МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Острозька академія»
Економічний факультет
Кафедра економіко-математичного моделювання та інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня бакалавра

на тему: «**Розробка системи для обліку програмного та апаратного забезпечення в компанії»**

Виконав: студент 4 курсу, групи КН-42
першого (бакалаврського) рівня вищої освіти
спеціальності 122 Комп'ютерні науки
освітньо-професійної програми «Комп'ютерні науки»
Поліщук Юрій Вадимович

Керівник: ***Клебан Ю.В., старший викладач кафедри ЕММІТ***

Рецензент: *кандидат технічних наук, доцент, доцент кафедри прикладної математики та кібербезпеки Донецького національного університету імені Василя Стуса*
Загоруйко Любов Василівна

РОБОТА ДОПУЩЕНА ДО ЗАХИСТУ

Завідувач кафедри економіко-математичного моделювання та інформаційних технологій _____ (проф., д.е.н. Кривицька О.Р.)

Протокол № 11 від «30» травня 2024 р.

Острог, 2024

Міністерство освіти і науки України
Національний університет «Острозька академія»

Факультет: *економічний*

Кафедра: *економіко-математичного моделювання та інформаційних технологій*

Спеціальність: 122 Комп'ютерні науки
Освітньо-професійна програма: Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Ольга КРИВИЦЬКА
«____» _____20__ р.

ЗАВДАННЯ
на кваліфікаційну роботу студента

Поліщука Юрія Вадимовича

1. Тема роботи: **Розробка системи для обліку програмного та апаратного забезпечення в компанії**.

Керівник роботи: Клебан Ю.В., старший викладач кафедри ЕММІТ

Затверджено наказом ректора НаУОА від 03.11.2023 р., № 98.

2. Термін здачі студентом закінченої роботи: 31 травня 2024 року.

3. Вихідні дані до роботи: Figma, Visual Studio, SSMS, Google Documents.

4. Перелік завдань, які належить виконати: проаналізувати ринок інвентаризаційних систем управління; визначити основні цілі використання систем управління користувачів; розробити серверну частину веб-застосунку; знайти дизайн який буде інтегрований; інтегрувати дизайн в клієнтську частину веб-застосунку; створити власний логотип проєкту;

5. Перелік графічного матеріалу: результати досліджень та розроблений веб-застосунок.

6. Консультанти розділів роботи:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Клебан Ю.В.	01.12.2023	01.12.2023
2	Клебан Ю.В.	01.12.2023	01.12.2023
3	Клебан Ю.В.	01.12.2023	01.12.2023

7. Дата видачі завдання: 01.12.2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів	Примітка
1	Затвердження теми роботи	до 31.10.2023 р.	
2	Постановка технічного завдання	до 01.12.2023 р.	
3	Ознайомлення з документацією	до 14.12.2023 р.	
4	Написання розділу 1	до 28.01.2024 р.	
5	Написання розділу 2	до 10.03.2024 р.	
6	Написання розділу 3	до 20.04.2024 р.	
7	Тестування серверної та клієнтської частини	до 01.05.2024 р.	
8	Виправлення помилок	до 16.05.2024 р.	
9	Попередній захист та перевірка на рівень унікальності кваліфікаційної роботи	до 31.05.2024 р.	
10	Здача кваліфікаційної роботи на кафедру	31.05.2024 р.	

Студент: _____ Юрій ПОЛІЩУК

Керівник кваліфікаційної роботи: _____ Юрій КЛЕБАН

АНОТАЦІЯ
кваліфікаційної роботи
на здобуття освітнього ступеня бакалавра

Тема: Розробка системи для обліку програмного та апаратного забезпечення в компанії
Автор: *Поліщук Юрій Вадимович*
Науковий керівник: Клебан Ю.В., старший викладач кафедри ЕММІТ

Захищена «……»…… 2024 року.

Пояснювальна записка до кваліфікаційної роботи: *60 с., 31 рис., 1 табл., 3 додатки, 28 джерел.*
Ключові слова: *розробка застосунку, аналіз конкурентів, пошук дизайну, Figma, ASP.NET, Visual Studio.*
Короткий зміст праці:
Ця кваліфікаційна робота досліджує сферу систем управління товарами, або запасами, зосереджене на аналізі конкурентів та пошук найкращих рішень для створення власного веб-застосунку. Проєкт описує пошук найкращих рішень опираючись на користувацький досвід в межах певної компанії. Окрім пошуку рішень та аналізу конкурентів, було розроблено веб-застосунок на платформі ASP.NET, де розроблено клієнтська та серверна частина. А також була пророблена робота пошуку дизайну та робота в Figma. Як результат, ми отримали досвід розробки власного проєкту та аналізу конкурентів за для найзручнішого користування вже освічених користувачів в цій сфері та легкого початку користування застосунком для нових користувачів. Під час розробки було використано програми: Visual Studio, SSMS, Figma.

_____

ANNOTATION
of qualification paper
for bachelor's degree

Theme: *Development of a system for accounting for software and hardware in the company*
Author: *Yurii Polishchuk*
Scientific supervisor: *Kleban Yu., senior lecturer of the EMMIT department*

Defended «……»…… of 2024.

Explanatory note to the qualification work: *60 p., 31 pic., 1 tables., 3 attachments, 28 sources.*
Keywords: *application development, competitor analysis, design search, Figma, ASP.NET, Visual Studio.*
Summary of the paper:
This qualification explores the field of inventory management systems, focusing on competitor analysis and finding the best solutions for creating your own web application. The project describes the search for the best solutions based on user experience within a specific company. free search for solutions and analysis of competitors, a web application was developed on the ASP.NET platform, where the client and server parts were developed. And a job search for design and work at Figma was also developed. As a result, we gained experience in developing our own project and analyzing competitors for the most convenient use of already educated users in this area and easy start of using the application for new users. During development, the following programs were used: Visual Studio, SSMS, Figma.

_____

ВСТУП

Актуальність теми: У сучасному світі підприємства постійно зіштовхуються з необхідністю оптимізації ресурсів та ефективної роботи з обладнанням і програмним забезпеченням. Комплексна автоматизація управління активами є ключовим чинником підвищення продуктивності та конкурентоспроможності. Розробка інтегрованої системи обліку дозволяє точно контролювати всі активи компанії, планувати та здійснювати технічне обслуговування, тим самим знижуючи витрати та підвищуючи ефективність роботи.

Фірмовий стиль і інтеграція: Розробка такої системи вимагає високого рівня технічної компетентності та креативу в дизайні інтерфейсу, що забезпечить легкість використання та високу адаптивність для користувачів різних рівнів. Це також включає створення єдиної, інтегрованої структури, яка відповідає корпоративним стандартам та бренду компанії.

Мета дослідження: Основною метою дослідження є розробка веб-застосунку на Blazor WebAssembly для обліку програмного та апаратного забезпечення, який би інтегрувався з існуючою IT-інфраструктурою компанії. Це включає в себе дизайн користувацького інтерфейсу, розробку backend системи та їх інтеграцію для забезпечення максимальної ефективності та взаємодії між відділами.

Об'єкт дослідження: Процес розробки і впровадження комплексної системи обліку активів на базі новітніх технологій.

Предмет дослідження: Вплив впровадження інтегрованої системи обліку на ефективність управління активами компанії.

Методи дослідження:

- Теоретичний аналіз сучасних методів розробки програмного забезпечення.
- Практична розробка веб-застосунку.
- Тестування і аналіз впровадження системи в реальних умовах роботи компанії.

Переваги використання інформаційних технологій:

Впровадження автоматизованої системи дозволяє суттєво знизити людські помилки, швидко оцінювати стан активів і планувати їх обслуговування, що призводить до зменшення витрат і підвищення доходів компанії.

Досягнення цілей: Розробка та впровадження системи не тільки покращує процеси обліку та управління активами, але й забезпечує компанії важливу стратегічну перевагу на ринку завдяки оптимізації внутрішніх ресурсів і покращенню якості обслуговування клієнтів

РОЗДІЛ 1

ТЕОРЕТИЧНІ АСПЕКТИ ІНВЕНТАРИЗАЦІЙНИХ СИСТЕМ В МЕЖАХ КОМПАНІЙ

1.1. Постановка проблеми, опис базових понять бізнес-процесу

У межах сучасної компанії існує потреба в системі обліку системного та апаратного забезпечення, яка є важливою для оптимізації процесів і забезпечення ефективності управління інвентаризацією. Однак існуючі інструменти часто не забезпечують зручного та простого для розуміння інтерфейсу, що ускладнює роботу працівників, відповідальних за ці ключові ресурси компанії.

Опис базових понять бізнес-процесу:

Для забезпечення ефективності управління інвентаризацією в межах компанії було розроблено веб-застосунок на платформі ASP.NET. Основна мета проекту полягає в полегшенні процесу інвентаризації товарів та їх переміщення в межах компанії.

Застосунок спрямований на створення зручного інтерфейсу для працівників, що відповідають за системне та апаратне забезпечення. Він пропонує ефективний і простий у використанні інструмент для ідентифікації, обліку та моніторингу руху ресурсів.

У цьому контексті система повинна забезпечувати:

1. Інвентаризацію- це можливість точного обліку всього інвентару в компанії, включаючи системне та апаратне забезпечення.
2. Переміщення ресурсів- це функціонал для зручного та швидкого переміщення інвентарю в межах підприємства, з можливістю розуміння який товар, з яким інвентаризаційним номерком видавався і для кого.

3. Ефективний інтерфейс- це інтуїтивний і зрозумілий інтерфейс для працівників, що дозволяє легко взаємодіяти з системою та швидко виконувати потрібні операції без великого навантаження на навчання.

Зважаючи на те, що часто компанії змушені використовувати програми інших розробників чи компаній, це не є зручним рішенням для всіх, часто є незрозумілим та заплутаним. Особливо в умовах коли немає людини яка мала досвід роботи, або хоча б мала б посібник для вивчинення певного продукту, який рідко буває в IMS системах. Та й використання чужих систем безпечним і надійним рішенням не назвеш.

