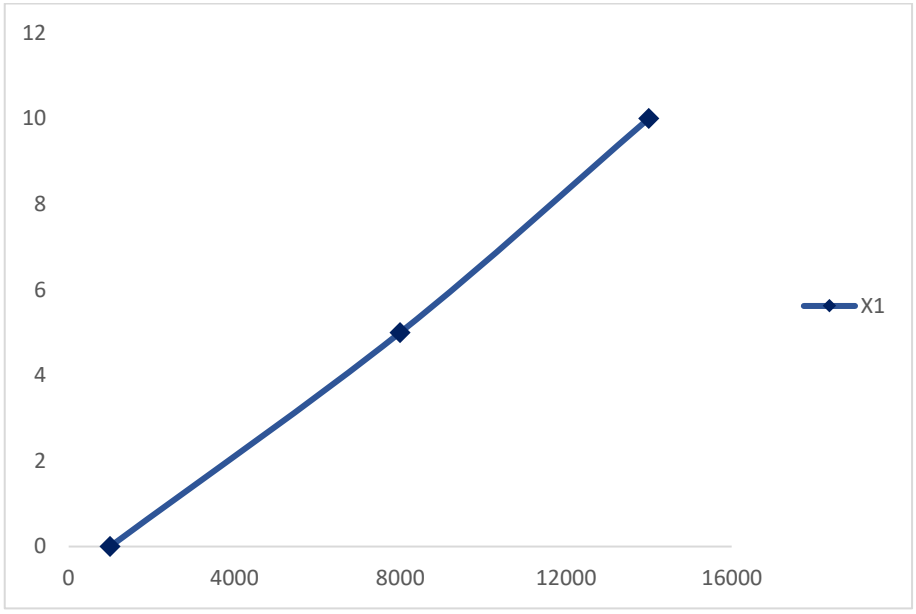


Рисунок 5.2 – X1, швидкодія мови програмування

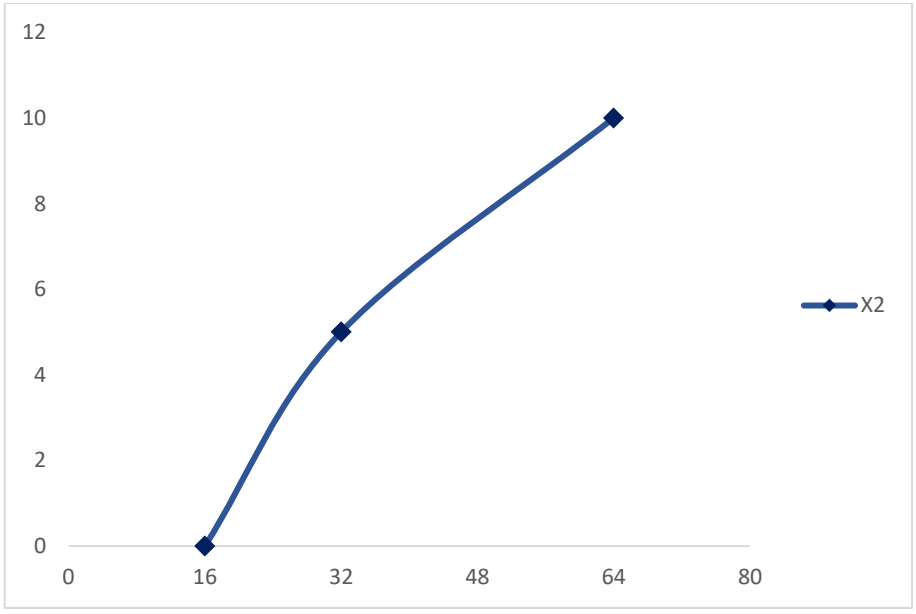


Рисунок 5.3 – X2, об’єм пам’яті для збереження даних

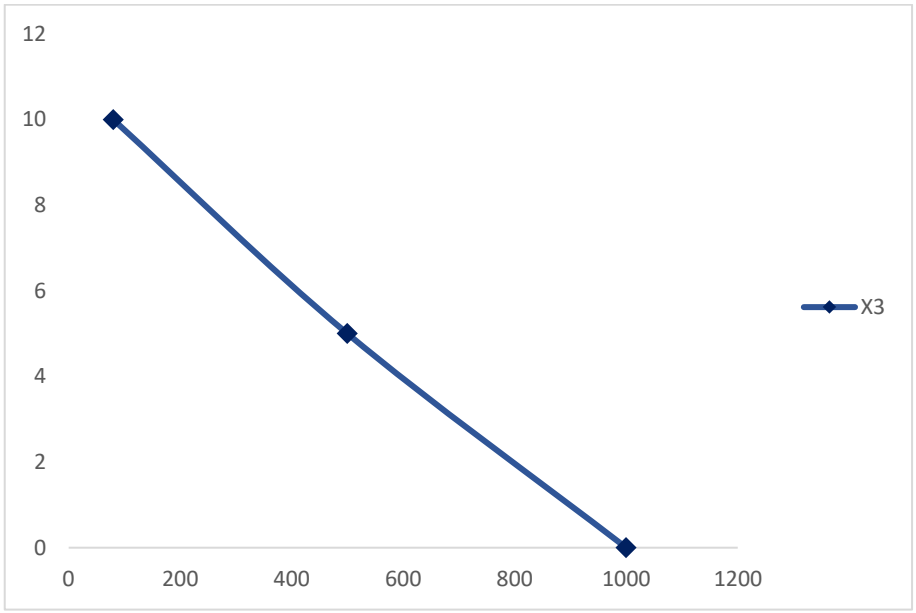


Рисунок 5.4 – X3, час обробки даних сторінки

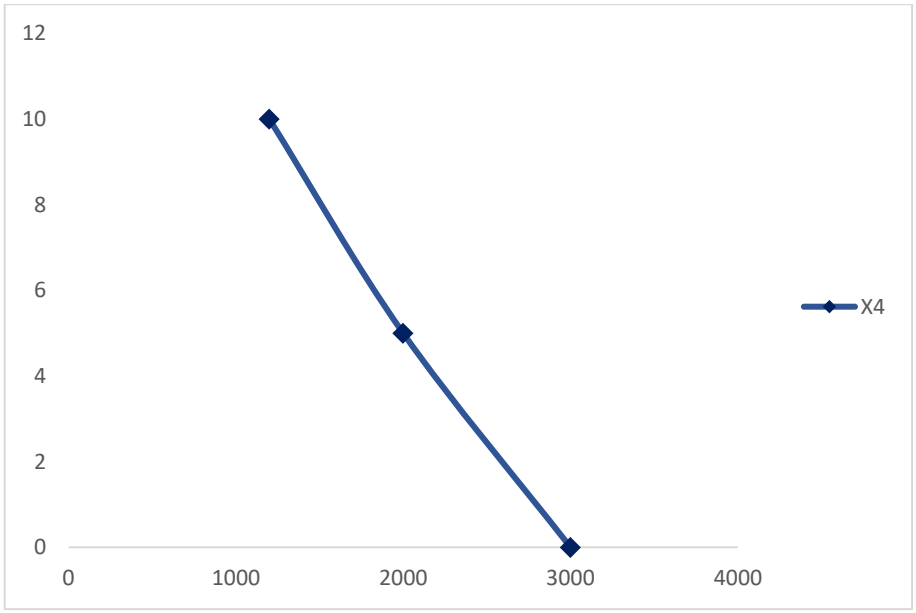


Рисунок 5.5 – X4, потенційний об’єм програмного коду

### 5.2.3 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробки

програмного продукту, який дозволяє користувачам зручно та швидко знаходити та купувати продукти харчування.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значимості передбачає:

– визначення рівня значимості параметра шляхом присвоєння різних рангів;

– перевірку придатності експертних оцінок для подальшого використання;

– визначення оцінки попарного пріоритету параметрів;

– обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 5.3.

Таблиця 5.3 – Результати ранжування параметрів

| Позначення параметра | Назва параметра | Одиниці виміру | Ранг параметра за оцінкою експерта | | | | | | | Сума рангів $R_i$ | Відхи-лення $\Delta_i$ | $\Delta_i^2$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | | | |
| *X1* | Швидкодія мови програму-вання | Оп/мс | 2 | 1 | 2 | 3 | 2 | 1 | 2 | 13 | -4,5 | 20,25 |
| *X2* | Об’єм пам’яті для збереження даних | Мб | 3 | 3 | 4 | 4 | 3 | 3 | 3 | 23 | 5,5 | 30,25 |
| *X3* | Час обробки даних сторінки | Мс | 1 | 2 | 1 | 1 | 1 | 2 | 1 | 9 | -8,5 | 72,25 |
| *X4* | Потенційний об’єм програмного коду | К-сть строк коду | 4 | 4 | 3 | 2 | 4 | 4 | 4 | 25 | 7,5 | 56,25 |
| | Разом | | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 70 | 0 | 179 |

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 70$$

де $N$ – число експертів, $n$ – кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 17{,}5$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

г)загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 179$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 179}{7^2(4^3 - 4)} = 0{,}73 > W_k = 0{,}67$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 5.4.

Таблиця 5.4 – Попарне порівняння параметрів

| Параметри | Експерти | | | | | | | Кінцева оцінка | Числове значення |
|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | | |
| X1 і X2 | < | < | < | < | < | < | < | < | 0,5 |
| X1 і X3 | > | < | > | > | > | < | > | > | 1,5 |
| X1 і X4 | < | < | < | > | < | < | < | < | 0,5 |
| X2 і X3 | > | > | > | > | > | > | > | > | 1,5 |
| X2 і X4 | < | < | > | > | < | < | < | < | 0,5 |
| X3 і X4 | < | < | < | < | < | < | < | < | 0,5 |

Числове значення, що визначає ступінь переваги *i*–го параметра над *j*–тим, $a_{ij}$ визначається по формулі:

$$a_{ij} = \begin{cases} 1{,}5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j \\ 0.5 \text{ при } X_i < X_j \end{cases}$$

З отриманих числових оцінок переваги складемо матрицю $A = \| a_{ij} \|$.

Для кожного параметра зробимо розрахунок вагомості $K_{вi}$ за наступними формулами:

$$K_{\text{вi}} = \frac{b_i}{\sum_{i=1}^{n} b_i}, \text{де } b_i = \sum_{i=1}^{N} a_{ij}.$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%).На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{\text{вi}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'}, \text{де } b_i' = \sum_{i=1}^{N} a_{ij} b_j.$$

Як видно з таблиці 5.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 5.5 – Розрахунок вагомості параметрів

| Параметри $x_i$ | Параметри $x_j$ | | | | Перша ітер. | | Друга ітер. | | Третя ітер | |
|---|---|---|---|---|---|---|---|---|---|---|
| | X1 | X2 | X3 | X4 | $b_i$ | $K_{\text{вi}}$ | $b_i^1$ | $K_{\text{вi}}^1$ | $b_i^2$ | $K_{\text{вi}}^2$ |
| X1 | 1,0 | 0,5 | 1,5 | 0,5 | 3,5 | 0,219 | 21,38 | 0,216 | 93,525 | 0,215 |
| X2 | 1,5 | 1,0 | 1,5 | 0,5 | 4,5 | 0,281 | 27,92 | 0,282 | 123,11 | 0,283 |
| X3 | 0,5 | 0,5 | 1,0 | 0,5 | 2,5 | 0,156 | 15,35 | 0,155 | 66,99 | 0,154 |
| X4 | 1,5 | 1,5 | 1,5 | 1,0 | 5,5 | 0,344 | 34,25 | 0,346 | 151,38 | 0,348 |
| Всього: | | | | | 16 | 1 | 99 | 1 | 435 | 1 |

## 5.3 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів *X2* (об'єм пам'яті для збереження даних) та *X1* (швидкодія мови програмування) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра *X3 (*час обробки даних сторінки) обрано не найгіршим (не максимальним), тобто це значення відповідає або варіанту а) 1000 мс або варіанту б) 80мс.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 5.6):

$$K_K(j) = \sum_{i=1}^{n} K_{вi,j} B_{i,j} ,$$

де $n$ – кількість параметрів;$K_{вi}$– коефіцієнт вагомості $i$–го параметра;$B_i$ – оцінка $i$–го параметра в балах.

Таблиця 5.6 – Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

| Основні функції | Варіант реалізації функції | Абсолютне значення параметра | Бальна оцінка параметра | Коефіцієнт вагомості параметра | Коефіцієнт рівня якості |
|---|---|---|---|---|---|
| F1(X1) | А | 8000 | 1,9 | 0,215 | 0,409 |
| F2(X2) | А | 32 | 3,3 | 0,283 | 0,934 |
| F3(X3,X4) | А | 1000 | 2,4 | 0,154 | 0,370 |
| | Б | 80 | 1 | 0,348 | 0,348 |

За даними з таблиці 4.6 за формулою

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \ldots + K_{\text{ТУ}}[F_{zk}],$$

визначаємо рівень якості кожного з варіантів:

$K_{K1}$ = 0,409 + 0,934 + 0,370 = 1,713

$K_{K2}$ = 0,409 + 0,934 + 0,348 = 1,691

Як видно з розрахунків, кращим є перший варіант, для якого коефіцієнт технічного рівня має найбільше значення.