Тож інвентаризаційні системи відіграють критично важливу роль у бізнес-операціях, особливо в контексті управління запасами та оптимізації логістики. Вони не тільки забезпечують точність даних про товарно-матеріальні цінності але й впливають на швидкість та ефективність виконання замовлень та загальну продуктивність. В сучасному глобалізованому світі, де рівень конкуренції постійно зростає, компанії вважають важливим впроваджувати продвинуті інвентаризаційні системи для забезпечення конкурентноспроможності.

Ефективна інвентаризаційна система дозволяє компанії значно скоротити витрати, оптимізувати складські запаси та забезпечити високу точність звітності. Крім того, такі системи сприяють кращому плануванню та прогнозуванню, що є важливим для підтримки безперервності бізнесу та уникнення дефіциту або надлишку товарів. Важливою є інтеграція з іншими інформаційними системами підприємства, такими як системи фінансового обліку, CRM та ERP системи.

Стикаючись щодня на роботі з проблемами однієї з програм, було прийняте ріше розпочати власний застосунок.

Основні проблеми які сторонні застосунки:

- Відсутність посібника з інструкцією по експлоатації
- Важкий процес отримання аккаунту користувача
- Можливість витоку даних
- Відсутність можливості контролювати процесом ієрархій в межах компанії
- Відсутність можливості редагування системи під потреби користувача

- Відсутність техпідтримки

Переваги конкурентів:

- Наявність великої команди
- Наявність додаткових API які вже інтегрованні в застосунок
- Програми у вигляді окремого застосунку

Ці та інші виклики потребують ретельного вивчення та грамотного підходу до розробки інвентаризаційних систем, щоб максимально використовувати потенціал сучасних технологій та забезпечити їх відповідність корпоративним потребам та стандартам.

Також варто звернути увагу на те, що наша продукт буде реалізованим як вебзастосунок, на відміну від більшості конкурентів

Переваги веб-застосунків

1. **Доступність.** Веб-застосунки можна використовувати з будь-якого комп'ютера, телефону, ноутбуку, планшету, тощо, з доступом до інтернету. Це забезпечує більшу гнучкість та мобільність.
2. **Оновлення**. Оновлення веб-застосунків виконуються на сервері, тому користувачі автоматично мають доступ до останніх версій без необхідності встановлювати оновлення.
3. **Сумісність.** Веб-застосунки працюють у браузерах, що знімає проблеми сумісності з різними операційними системами.

Недоліки веб-застосунків

1. **Залежність від інтернет.** Веб-застосунки потребують стабільного інтернет-з'єднання для роботи, що може бути обмеженням у віддалених або малодоступних місцях.
2. **Безпека.** Веб-застосунки частіше схильні до атак, таких як XSS або CSRF, тому потрібно особливо дбати про захист даних.
3. **Швидкість та реакційність.** Вони можуть працювати повільніше, ніж настільні застосунки, особливо якщо обробляють великі об'єми даних.

4. **Обмежена функціональність.** Веб-застосунки можуть мати обмеження у функціональності порівняно з настільними застосунками, особливо у взаємодії з апаратним забезпеченням.

Переваги desktop application

1. **Працюють офлайн.** Desktop apps можуть працювати без інтернет-з'єднання.

2. **Швидкість та реакційність.** Вони зазвичай швидші і мають кращу реакційність, оскільки ресурси комп'ютера використовуються ефективніше.

3. **Безпечніше зберігання даних.** Часто безпечніше зберігати чутливі дані локально на комп'ютері, ніж на сервері в інтернеті.

4. **Повний доступ до апаратних ресурсів.** Desktop apps можуть безпосередньо взаємодіяти з апаратним забезпеченням комп'ютера, що забезпечує більшу потужність та можливості.

Недоліки desktop apps

1. **Обмежена доступність.** Вони обмежені конкретним комп'ютером, на якому встановлені.

2. **Витрати часу на оновлення та сумісність**. Оновлення потрібно встановлювати окремо на кожному комп'ютері, що може викликати проблеми з сумісністю.

Хочеться одразу проговорити всі за та проти цих варіантів на прикладах. Розпочнемо з залежностей. Перша проблема, яка стосується обох. Першими є вебзастосунки які повинні мати підключення до інтернету. Тобто в умовах відключення світла, мобільних операторів, проблем з мережевим обладнанням, тощо, ми не зможемо мати доступу до застосунку. Але давайте розглянемо звичайні десктопні застосунки які хоч і не потребують для запуску підключення інтернету, але так само працюють на запитах до баз даних які просят підключення. Але крім того вони ще вимагають встановлення та оновлення, без яких ви не зможете працювати, хоча й можуть швидше обробляти запити

1.2. Аналіз продуктів-конкурентів на ринку

У сфері систем обліку інвентаризації існує кілька продуктів-конкурентів, які пропонують різноманітні рішення для управління інвентарем та переміщенням

ресурсів в межах компаній. Один з найбільш відомих конкурентів у цій галузі є SAP, зокрема, їхнє рішення SAP Logon.

Так як в моїй компанії використовують SAP LOGON + Microsoft Excel, я вирішив поєднати ці прогррами в одне ришення, щоб мати одне середовище для всіх і було зрозумілим і зручним.

SAP Logon - це клієнтська програма, що дозволяє користувачам підключатися до різних систем SAP. Вона надає можливість управління і доступу до різних модулів SAP ERP, включаючи модулі для управління запасами та інвентаризацією. Це потужне рішення, яке дозволяє ефективно вести облік ресурсів, виконувати операції переміщення та взаємодіяти з базами даних компаній.

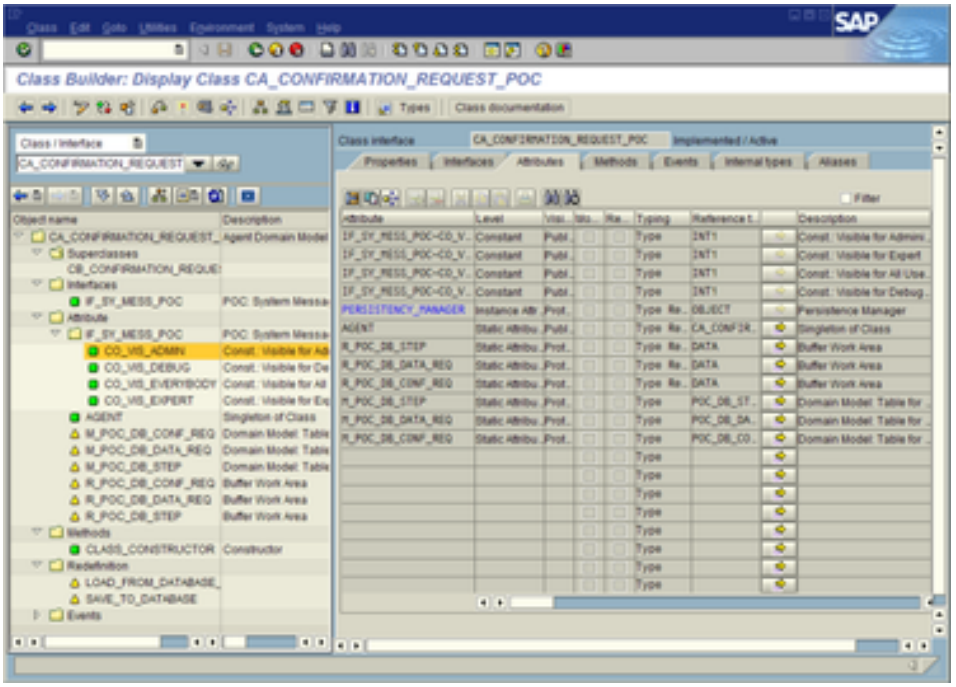

Рис. 1.1. інтерфейс SAP Logon

Джерело: [8]

Переваги SAP:

1. SAP є дуже функціональною системою, яка інтегрує всі бізнес-процеси в єдину систему, забезпечуючи цілісний підхід до управління підприємством.

2. SAP підтримує різні валюти, мови та стандарти звітності, роблячи її ідеальним вибором для міжнародних компаній.

3. SAP постійно інвестує в нові технології, такі як штучний інтелект, машинне навчання та Інтернет речей, для покращення своїх продуктів і послуг.

4. SAP ефективно масштабується для задоволення потреб різних бізнесів, від малих до великих корпорацій.

5. Велика користувацька база та спільнота SAP забезпечують значні ресурси для навчання, підтримки та співпраці.

Недоліки SAP:

1. Вартість впровадження і підтримки SAP може бути дуже високою, особливо для малого та середнього бізнесу.

2. Впровадження SAP може бути тривалим і складним процесом, який вимагає значних ресурсів і змін у бізнес-процесах.

3. SAP має складну конфігурацію, і для ефективного використання системи часто потрібні консультанти та спеціалізовані тренінги.

4. Деякі користувачі відзначають, що система може бути негнучкою у плані адаптації до специфічних потреб підприємства без складних налаштувань і модифікацій.

5. Як і в інших хмарних системах, стабільний доступ до інтернету є критично важливим для доступу до SAP.

SAP є потужним інструментом для управління підприємством, але вибір цієї системи має бути ретельно зваженим, з урахуванням фінансових та оперативних можливостей компанії.

Окрім SAP Logon, існують інші конкуруючі продукти, такі як:

1.Oracle NetSuite: Ця система ERP також має модулі для обліку запасів та інвентаризації. Вона пропонує рішення для точного обліку товарів, взаємодії з клієнтами та управлінням операціями у поєднанні з іншими бізнес-процесами.