## 5.4 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. Розробка проекту програмного продукту;

2. Розробка програмної оболонки;

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 3.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань. Загальна трудомісткість обчислюється як

$$T_O = T_P \cdot K_П \cdot K_{СК} \cdot K_M \cdot K_{СТ} \cdot K_{СТ.М},$$

де $T_P$ – трудомісткість розробки ПП;

$K_П$ – поправочний коефіцієнт;

$K_{СК}$ – коефіцієнт на складність вхідної інформації;

$K_M$ – коефіцієнт рівня мови програмування;

$K_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;

$K_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення.

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: $T_P$ =90 людино-днів. Поправочний коефіцієнт, який враховує вид

нормативно-довідкової інформації для першого завдання: $К_П$ = 1,7. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 1: $К_{СК}$ = 1. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $К_{СТ}$=0,8. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$Т_1$ = 90 * 1,7 * 0,8 = 122,4 людино-днів.

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $Т_Р$ =27 людино-днів, $К_П$ =0,9, $К_{СК}$ = 1, $К_{СТ}$=0,8:

$Т_2$ = 27 · 0,9 · 0,8 = 19,44 людино-днів.

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$Т_I$ = (122,4 + 19,44 + 4,8 + 19,44) · 8 = 1328,64 людино-годин;

$Т_{II}$ = (122,4 + 19,44 + 6,91 + 19,44) · 8 = 1345,52 людино-годин;

Найбільш високу трудомісткість має варіант II.

В розробці беруть участь два програмісти з окладом 15 200 грн. Визначимо зарплату за годину за формулою:

$$С_ч = \frac{М}{T_m \cdot t} \text{ грн},$$

де М – місячний оклад працівників;$T_m$ – кількість робочих днів тиждень; $t$ – кількість робочих годин в день.

$$С_ч = \frac{2 \cdot 15\,200}{2 \cdot 21 \cdot 8} = 90{,}48 \text{ грн}$$

Тоді, розрахуємо заробітну плату за формулою

$$С_{ЗП} = С_ч \cdot Т_i \cdot К_Д,$$

де $С_ч$– величина погодинної оплати праці програміста;$Т_i$ – трудомісткість відповідного завдання; $К_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $С_{ЗП} = 90,48 \cdot 1328,64 \cdot 1,2 = 144\ 258,42$ грн

II. $С_{ЗП} = 90,48 \cdot 1345,52 \cdot 1,2 = 146\ 091,18$ грн

Відрахування на єдиний соціальний внесок становить 22%:

I. $С_{ВІД} = С_{ЗП} \cdot 0,22 = 144\ 258,42 \cdot 0,22 = 31\ 736,85$ грн

II. $С_{ВІД} = С_{ЗП} \cdot 0,22 = 146\ 091,18 \cdot 0,22 = 32\ 140,06$ грн

Тепер визначимо витрати на оплату однієї машино-години. ($С_М$)

Так як одна ЕОМ обслуговує одного програміста з окладом 15 200 грн, з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 15\ 200 \cdot 0,2 = 36\ 480$ грн

З урахуванням додаткової заробітної плати:

$С_{ЗП} = С_Г \cdot (1 + К_З) = 36\ 480 \cdot (1 + 0,2) = 43\ 776$ грн

Відрахування на єдиний соціальний внесок:

$С_{ВІД} = С_{ЗП} \cdot 0,22 = 43\ 776 \cdot 0,22 = 9\ 630,72$ грн

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 18 000 грн.

$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1,15 \cdot 0,25 \cdot 18\ 000 = 5\ 175$ грн,

де $К_{ТМ}$– коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

$К_А$– річна норма амортизації;

$Ц_{ПР}$– договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1,15 \cdot 18\ 000 \cdot 0,05 = 1\ 035$ грн,

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$Т_{ЕФ} = (Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 104 – 8 – 16) \cdot 8 \cdot 0,9 = 1706,4$ годин,

де $Д_К$ – календарна кількість днів у році;

$Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів;

$Д_Р$ – кількість днів планових ремонтів устаткування;

t –кількість робочих годин в день; $К_В$– коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 1706,4 \cdot 0,4 \cdot 0,67 \cdot 2,48336 = 1\ 135,68$ грн,

де $N_С$ – середньо-споживча потужність приладу;

$К_З$– коефіцієнтом зайнятості приладу;

$Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$С_Н = Ц_{ПР} \cdot 0.67 = 18\ 000 \cdot 0,67 = 12\ 060$ грн

Тоді, річні експлуатаційні витрати будуть:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н$$

$С_{ЕКС} = 43\ 776 + 9\ 630,72 + 5\ 175 + 1\ 035 + 1\ 135,68 + 12\ 060 = 72\ 812,4$ грн

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$С_{М\text{-}Г} = С_{ЕКС} / Т_{ЕФ} = 72\ 812,4/1706,4 = 42,67$ грн/год

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$С_М = С_{М\text{-}Г} \cdot Т$$

I. $С_М = 42,67 \cdot 1328,64 = 56\ 693,07$ грн

II. $С_М = 42,67 \cdot 1345,52 = 57\ 413,34$ грн

Накладні витрати складають 67% від заробітної плати:

$$С_Н = С_{ЗП} \cdot 0,67$$

I. $С_Н = 144\ 258,42 \cdot 0,67 = 96\ 653,14$ грн

II. $С_Н = 146\ 091,18 \cdot 0,67 = 97\ 881,09$ грн

Отже, вартість розробки ПП за варіантами становить:

$$С_{ПП} = С_{ЗП} + С_{ВІД} + С_М + С_Н$$

I. $С_{ПП} = 144\ 258,42 + 31\ 736,85 + 56\ 693,07 + 96\ 653,14 = 329\ 341,48$ грн

II. $С_{ПП} = 146\ 091,18 + 32\ 140,06 + 57\ 413,34 + 97\ 881,09 = 333\ 525,67$ грн

## 5.5 Вибір кращого варіанта ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕРj} = К_{Кj} / С_{Фj},$$

$К_{ТЕР1} = 5{,}214 / 329\,341{,}48 = 0{,}16 \cdot 10^{-4}$;

$К_{ТЕР2} = 3{,}569 / 333\,525{,}67 = 0{,}11 \cdot 10^{-4}$;

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1} = 0{,}16 \cdot 10^{-4}$.

## 5.6 Висновки до розділу

В даному розділі проведено повний функціонально-вартісний аналіз ПП, який було розроблено в рамках дипломного проекту. Процес аналізу можна умовно розділити на дві частини.

В першій з них проведено дослідження ПП з технічної точки зору: було визначено основні функції ПП та сформовано множину варіантів їх реалізації; на основі обчислених значень параметрів, а також експертних оцінок їх важливості було обчислено коефіцієнт технічного рівня, який і дав змогу визначити оптимальну з технічної точки зору альтернативу реалізації функцій ПП.

Другу частину ФВА присвячено вибору із альтернативних варіантів реалізації найбільш економічно обґрунтованого. Порівняння запропонованих варіантів реалізації в рамках даної частини виконувалось за коефіцієнтом ефективності, для обчислення якого були обчислені такі допоміжні параметри, як трудомісткість, витрати на заробітну плату, накладні витрати.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є перший варіант реалізації програмного продукту. У

нього виявився найкращий показник техніко-економічного рівня якості $К_{ТЕР} = 0,16 \cdot 10^{-4}$.

Цей варіант реалізації програмного продукту має такі параметри:

– мова програмування – C#;

– використання СУБД MS SQL;

– веб-додаток, створений за технологією Blazor Server.

Даний варіант створення інтерактивного веб-додатка дає користувачу зручний інтерфейс, гарний функціонал і швидкодію.

# ВИСНОВКИ

В дипломній роботі проведено дослідження технології Blazor для розробки інтерактивних веб-додатків. Наведено приклади реалізації компонентів Blazor, їх складових елементів та механізмів взаємодії.

Проаналізовано розвиток платформи .NET Core та фреймворку Blazor, як проекту з відкритим вихідним кодом, та переваги які він надає розробникам, зокрема, можливість написання коду веб-додатків за допомогою C# замість JavaScript, використання можливостей екосистеми .NET, використання загальної логіки як клієнтською, так і серверною частиною програми, а також використання шаблонів Visual Studio для спрощення створення додатків.

Досліджено роль компонентів Blazor в розробці інтерактивних веб-додатків, їх будова, розміщення, способи виклику та механізми перенесення даних між ними.

На прикладі спроектованого додатка для продажу продуктів харчування було визначено переваги та недоліки технології Blazor при розробці інтерактивних веб-додатків на платформі ASP.NET Core. Створений веб-додаток може використовуватися як веб-сайт, а за необхідності, після незначної модифікації, як мобільний або настільний додаток.

# ПЕРЕЛІК ПОСИЛАНЬ

1. Коноваленко І.В. Програмування мовою C# 6. – Тернопіль: ТНТУ, 2016. – 227 с.

2. Albahari J. C# 8.0 in a Nutshell: The Definitive Reference. – O'Reilly, 2020. – 1102 p.

3. Aponte M. Building single page applications in .NET Core 3: Jumpstart Coding Using Blazor and C#. Apress, 2020. – 104 p.