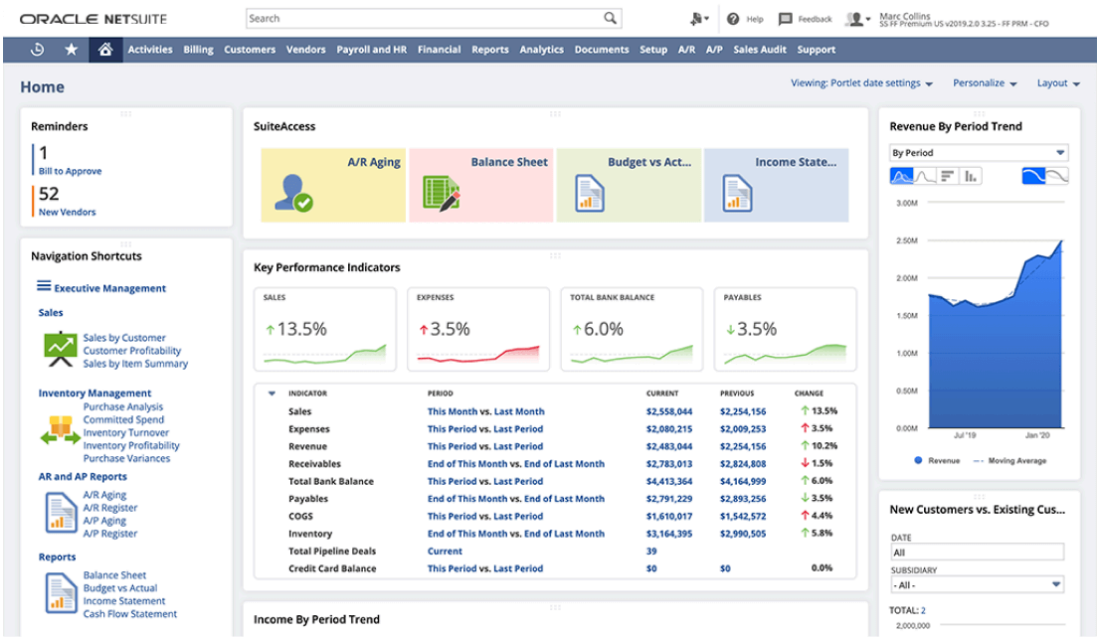

Рис. 1.2 інтерфейс Oracle NetSuite

Джерело: [9]

Переваги Oracle NetSuite:

1. NetSuite пропонує єдину інтегровану платформу для управління всіма основними аспектами бізнесу, включаючи фінанси, CRM, запаси і електронну комерцію, що усуває потребу в множинні окремих систем.

2. Як повністю хмарне рішення, NetSuite забезпечує високу доступність та зменшує витрати на IT-інфраструктуру, оскільки немає необхідності в покупці та обслуговуванні серверного обладнання.

3. NetSuite підходить як для малих, так і для великих компаній з можливістю масштабування в міру росту бізнесу.

4. Система забезпечує доступ до даних у реальному часі, що дозволяє керівництву швидко реагувати на зміни в бізнесі та ринкові умови.

5. NetSuite підтримує різні валюти, мови та вимоги до звітності, що робить його ідеальним рішенням для міжнародних компаній.

Недоліки Oracle NetSuite:

1. Як і багато інших ERP-систем, NetSuite може бути дорогим, особливо для малого бізнесу, з урахуванням витрат на ліцензування, налаштування та обслуговування.

2. Впровадження та налаштування NetSuite можуть вимагати значних зусиль і витрат часу, особливо для компаній зі складними процесами та індивідуальними потребами.

3. Через свою комплексність і широкий спектр функціональностей, користувачам може знадобитися значний час, щоб повністю освоїти систему.

4. Як хмарне рішення, NetSuite вимагає стабільного інтернет-з'єднання для доступу до системи, що може бути проблемою в районах з обмеженим доступом до швидкісного інтернету.

5. Хоча NetSuite є дуже гнучким, деякі користувачі вказують на обмеження у налаштуванні індивідуальних потреб без додаткових витрат.

Oracle NetSuite залишається одним з найбільш популярних хмарних ERP-рішень на ринку, пропонуючи компаніям могутній інструмент для управління різними аспектами їхнього бізнесу з одного додатка. Проте, потенційні користувачі мають уважно оцінити вартість і складність впровадження NetSuite у свої бізнес-операції.

2. Microsoft Dynamics 365: Ця платформа також має модулі для управління запасами, що дозволяє користувачам вести облік та взаємодіяти з інвентарем.

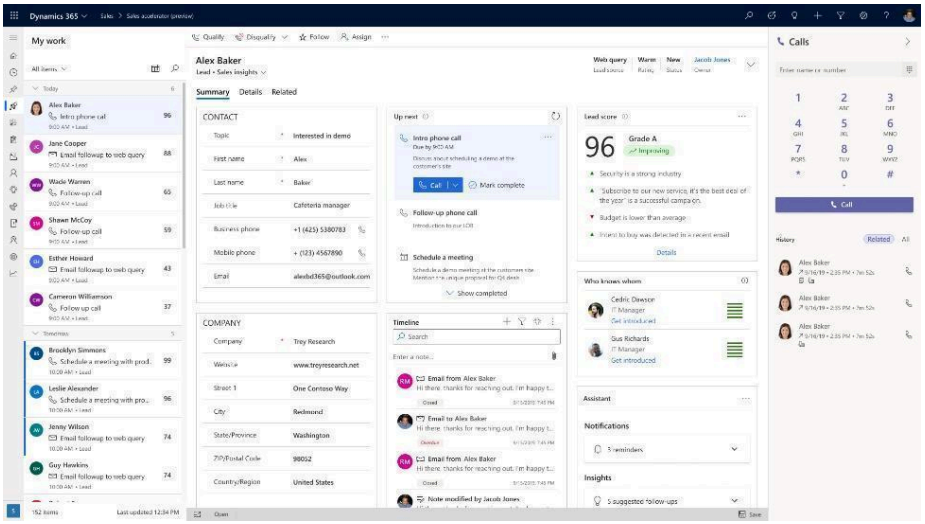

Рис. 1.3 інтерфейс Microsoft Dynamics 365

Джерело: [10]

Переваги Microsoft Dynamics 365:

1. Однією з ключових переваг є її тісна інтеграція з іншими продуктами Microsoft, що дозволяє користувачам легко синхронізувати дані та процеси з такими додатками, як Microsoft Office та Microsoft Teams.

2. Dynamics 365 легко масштабується, щоб відповідати потребам як малих, так і великих компаній, з можливістю легко додавати або видаляти модулі в залежності від потреб бізнесу.

3. Компанії можуть вибирати між хмарним розгортанням, на місці або гібридним підходом, що дозволяє їм оптимізувати свої IT-структури і витрати.

4. Від фінансів і операцій до збуту і обслуговування клієнтів, Dynamics 365 надає комплексні можливості для управління всіма аспектами бізнесу.

5. Microsoft регулярно оновлює Dynamics 365, включаючи новітні технологічні досягнення, такі як штучний інтелект і машинне навчання, що забезпечує клієнтам передові рішення.

Недоліки Microsoft Dynamics 365:

1. Як і багато інших корпоративних рішень, Dynamics 365 може бути досить дорогим, особливо з урахуванням ліцензування, налаштування і підтримки.

2. Через свою комплексність Dynamics 365 може вимагати значних зусиль для налаштування та управління, особливо для компаній без значного IT-персоналу.

3. Впровадження може займати значний час, особливо в великих організаціях з складними процесами.

4. Хмарна версія вимагає надійного інтернет-з'єднання для доступу до всіх функцій, що може бути обмеженням у деяких регіонах.

5. Завдяки своїм широким можливостям і комплексності, для користувачів може знадобитися значний час, щоб повністю освоїти всі аспекти системи.

Microsoft Dynamics 365 залишається потужним рішенням для бізнесу, яке може значно покращити ефективність управління компанією, але потенційні користувачі мають враховувати його вартість та складність при виборі рішення для своїх бізнес-потреб.

3. Zoho Inventory: Це онлайн-рішення для управління запасами, яке надає інструменти для обліку товарів, відстеження їх руху та управління замовленнями.

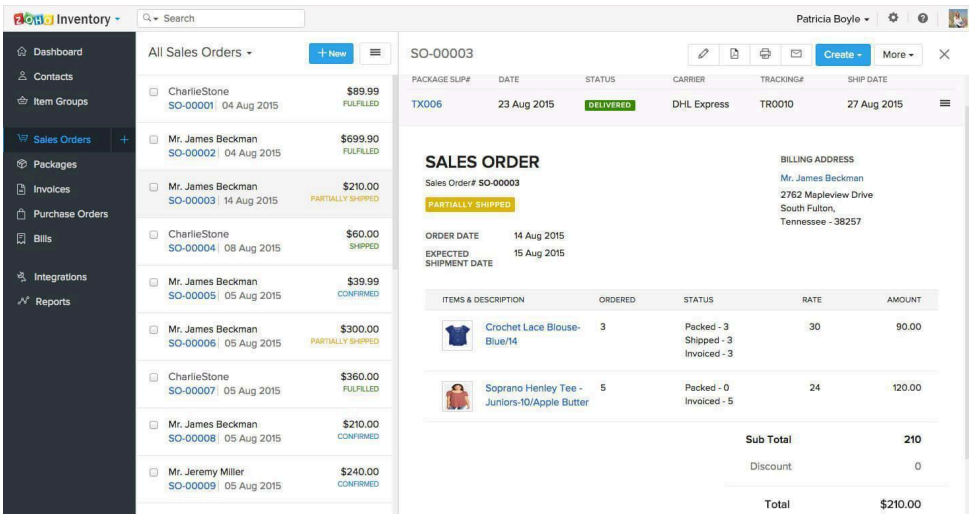

Рис. 1.4 інтерфейс Zoho Inventory

Джерело: [11]

Переваги Zoho Inventory:

1. Zoho Inventory дозволяє автоматизувати багато аспектів управління запасами, включаючи відстеження запасів, управління замовленнями та автоматизоване поповнення запасів.

2. Як частина екосистеми Zoho, Inventory легко інтегрується з іншими програмами Zoho, такими як Zoho Books (бухгалтерський облік) та Zoho CRM, що створює єдину платформу для управління бізнесом.

3. З можливістю використовувати різні валюти, Zoho Inventory є ідеальним для компаній, що ведуть бізнес на міжнародному рівні.

4. Завдяки зрозумілому інтерфейсу та багатьом наявним навчальним матеріалам, Zoho Inventory є доступним для користувачів без глибоких технічних знань.

5. Мобільний додаток дозволяє керувати запасами та продажами з будь-якого місця, що важливо для динамічного бізнесу.

Недоліки Zoho Inventory:

1. Безкоштовна версія має значні обмеження за кількістю транзакцій та користувачів, що може бути недостатньо для розширення бізнесу.

2. Як і багато інших хмарних рішень, Zoho Inventory вимагає стабільного інтернет-з'єднання для роботи, що може стати проблемою в районах з поганим покриттям.

3. Хоча інтерфейс і є зрозумілим, деякі користувачі відзначають складність налаштувань та конфігурації додаткових функцій.

4. Zoho Inventory може не включати деякі складні функції управління ланцюгами постачання, які потрібні великим організаціям.

Zoho Inventory вигідно відрізняється своєю багатофункціональністю та гнучкістю, однак потенційним користувачам варто розглянути специфіку їхнього бізнесу, щоб визначити, чи відповідає цей інструмент їхнім потребам.

4. QuickBooks: Ця платформа володіє також функціоналом для контролю за запасами та управлінням інвентарем для підприємств різного масштабу.

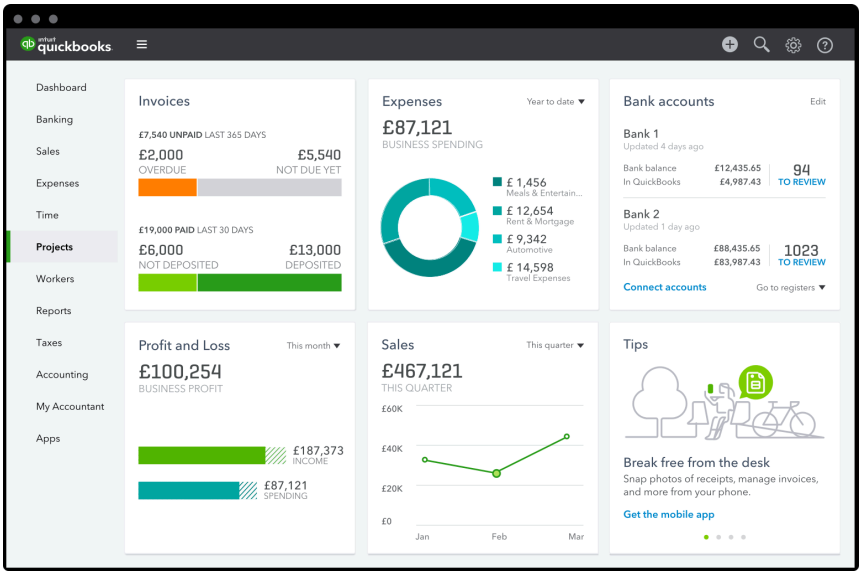

Рис. 1.5 інтерфейс QuickBooks

Джерело: [12]

Переваги QuickBooks:

1. QuickBooks зазвичай похвалився за свій зрозумілий і простий в користуванні інтерфейс, що робить його доступним навіть для не бухгалтерів.

2. Програма пропонує все необхідне для повного циклу бухгалтерського обліку, включаючи ведення рахунків, облік розрахунків, управління запасами, а також підготовку звітності.

3. QuickBooks може автоматизувати повторювані завдання, такі як виставлення рахунків, нарахування амортизації та інші, що зменшує час на ведення обліку.

4. QuickBooks легко інтегрується з іншими програмами, що дозволяє розширити його функціональність і забезпечити більшу ефективність роботи.

5. Є мобільні додатки, які дозволяють користувачам переглядати фінансові дані і управляти ними з будь-якого місця.

Недоліки QuickBooks:

1. Для деяких малих підприємств може бути дорого підтримувати ліцензію на QuickBooks, особливо на розширені пакети.

2. Базові версії QuickBooks можуть не мати всіх необхідних функцій для більших або більш складних бізнесів.

3. QuickBooks може не впоратися з великою кількістю транзакцій або дуже великими компаніями, що вимагають більш складних систем ERP.

4. Онлайн-версія QuickBooks залежить від стабільного інтернет-з'єднання, що може бути проблемою в районах з поганим покриттям.

5. Хоча QuickBooks пропонує заходи забезпечення безпеки, загальна залежність від хмарних рішень збільшує ризики пов'язані з витоком або втратою даних.

Вибір між QuickBooks та іншими системами залежить від специфіки бізнесу, бюджету, та потреб в обліку. QuickBooks залишається одним з найпопулярніших рішень для малого та середнього бізнесу завдяки його універсальності та зручності. Програма 1С є популярним рішенням для управління інвентаризацією в компаніях різних розмірів, особливо в країнах пострадянського простору.

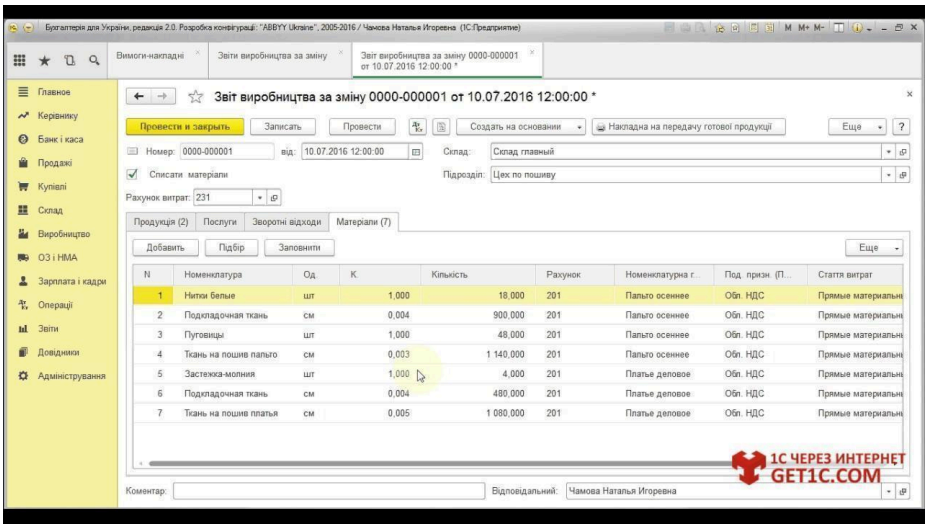

Рис. 1.6 інтерфейс 1С

Джерело: [16]

Переваги 1С в сфері інвентаризації:

1. 1С дуже добре інтегрована з іншими модулями бухгалтерського та податкового обліку, що забезпечує цілісність даних.
2. Програма має версії, повністю адаптовані під законодавство та бухгалтерські стандарти країн СНД, що є важливим для правильного обліку та звітності.
3. 1С дозволяє детально налаштовувати параметри обліку інвентарю залежно від специфіки діяльності підприємства.
4. Інтерфейс програми часто сприймається як більш зрозумілий і простий для користувачів, які вже знайомі з іншими продуктами 1С.

Недоліки 1С в сфері інвентаризації:

1. Для дуже великих компаній з великою кількістю транзакцій 1С може бути не так ефективна, як більш глобальні системи типу SAP.
2. У країнах, де 1С не є стандартом, може бути важко знайти кваліфікованих фахівців для підтримки та налаштування системи.
3. 1С може не так швидко впроваджувати новітні технологічні рішення та оновлення, як міжнародні конкуренти.
4. Може виникнути складність інтеграції 1С з іншими міжнародними системами ERP, якщо це необхідно для міжнародної діяльності компанії.

Аналіз конкурентів показав, що на ринку додатків ринку присутня значна конкуренція. Конкуренти сформували потужний функціонал, але це не заважає перейняти необхідні кейси і 10 покращити або змінити їх. Нище на основі висновків та досліджень сформований SWOT-аналіз програмного продукту який описуємо

SWOT-аналіз програмного продукту

Сильні сторони	Слабкі сторони
● Простий інтерфейс ● Швидкий та доступний для змін та редагування внутрішний процес ●	● Конкуренти більш розвинені в даній сфері ● Відсутність достатньої бази даних для самостійного перекладу

Можливості	Загрози
● Залучення та розвиток додаткових API ● Багатогранність та постійний пошук нових властивостей	● Неготовність компаній пробувати щось нове ● Оренда серверу яка може бути ненадійною

Джерело: Створено Автором

Отже, ми проаналізували наших конкурентів і сформували SWOT-аналіз. Нашому додатку потрібно розробити всі потрібні функції, зробити дружній і зручний інтерфейс, щоб мати значну перевагу перед конкурентами.

Висновки до розділу 1

У контексті теоретичних аспектів інвентаризаційних систем в рамках компаній, глибоке розуміння бізнес-процесів та їх оптимізація через застосування сучасних технологічних рішень є ключовими для підвищення ефективності управління запасами. Розроблені на платформі ASP.NET веб-застосунки надають можливість значно спростити процеси інвентаризації, забезпечуючи точний облік інвентару та ефективне переміщення товарів в межах компанії. Ці інноваційні системи надають інтуїтивно зрозумілий інтерфейс, що дозволяє працівникам легко взаємодіяти з системою, що знижує час на навчання та підвищує загальну продуктивність роботи.

Використання веб-застосунків дозволяє компаніям мати стабільний доступ до даних з будь-якого пристрою з інтернет-з'єднанням, забезпечуючи гнучкість і мобільність в управлінні запасами. Однак, це також вносить певні ризики, такі як залежність від стабільності інтернет-з'єднання та потенційні загрози кібербезпеки, що вимагає від компаній додаткових зусиль для забезпечення захисту даних.

Порівняння з іншими продуктами, такими як SAP та Microsoft Dynamics 365, показує, що попри високу вартість та складність впровадження, ці системи пропонують комплексне управління бізнес-процесами, глобальну підтримку та

широкі можливості масштабування. Ці платформи інтегрують різні аспекти управління підприємством, що робить їх незамінними для великих міжнародних компаній, які шукають стабільність і надійність у своїх інвентаризаційних системах.

В контексті розробки власного веб-застосунку, важливо зосередитися на створенні користувацького інтерфейсу, який би відповідав конкретним потребам компанії, забезпечуючи легкість у використанні та можливість швидкого доступу до важливих функцій без необхідності довготривалого навчання. Також, розробка повинна включати стратегії захисту даних та забезпечення високого рівня безпеки, щоб протистояти можливим кіберзагрозам. Враховуючи виклики, з якими стикаються компанії при використанні сторонніх систем, такі як складнощі із отриманням технічної підтримки або обмеження у налаштуванні продукту під специфічні потреби, розробка власного рішення може стати значним кроком до оптимізації інвентаризаційних процесів і підвищення контролю над корпоративними ресурсами.