4. Charbeneau E. Blazor: A Beginners GuideA quick-start guide to productivity with Blazor. Progress, 2020. – 121 p.

5. Freeman A. Pro ASP.NET Core 3: Develop Cloud-Ready Web Applications Using MVC, Blazor, and Razor Pages. Apress, 2020. – 1080 p.

6. Himschoot P. Microsoft Blazor: Building Web Applications in .NET. Apress, 2020. – 277 p.

7. Litvinavicius T. Exploring Blazor: Creating Hosted, Server-side, and Client-side Applications with C#. – Apress, 2019. – 199 p.

8. Sharp J. Microsoft Visual C# step by step. – Pearson Education: Microsoft Press, 2018. – 1034 p.

9. Awesome Blazor [Електронний ресурс] – Режим доступу: https://github.com/AdrienTorris/awesome-blazor

10. Blazor: что такое Blazor [Електронний ресурс] – Режим доступу: https://metanit.com/sharp/blazor/1.1.php

11. ASP.NET Core Blazor Hosting Models [Електронний ресурс] – Режим доступу: https://docs.microsoft.com/en-us/aspnet/core/blazor/hosting-models

12. ASP.NET Documentation [Електронний ресурс] – Режим доступу: https://docs.microsoft.com/en-us/aspnet/core/

13. C# documentation [Електронний ресурс] – Режим доступу: https://docs.microsoft.com/en-us/dotnet/csharp/

14. Blazor – Build client web apps with C# [Електронний ресурс] – Режим доступу: https://dotnet.microsoft.com/apps/aspnet/web-apps/blazor

15. Introduction to ASP.NET Core Blazor [Електронний ресурс] – Режим доступу: https://docs.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-5.0

16. Complete Blazor Course - e-Commerce App & Clean Architecture [Електронний ресурс] – Режим доступу: https://www.udemy.com/course/complete-blazor-course-e-commerce-app-clean-architecture

17. What is Blazor vs (Angular, React, and Vue) [Електронний ресурс] – Режим доступу: https://devathon.com/blog/blazor-vs-angular-vs-react-vs-vue/

# Додаток А

# Лістинг програм

***До розділу 2.4:***

```
@page "/fetchdata"
@using BlazorServerApp.Data
@inject WeatherForecastService ForecastService

<h1>Weather forecast</h1>

<p>This component demonstrates fetching data from a service.</p>

@if (forecasts == null)
{
    <p><em>Loading...</em></p>
}
else
{
    <table class="table">
        <thead>
            <tr>
                <th>Date</th>
                <th>Temp. (C)</th>
                <th>Temp. (F)</th>
                <th>Summary</th>
            </tr>
        </thead>
        <tbody>
            @foreach (var forecast in forecasts)
            {
                <tr>
                    <td>@forecast.Date.ToShortDateString()</td>
                    <td>@forecast.TemperatureC</td>
                    <td>@forecast.TemperatureF</td>
                    <td>@forecast.Summary</td>
                </tr>
            }
        </tbody>
    </table>
}

@code {
    private WeatherForecast[] forecasts;

    protected override async Task OnInitializedAsync()
    {
        forecasts = await ForecastService.GetForecastAsync(DateTime.Now);
    }
}
```

***До розділу 4.3:***

**ItemSearchComponent.razor (головна сторінка)**

```
@page "/"
@inject IItemSearchUC ItemSearch
<div class="container">
    <SearchBarComponent OnSearch="HandleSearch"></SearchBarComponent>
```

```
    <br />
    @if (products != null && products.Count() > 0)
    {
        int i = 0;
        <div class="row">
            @foreach (var product in products)
            {
                i++;
                if (i % 6 != 0)
                {
                    <ProductCardComponent
Product="product"></ProductCardComponent>
                }
                else
                {
                    @*<div class="w-100"></div>*@
                }
            }
        </div>
    }
    else
    {
        <text>Вибачте, такого печива в нас немає.. :(</text>
    }
</div>
@code {
    private IEnumerable<Product> products;
    protected override void OnInitialized()
    {
        base.OnInitialized();
        products = ItemSearch.Execute();
    }
    private void HandleSearch(string filter)
    {
        products = ItemSearch.Execute(filter);
    }
}
```