Загалом, ефективне управління інвентаризаційними системами в компанії має вирішальне значення для підтримки операційної ефективності і досягнення стратегічних бізнес-цілей, особливо в умовах глобалізації та зростаючої конкуренції. Підприємства повинні прагнути до впровадження технологічно продвинутих, гнучких та безпечних систем, які зможуть адаптуватися до мінливих ринкових умов та вимог, забезпечуючи стійкий розвиток і конкурентоспроможність на ринку.

РОЗДІЛ 2

ВИБІР ПРОГРАМ ДЛЯ РОЗРОБКИ СЕРВЕРНОЇ ТА КЛІЄНТСЬКОЇ ЧАСТИНИ

2.1. Порівняння різних програм для розробики клієнтської та серверної частини

Вибір програмного забезпечення для розробки серверної та клієнтської частин веб-додатку є ключовим кроком, який визначає ефективність розробки, можливості масштабування проекту та зручність подальшої підтримки і оновлень. В цьому контексті, Visual Studio та SQL Server Management Studio (SSMS),Visual Studio Code є трьома інструментами, які часто вибирають розробники для створення рішень, що включають як серверну, так і клієнтську логіку. Також за дизайн відповідатиме Figma

Visual Studio - це інтегроване середовище розробки (IDE) від Microsoft, яке підтримує множину мов програмування (C#, VB.NET, JavaScript тощо) та інструментів для розробки різних типів програмних продуктів, включаючи веб-додатки, мобільні застосунки, настільні програми та багато інших.

SQL Server Management Studio - це спеціалізований інструмент для управління базами даних Microsoft SQL Server. SSMS дозволяє адмініструвати базу даних, налаштовувати її параметри, виконувати скрипти SQL, а також забезпечує засоби для моніторингу та оптимізації роботи сервера баз даних.

Visual Studio Code - це легкий, але потужний редактор коду, який підтримує багато мов програмування і технологій. Розроблений Microsoft, він працює на Windows, macOS та Linux. VS Code став популярним серед розробників завдяки своїй швидкості, гнучкості, а також великій кількості розширень, що дозволяють налаштувати редактор під особливі потреби розробки.

Figma — це веб-додаток для дизайну та створення прототипів, який стрімко здобуває популярність завдяки своїй легкості у використанні та функціоналу для колективної роботи над проектами в режимі реального часу. Також є зручним рішенням для веб-розробників, адже має можливість перетворювати дизайн який розробляють на CSS код.

WebStorm – вважається потужним інтегрованим середовищем розробки (IDE) від JetBrains, призначеним спеціально для розробки веб-додатків. Цей інструмент підтримує кілька мов програмування, таких як JavaScript, CSS, HTML, і надає розширену підтримку для сучасних фреймворків і бібліотек. WebStorm включає в себе вдосконалені функції, такі як автодоповнення, відладка, рефакторинг коду та підтримка різноманітних систем контролю версій, що робить його ідеальним інструментом для розробників веб-додатків.

Rider — це кросплатформене інтегроване середовище розробки (IDE) від компанії JetBrains, яке призначене для розробки на мові програмування C#. Це IDE підтримує розробку на .NET Framework, .NET Core та Mono, забезпечуючи можливість роботи на операційних системах Windows, macOS та Linux.

Visual Studio вирізняється серед інших інструментів розробки, особливо коли йдеться про роботу з Blazor WebAssembly, завдяки своїй глибокій інтеграції з екосистемою Microsoft. Це інтегроване середовище розробки оптимізовано для використання всіх переваг .NET і C#, що є основою Blazor. Робота в Visual Studio з проектами Blazor WebAssembly підкріплена могутнім набором інструментів для відлагодження, які дозволяють відстежувати і коригувати код у реальному часі прямо в браузері. Це забезпечує високий рівень продуктивності і значно спрощує процес виявлення та усунення помилок.

Крім того, Visual Studio підтримує розгортання та тестування додатків Blazor безпосередньо на різних серверах і платформах, забезпечуючи гнучкість і швидкість при перевірці та впровадженні проектів. Єдина екосистема з інструментами, як Visual Studio Code, Azure DevOps і GitHub, інтегрованими безпосередньо в IDE, дозволяє розробникам легко координувати свою роботу в команді, підтримуючи високий рівень синхронізації і співпраці.

Також Visual Studio автоматично забезпечує оновлення всіх необхідних бібліотек та фреймворків, забезпечуючи розробникам доступ до останніх інструментів та технологій, що постійно оновлюються. Це дозволяє зосередитися на розробці функціоналу без потреби постійно стежити за оновленнями інструментарію. Visual Studio надає широкі можливості для налаштування робочого середовища, що дозволяє кожному розробнику створити ідеальне середовище під свої завдання і стиль роботи.

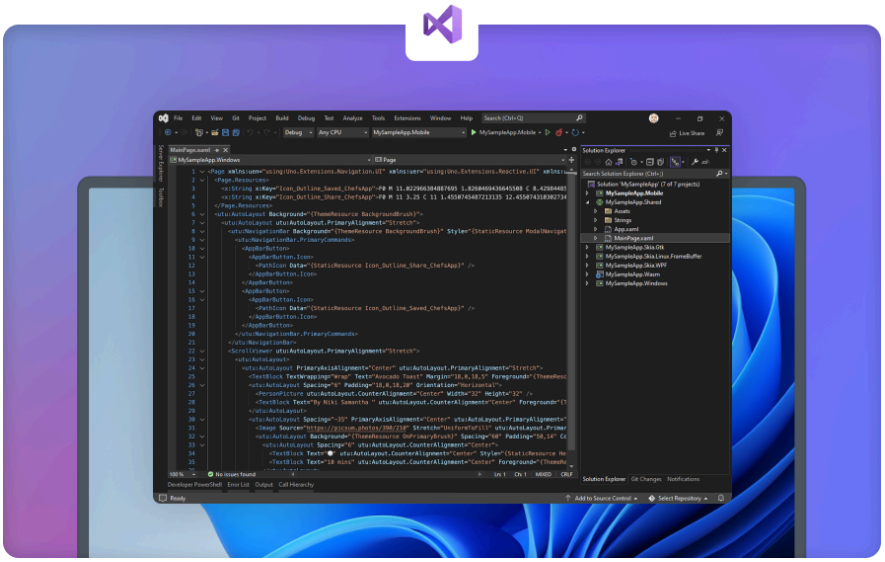

Рис. 2.1 Вигляд програми Visual Studio

Джерело: [17]

SQL Server Management Studio (SSMS) є незамінним інструментом для розробників, які використовують Microsoft SQL Server у своїх проектах, особливо коли вони працюють у зв'язці з Visual Studio. Однією з ключових переваг SSMS є його глибока інтеграція з Visual Studio, що створює єдине нерозривне середовище для розробки додатків. Це інтегроване середовище значно спрощує процеси управління базами даних, від моніторингу та оптимізації до налагодження та управління змінами.

Коли розробники використовують SSMS разом із Visual Studio, вони отримують доступ до потужних інструментів SQL, які покращують робочі процеси, включаючи графічне створення та аналіз запитів, яке значно збільшує швидкість розробки. Ця інтеграція дозволяє розробникам виконувати складні SQL-операції

безпосередньо з Visual Studio, що зменшує необхідність перемикання між різними інструментами і забезпечує більш плавний процес розробки.

Також SSMS підтримує автоматизацію багатьох задач адміністрування баз даних, таких як резервне копіювання та відновлення, що є надзвичайно корисним для забезпечення цілісності даних та відновлення після збоїв. Ця функціональність дозволяє розробникам більш впевнено працювати з критично важливими даними, знаючи, що інструментарій SSMS забезпечить високий рівень захисту та доступності інформації.

Використання SSMS у зв'язці з Visual Studio дозволяє розробникам максимально ефективно використовувати можливості SQL Server, оптимізуючи таким чином продуктивність і скалиброваність своїх додатків. Ці два інструменти разом формують потужний набір інструментів для розробки додатків на платформі .NET, що робить їх вибіром номер один для багатьох корпоративних розробників по всьому світу.

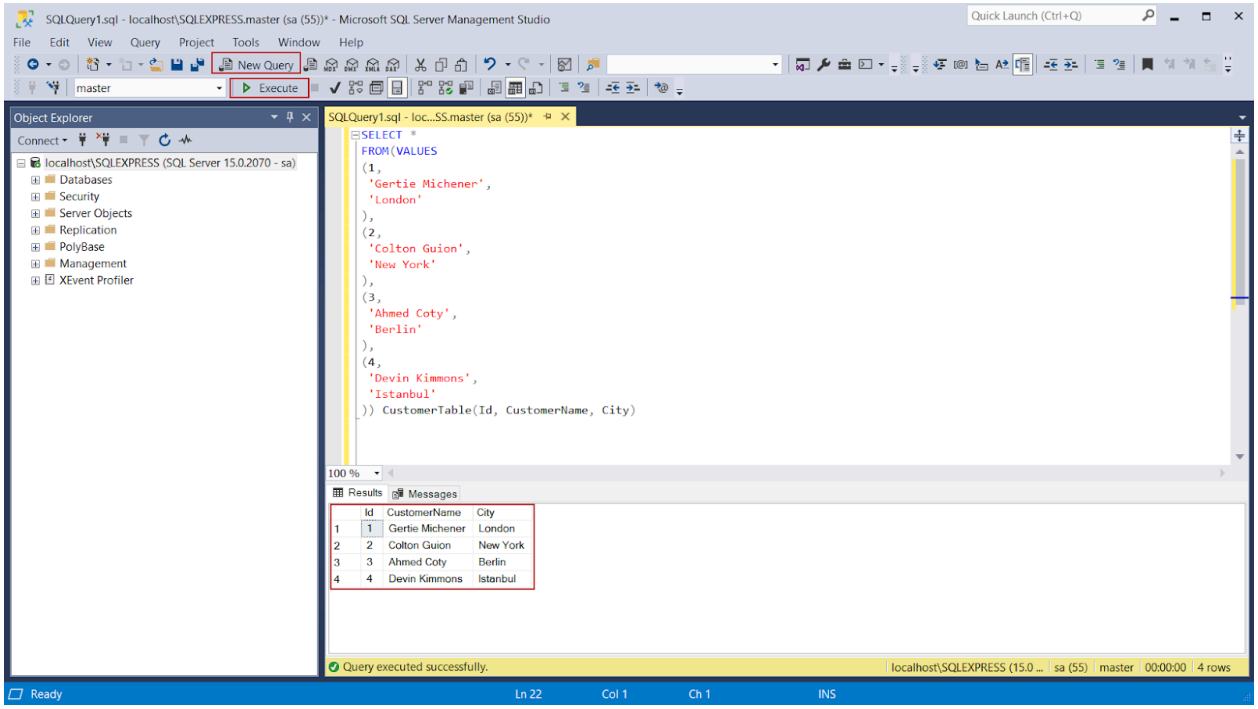

Рис. 2.2 Вигляд програми SSMS

Джерело: [18]

Використання у клієнтській частині розробки

У клієнтській частині розробки, VS Code часто використовується для написання HTML, CSS і JavaScript, які є основними технологіями веб-розробки. Редактор має вбудовані функції, які спрощують роботу з цими мовами, такі як підсвічування синтаксису, автодоповнення коду та відладка. Також можливість інтеграції з Git дозволяє розробникам ефективно керувати версіями своїх проектів.