**CartComponent.razor (кошик)**

```
@page "/cart"
@inject ICartStateStore CartStateStore
@inject IViewCartUC viewCartUC
<h3>Кошик замовлення</h3>
<br/>
@if (order != null && order.LineItems != null && order.LineItems.Count > 0)
{
    <div class="row">
        <div class="col-6">
            <ul>
                @*@foreach(var item in order.LineItems)*@
                @for (int i = 0; i < order.LineItems.Count; i++)
                {
                    <LineItemComponent
                                    LineItem="order.LineItems[i]"
                                    OnDeleteProduct="HandleDeleteProduct"

OnUpdateQuantity="HandleUpdateQuantity"></LineItemComponent>
                    @if (i < order.LineItems.Count - 1)
                    {
                        <li style="list-style:none">
```

```
                            <hr />
                        </li>
                    }
                }
            </ul>
        </div>
        <div class="col">
            <OrderReviewComponent Order="order"></OrderReviewComponent>
        </div>
    </div>
}
@code {
    private Order order;
    protected override async Task OnAfterRenderAsync(bool firstRender)
    {
        if (firstRender)
        {
            CartStateStore.AddStateChangeListeners(HandleCartStateChange);
            order = await viewCartUC.Execute();
            StateHasChanged();
        }
    }
    private void HandleDeleteProduct(Order order) => this.order = order;
    private void HandleUpdateQuantity(Order order) => this.order = order;
    private async void HandleCartStateChange()
    {
        order = await viewCartUC.Execute();
        StateHasChanged();
    }

}
```

**OrderReviewComponent.razor (деталі замовлення)**

```
@inject NavigationManager NavigationManager

    <div class="card m-2" style="width: 25rem;">
        <div class="card-body">
            <h5 class="card-title">Деталі замовлення</h5>
            <div class="card-text d-flex justify-content-between">
                <div>
                    Товари (@itemsCount):
                </div>
                <div>
                    @itemsTotalPrice.ToString("c")
                </div>
            </div>
            <div class="card-text d-flex justify-content-between">
                <div>
                    Доставка:
                </div>
                <div>
                    <text>0,00 ₴</text>
                </div>
            </div>
            @*<div class="card-text d-flex justify-content-between">
                <div>
                    Estimated Tax:
                </div>
                <div>
                    <text>$0.00</text>
```

```
                </div>
            </div>*@
            <hr />
            <div class="card-text d-flex justify-content-between"
style="color:darkred">
                <div>
                    <b>Сума замовлення:</b>
                </div>
                <div>
                    <b>@itemsTotalPrice.ToString("c")</b>
                </div>
            </div>
        </div>
        @if (!HideFinishOrderButton)
        {
            <button class="button btn-primary" @onclick="FinishOrder">Розмістити
замовлення</button>
        }
    </div>
@code {
    int itemsCount = 0;
    double itemsTotalPrice = 0;
    [Parameter]
    public Order Order { get; set; }
    [Parameter]
    public bool HideFinishOrderButton { get; set; } = false;
    protected override void OnParametersSet()
    {
        base.OnParametersSet();
        if (Order != null)
        {
            itemsCount = Order.LineItems.Count;
            itemsTotalPrice = 0;
            foreach (var item in Order.LineItems)
            {
                itemsTotalPrice += item.Price * item.Quantity;
            }
        }
    }
    private void FinishOrder()
    {
        NavigationManager.NavigateTo("/FinishOrder");
    }
}
```

**ConfOrderComponent.razor (підтвердження замовлення)**

```
@page "/orderconfirm/{uid}"
@inject IViewConfOrderUC viewConfOrderUC
<h3>Ваше замовлення успішно розміщено!</h3>
<br />
@if (order != null)
{
    <p>
        <b>Інформація щодо доставки:</b>
    </p>
    <p>@order.CustomerName</p>
    <p>@order.CustomerAddress</p>
    <p>@order.CustomerCity</p>
    <p>@order.CustomerStateProvince</p>
    <p>@order.CustomerCountry</p>
```

```
    <br />
    <OrderReviewComponent Order="order"
HideFinishOrderButton="true"></OrderReviewComponent>
    <br />
    <table class="table">
        <thead>
            <tr>
                <th>Товар</th>
                <th>Кількість</th>
                <th>Ціна</th>
            </tr>
        </thead>
        <tbody>
            @foreach (var item in order.LineItems)
            {
                <tr>
                    <td>@item.Product.Name</td>
                    <td>@item.Quantity</td>
                    <td>@item.Product.Price</td>
                </tr>
            }
        </tbody>
    </table>
}
```