Фреймворки для клієнтської частини

Існує багато фреймворків та бібліотек, які використовуються у клієнтській частині розробки, і багато з них добре інтегровані з VS Code через розширення та плагіни. Ось декілька популярних:

1. React - бібліотека від Facebook для створення користувацьких інтерфейсів. Вона дозволяє розробникам створювати великі веб-додатки, які можуть оновлювати дані без перезавантаження сторінки.
2. Angular - платформа розробки від Google для створення динамічних веб-додатків. Підтримує розширені можливості, такі як двосторонній зв'язок даних, модульність та автоматичне тестування.
3. Vue.js - ще один популярний фреймворк, який зосереджений на легкості інтеграції та гнучкості. Цей фреймворк є ідеальним вибором для тих, хто хоче швидко створювати прототипи інтерфейсів.
4. Bootstrap - потужний набір інструментів для створення адаптивних і мобільно-орієнтованих веб-сайтів. Включає в себе HTML- та CSS-шаблони для кнопок, форм, навігації та інших компонентів інтерфейсу.

Використовуючи VS Code для клієнтської розробки, програмісти можуть значно прискорити процес розробки завдяки широкому спектру доступних розширень та плагінів, які підтримують різні фреймворки і технології. Це дозволяє з легкістю переключатися між проектами, технологіями та інструментами, забезпечуючи високу ефективність та гнучкість роботи.

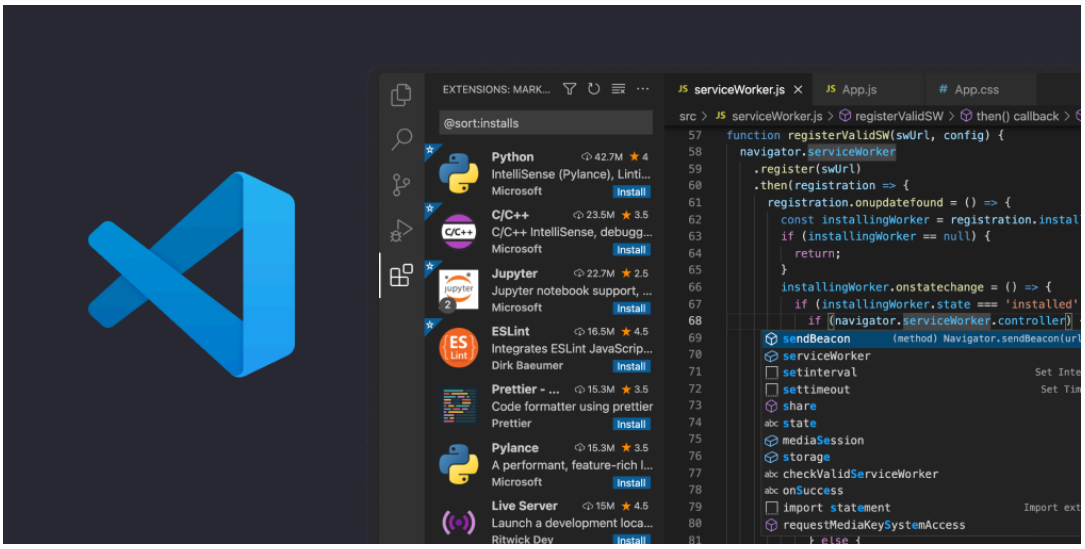

Рис. 2.3 Вигляд програми Visual Studio Code

Джерело: [19]

Figma вирізняється серед інших інструментів дизайну та прототипування своєю унікальною спроможністю до спільної роботи в режимі реального часу, що робить її особливо популярною серед команд, які працюють у великих і розподілених форматах. Відрізняючись від інших платформ, таких як Adobe XD та Sketch, які теж мають сильні сторони у дизайні та прототипуванні, Figma працює повністю у хмарі, що дозволяє користувачам доступатися до своїх проектів з будь-якого пристрою без потреби встановлення додаткового програмного забезпечення.

Однією з основних переваг Figma є її інтуїтивно зрозумілий інтерфейс, який дозволяє дизайнерам швидко адаптуватися та почати ефективну роботу, не витрачаючи час на складне навчання. Це стає в нагоді, особливо коли до проектів залучені особи, які не є професійними дизайнерами, такі як менеджери продуктів або маркетологи, дозволяючи їм легко вносити коментарі та редагування. Крім того, Figma має потужні інструменти для створення динамічних прототипів та комплексного тестування інтерфейсу, забезпечуючи детальний перегляд взаємодій користувачів з продуктом ще до його реалізації.

Ще одна значуща перевага Figma полягає в можливості інтеграції з іншими інструментами та платформами, що дозволяє дизайнерам ефективно взаємодіяти з іншими елементами розробки продукту, включаючи інженерію та маркетинг. Така

інтеграція сприяє гладкій та безперервній робочій течії, підвищуючи продуктивність та зменшуючи час виходу продуктів на ринок. Figma також підтримує широкий діапазон плагінів, які розширюють її функціональність та автоматизують рутинні задачі, що забезпечує додаткові можливості для кастомізації та оптимізації робочого процесу.

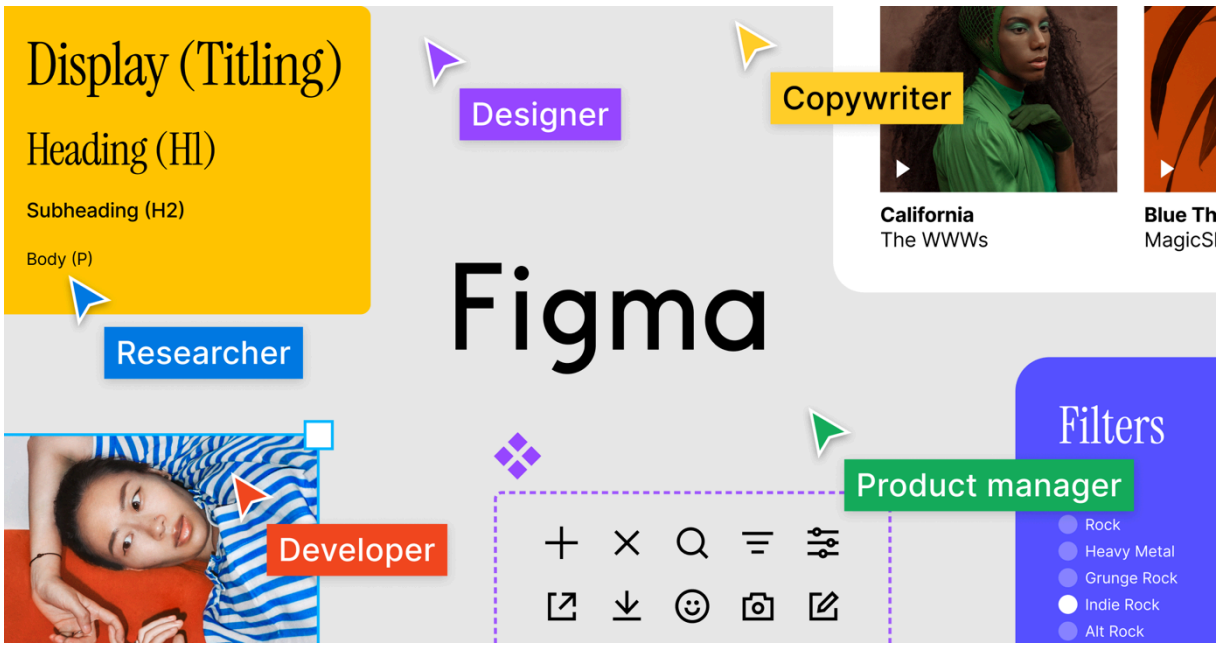

Рис. 2.4 Вигляд програми Figma

Джерело: [20]

Rider використовує багато з функціональних можливостей ReSharper, який є одним із найпопулярніших плагінів для Microsoft Visual Studio, що надає розширені можливості для рефакторингу, аналізу коду, швидшого переходу до визначень та багато іншого. Окрім того, Rider включає всі основні можливості інших IDE від JetBrains, такі як підтримка різних систем контролю версій, баз даних, Docker контейнерів, веб-розробки та багато інших.

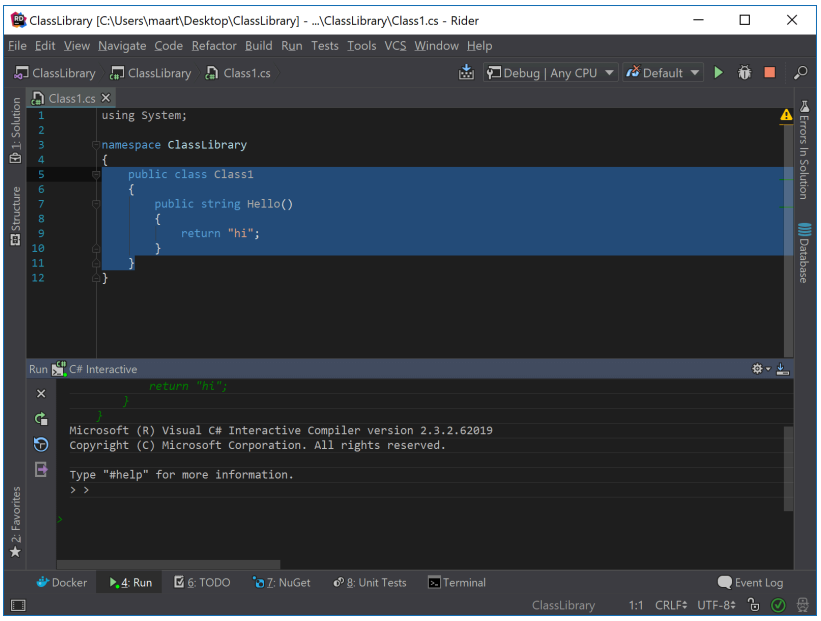

Рис. 2.5 Вигляд програми Rider

Джерело: [21]

Webstorm так само як і VS Code підтримує фрейморки для роботи клієнтської частини та є дуже популярним у великої кількості розробників. Все через стабільність JetBrains, який має багато цікавих рішень, є досить простим та інтуативно зрозумілим.

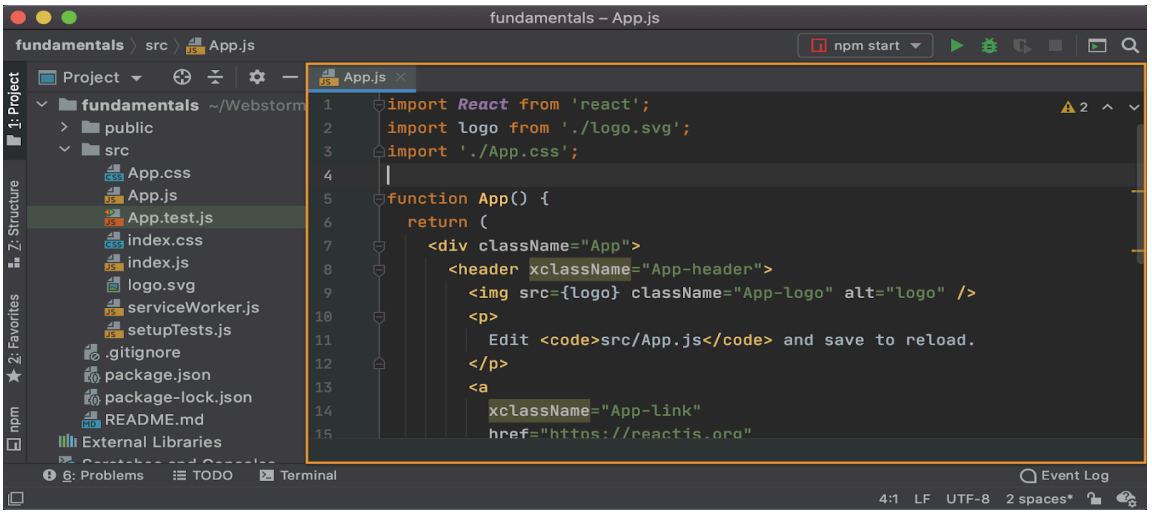

Рис. 2.6 Вигляд програми WebStorm

Джерело: [22]

2.2. Обгрунтування вибору конкретних програм для розробки вебзастосунку

Вибір середовища розробки для проектів на ASP.NET часто залежить від особистих уподобань, вимог проекту та зручності інструментів, які надає середовище. Microsoft Visual Studio (VS) та Visual Studio Code (VS Code) є двома дуже популярними варіантами для розробки на ASP.NET, і обидва мають певні переваги порівняно з іншими IDE, такими як Rider та WebStorm від JetBrains, особливо для проектів, що засновані на технологіях Microsoft.

Обираючи з яким IDE працювати, перш за все, потрібно врахувати функціанальність, зручність та доступність в роботі з .NET, який перш за все має дуже багато зв’язків в роботі з Microsoft. Так як C# це перш за все розробка Microsoft, а Visual Studio є повнофункціональним IDE, яке розроблено спеціально для розробників .NET, самим же Microsoft. Воно має глибоку інтеграцію з усіма технологіями та фреймворками Microsoft, зокрема з ASP.NET. Це означає, що розробники мають доступ до високоінтегрованих інструментів для дизайну, розробки, тестування та розгортання їхніх веб-додатків.

То ж коли ми маємо на меті роботу з VS, VS Code стає більш цікавою та привабливішою для нас при виборі IDE для клієнтської частини.

Що стосується Rider та WebStorm від JetBrains, ці інструменти також мають свої сильні сторони, особливо у підтримці різноманітних мов і фреймворків. Rider, зокрема, є відмінним варіантом для розробки на C# та .NET, забезпечуючи всі переваги ReSharper. Однак, хоча Rider ефективно підтримує розробку .NET та ASP.NET, його інтеграція з продуктами Microsoft може бути не настільки глибокою, як у Visual Studio. Це може створювати певні незручності.

Тому подальша робота виконувалася лише на Visual Studio та Visual Studio Code.

Також давайте не будемо забуватися про роботу з базою даних і тут також було декілька варіантів. Починаючи з думок роботи в Docker, який має можливість роботи з різними типами систем управління баз даних, таких як SQL Lite, PostgreSQL, MySQL і тд. Але тут вибір припав на те з чим було більше досвіду, тим більше

можливість зміни бази даних завжди залишається. Тож вибір впав Microsoft SQL server. А саме тому, що ця система управління базами даних має чудовий зв’язок з Visual Studio

Зв’язок між Visual Studio та SSMS

Інтеграція між Visual Studio та SSMS дозволяє розробникам ефективно управляти кодом, базами даних і ресурсами всередині одного проекту. Visual Studio може підключатися до SQL Server через Server Explorer, де розробники можуть легко переглядати, створювати та редагувати бази даних безпосередньо з IDE. Це значно спрощує процес розробки, оскільки не потрібно перемикатися між різними інструментами.

Висновки до розділу 2

У цьому розділі було проведено детальне порівняння та обґрунтування вибору програмного забезпечення для розробки серверної та клієнтської частин веб-додатку. Вибір відповідних інструментів є ключовим етапом, який впливає на ефективність розробки, можливості масштабування проекту, а також зручність подальшої підтримки і оновлень.

Порівнюючи різні інструменти для розробки, було визначено, що Microsoft Visual Studio та Visual Studio Code є одними з найкращих рішень для проектів на ASP.NET. Visual Studio забезпечує розробникам повнофункціональне інтегроване середовище розробки (IDE), яке глибоко інтегровано з усіма технологіями та фреймворками Microsoft, зокрема з ASP.NET. Це означає, що розробники можуть користуватися високорівневими інструментами для дизайну, написання коду, тестування та розгортання своїх веб-додатків. Visual Studio також надає потужні можливості для налагодження та управління проектами, що робить його незамінним інструментом для комплексних проектів.

З іншого боку, Visual Studio Code є легшим і більш гнучким редактором коду, який також підтримує широкий спектр розширень і налаштувань. Це робить його привабливим вибором для розробки клієнтської частини, де важливими є швидкість,

легкість у використанні та можливість налаштування під конкретні потреби розробника. Незважаючи на свою легкість, Visual Studio Code підтримує роботу з технологіями Microsoft і забезпечує відмінну інтеграцію з іншими інструментами розробки.

Також було розглянуто альтернативні IDE, такі як Rider та WebStorm від JetBrains. Rider є потужним інструментом для розробки на C# та .NET і має всі переваги ReSharper. Однак, хоча Rider ефективно підтримує розробку .NET та ASP.NET, його інтеграція з продуктами Microsoft може бути не настільки глибокою, як у Visual Studio, що може створювати певні незручності для розробників, які звикли до екосистеми Microsoft. WebStorm, своєю чергою, добре підходить для розробки фронтенду завдяки своїм можливостям роботи з різними мовами та фреймворками, але не надає такої глибокої інтеграції з ASP.NET.

При виборі системи управління базами даних було розглянуто кілька варіантів, включаючи використання Docker з різними системами управління базами даних, такими як SQLite, PostgreSQL, MySQL тощо. Однак, враховуючи досвід та можливості інтеграції, вибір припав на Microsoft SQL Server. Ця СУБД має чудову інтеграцію з Visual Studio, що дозволяє розробникам ефективно управляти кодом, базами даних і ресурсами всередині одного проекту.

Інтеграція між Visual Studio та SQL Server Management Studio (SSMS) є ще однією значущою перевагою. Visual Studio може підключатися до SQL Server через Server Explorer, що дозволяє розробникам легко переглядати, створювати та редагувати бази даних безпосередньо з IDE. Це значно спрощує процес розробки та управління базами даних, оскільки розробникам не потрібно перемикатися між різними інструментами, що підвищує продуктивність і зменшує ймовірність помилок.

Загалом, вибір Microsoft Visual Studio, Visual Studio Code та Microsoft SQL Server для розробки серверної та клієнтської частин веб-додатку є обґрунтованим та стратегічно правильним рішенням. Ці інструменти забезпечують ефективну розробку, зручність в управлінні проектами та можливість масштабування, що є ключовими факторами для успішної реалізації та підтримки веб-додатків.

РОЗДІЛ 3

ЕТАП РОЗРОБКИ ТА ВИБОРУ ТЕХНОЛОГІЙ ДЛЯ ПРОЄКТУ

3.1 Опис технологічного стеку та архітектури рішення

Почнемо з визначень та пояснень головних термінологій, які ми використовували готуючи цей проєкт.

Архітектура програмного забезпечення - це визначення структури, організації та взаємодії різних компонентів програми. Це концептуальний план, який визначає спосіб, яким програма буде організована та взаємодіяти з іншими системами або компонентами. В C# архітектура може бути побудована з використанням різних шаблонів проектування, таких як MVC (Model-View-Controller), MVVM (Model-View-ViewModel) тощо, для створення ефективних та добре організованих програм.

Фреймворк - це набір бібліотек, інструментів та ресурсів, які надають різноманітні функціональні можливості для розробки програмного забезпечення. У C# популярними фреймворками є .NET Framework і .NET Core (тепер перейменований в .NET), які надають широкі можливості для розробки програм на різних платформах. Фреймворки надають готові рішення для роботи з мережами, базами даних, графікою, безпекою та іншими аспектами розробки.