**FinishOrderComponent.razor (розміщення замовлення)**

```
@page "/FinishOrder"
@inject NavigationManager NavigationManager
@inject IViewCartUC viewCartUC
@inject IFinishOrderUC FinishOrderUC
<MessageComponent @ref="msgComponent"></MessageComponent>
<h3>Розміщення замовлення</h3>
<br />

@if (order != null)
{
<div class="row">
    <div class="col">
        <ClientFormComponent
OnCustomerInfoSubmitted="HandleCustomerInfoSubmitted"></ClientFormComponent>
    </div>
    <div class="col">
        <OrderReviewComponent Order="order"
HideFinishOrderButton="true"></OrderReviewComponent>
    </div>
</div>
}
@code {
    private Order order;
    private MessageComponent msgComponent;
    protected override async Task OnAfterRenderAsync(bool firstRender)
    {
        if (firstRender)
        {
            order = await viewCartUC.Execute();
            StateHasChanged();
        }
    }
    private async void HandleCustomerInfoSubmitted(CustomerViewModel customer)
    {
```

```
        var mapper = new AutoMapper.MapperConfiguration(cfg =>
cfg.CreateMap<CustomerViewModel, Order>()).CreateMapper();
        mapper.Map<CustomerViewModel, Order>(customer, order);
        var orderUniqueId = await FinishOrderUC.Execute(order);
        if (string.IsNullOrWhiteSpace(orderUniqueId))
        {
            // error occured, not able to place order, please contact the admin
            msgComponent.ShowError("Information in the order is invalid, please
double check.");
        }
        else
        {
            NavigationManager.NavigateTo($"/orderconfirm/{orderUniqueId}");
        }
    }

}
@code {
    private Order order;
    [Parameter]
    public string uId { get; set; }
    protected override void OnParametersSet()
    {
        base.OnParametersSet();
        if (!string.IsNullOrWhiteSpace(uId))
        {
            order = viewConfOrderUC.Execute(uId);
        }
    }
}
```

**ClientFormComponent.razor (форма для заповнення адреси доставки)**

```
@if (customer != null)
{
    <EditForm Model="customer" class="form-line"
OnValidSubmit="HandleValidSubmit">
        <DataAnnotationsValidator></DataAnnotationsValidator>
        <ValidationSummary></ValidationSummary>
        <div class="form-group">
            <label for="name">ПІБ (повне)</label>
            <InputText id="name" @bind-Value="customer.CustomerName"
class="form-control"></InputText>
        </div>
        <div class="form-group">
            <label for="address">Адреса</label>
            <InputText id="address" @bind-Value="customer.CustomerAddress"
class="form-control"></InputText>
        </div>
        <div class="form-group">
            <label for="city">Місто</label>
            <InputText id="city" @bind-Value="customer.CustomerCity"
class="form-control"></InputText>
        </div>
        <div class="form-group">
            <label for="province">Область</label>
            <InputText id="province" @bind-
Value="customer.CustomerStateProvince" class="form-control"></InputText>
        </div>
        <div class="form-group">
            <label for="country">Країна</label>
```

```
            <InputText id="country" @bind-Value="customer.CustomerCountry"
class="form-control"></InputText>
        </div>
        <button type="submit" class="btn btn-primary">Розмістити
замовлення</button>
    </EditForm>
}
@code {
    private CustomerViewModel customer;

    [Parameter]
    public EventCallback<CustomerViewModel> OnCustomerInfoSubmitted { get; set;
}
    protected override void OnInitialized()
    {
        base.OnInitialized();
        customer = new CustomerViewModel();
    }
    private void HandleValidSubmit()
    {
        OnCustomerInfoSubmitted.InvokeAsync(customer);
    }
}
```

**OutstandOrdersComponent.razor (невиконані замовлення – адмін)**

```
@page "/OutstandOrders"
@page "/admin"
@attribute [Authorize]
@inject IViewOutstandingOrderUC viewOutstandingOrderUC
<h3>Невиконані замовлення</h3>
<br/>
<OrderListComponent Orders="orders"></OrderListComponent>
@code {
    private IEnumerable<Order> orders;
    protected override void OnInitialized()
    {
        base.OnInitialized();
        orders = viewOutstandingOrderUC.Execute();
    }
}
```

**ProcOrdersComponent.razor (опрацьовані замовлення)**

```
@page "/ProcOrders"
@inject IViewProcOrdersUC viewProcOrdersUC
@attribute [Authorize]
<h3>Опрацьовані замовлення</h3>
<br />
<OrderListComponent Orders="orders"></OrderListComponent>
@code {
    private IEnumerable<Order> orders;
    protected override void OnInitialized()
    {
        base.OnInitialized();
        orders = viewProcOrdersUC.Execute();
    }
}
```