Бібліотеки - це набір підготовлених кодових модулів, які можуть використовуватися для різних цілей в програмі. Вони забезпечують певний функціонал, який можна використовувати в програмі без необхідності написання коду з нуля. У C# багато бібліотек, таких як Entity Framework для роботи з базами даних, ASP.NET для розробки веб-додатків, Newtonsoft.Json для роботи з JSON, є популярними і широко використовуються.

Отже, архітектура визначає організацію програми, фреймворки надають інструменти для розробки, а бібліотеки - готові модулі для використання у програмі. Використання цих концепцій сприяє розробці ефективного та функціонального програмного забезпечення на мові програмування C#.

Архітектура проєкту

За основу для було обрано багатоівневу архітектуру N-Layer. Так як цей проєкт передбачає в собі роботу вебзастосунку віддали перевагу найбільш популярну архітектуру яка дає можливість працювати з інтерфейсом користувача (UI), бізнес логікою (BLL) та доступ до данних (Data Access).

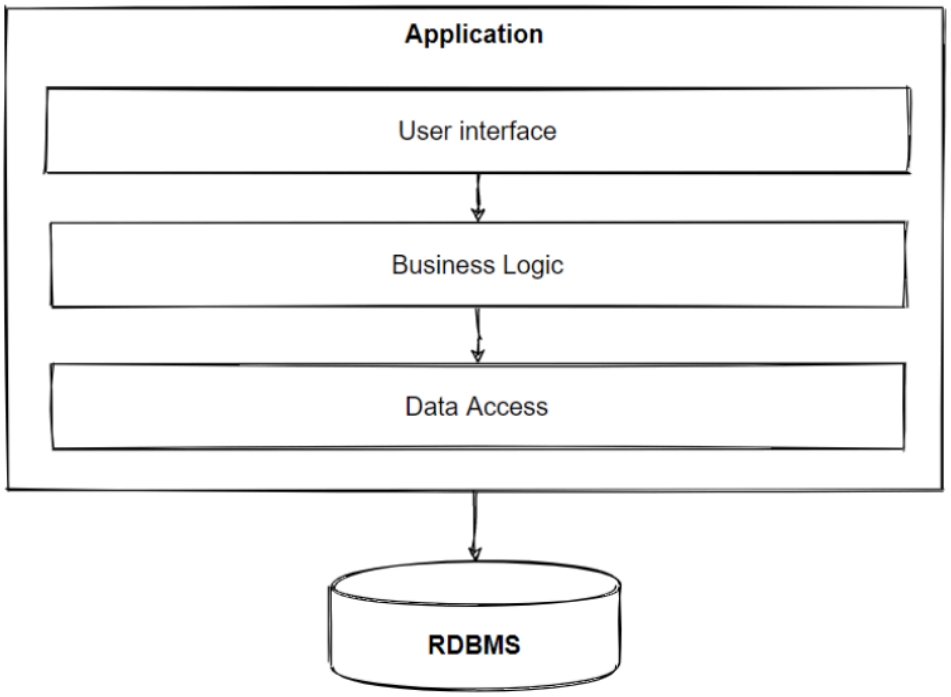

Рис. 3.1 Приклад N-Layer архітектури

Джерело: [24]

3.1.1 Опис архітектури вебзастосунку написаному на Blazor

Архітектура N-Layer використовується для організації та розділення різних рівнів функціональності в програмі. Цей підхід часто застосовується в проєктах, які написані на Blazor Server App, для створення структурованого та модульного коду. Основні рівні (шари) в архітектурі N-Layer для проєкту Blazor Server App можуть виглядати так.

1. User Interface (інтерфейс користувача):

Цей рівень відповідає за інтерфейс користувача, включаючи усі компоненти, сторінки та взаємодію з користувачем. У Blazor Server App це може бути компоненти Blazor, які відображають інформацію користувачеві та обробляють його взаємодію з додатком. В проєкті прописувалися сторінки застосунку.

2. Business Logic Layer (Шар бізнес-логіки):

Цей рівень містить бізнес-логіку та правила обробки даних. Тут розміщуються класи, які виконують обчислення, валідацію, обробку даних та будь-яку логіку, пов'язану з бізнес-процесами додатку. В Blazor Server App це сервіси або класи, які відповідають за обробку даних та бізнес-логіку.

3. Data Access Layer (Шар доступу до даних):

Цей рівень відповідає за доступ до даних, включаючи взаємодію з базами даних або зовнішніми джерелами даних. У Blazor Server App це класи або сервіси, які взаємодіють з базою даних або іншими джерелами даних для отримання та збереження інформації.

4. Infrastructure Layer (Інфраструктурний шар):

Цей рівень містить загальні утиліти, сервіси та інфраструктурні компоненти, такі як логування, кешування, аутентифікація, налаштування та інше. В Blazor Server App це може бути інфраструктурний код, який відповідає за загальні функції та утиліти, які використовуються в різних частинах додатку.

Кожен з цих рівнів може бути розділений на підшари або модулі, які дозволяють краще організувати функціональність та полегшують розробку, тестування та підтримку додатку в Blazor Server App. Застосування архітектури N-Layer дозволяє створити добре структурований та легко розширюваний додаток.

Або можна це назвати простими словами Clean Architecture, що має на увазі розділення проєкту на логічні шари та чистоту коду.

3.1.2 Стек технологій використаних у розробці

- ASP.NET Core 6 – це кросплатформове, високопродуктивне середовище з відкритим вихідним кодом для створення сучасних хмарних програм, підключених до Інтернету.[1]

- Blazor Server App - це тип Blazor додатків, в якому весь код на стороні клієнта виконується на сервері. У цьому підході весь процес взаємодії з додатком відбувається через веб-сокети, ініційовані SignalR, що забезпечує зв'язок між клієнтом (браузером) та сервером.[2]
- SQLServer - постачальник бази даних SqlServer дозволяє використовувати Entity Framework Core з Microsoft SQL Server[3]
- EntityFrameworkCore - це легка, розширювана, кросплатформна версія популярної технології доступу до даних Entity Framework з відкритим кодом, що дозволяє створювати моделі для БД різних типів[3]

3.2 Опис створення вебзастосунку

Головним користувачем цього застосунку буде людина яка працює по основних засобах в компанії. За часту це хтось з відділу бухгалтерії. Тож беручи до уваги всі побажання та скарги таких користувачів було розроблено застосунок який має все, що потрібно для таких користувачів. Для цього було прорблено наступні кроки.

Спочатку створюємо проєкт в Visual Studio. Щоб створити проєкт, ми повинні обрати рішення та створити його в VS. Для цього обираємо *blazor server app*

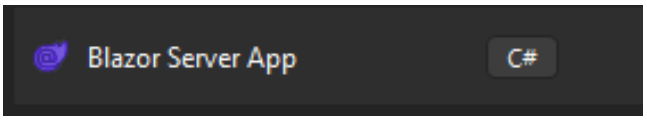

Рис. 3.2 Приклад створення Solution

Джерело: створено автором

Одразу додаємо в наш Solution декілька додаткових проєктів в яких будемо зберігати певні класи для того, щоб розділити наш проєкт на певні частини, мати більше структури, порядку та розуміння що за що відповідає.

Тож натискаємо на наш Solution ПКМ, обираємо додати та створити новий проєкт. Все як на фото

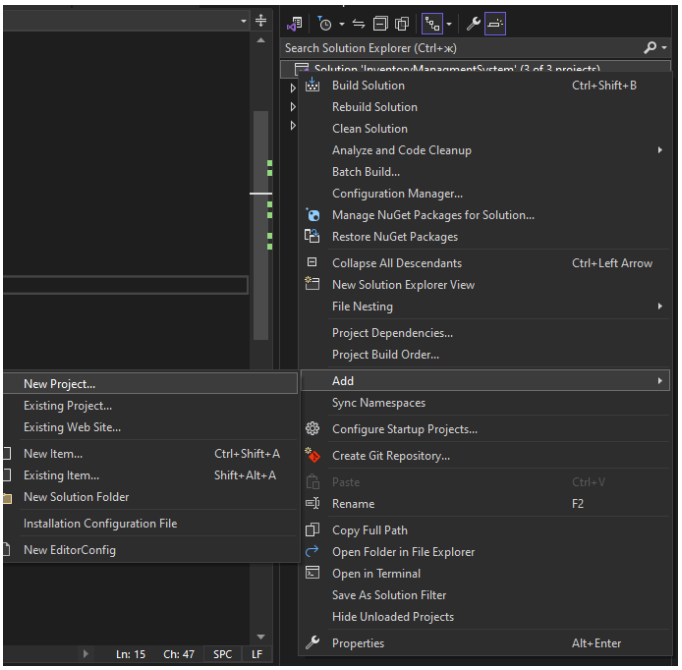

Рис. 3.3 Приклад додавання проєктув в Solution

Джерело: створено автором

Далі додаємо два проєкти, а саме Console App

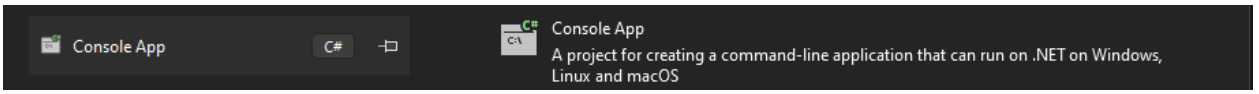

Рис. 3.4 Створення консольного проєкту

Джерело: створено автором

Після усіх цих дій ми повинні отримати дві консольні програми та одну серверну програму як на фото.

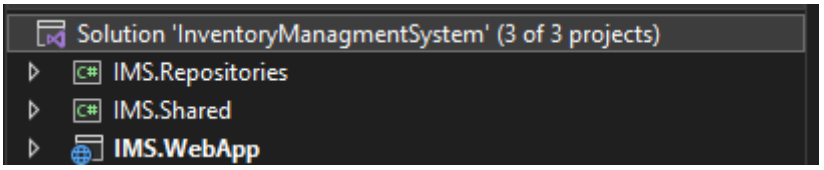

Рис. 3.5 Вигляд усіх проєктів в Solution

Джерело: створено автором

Далі переходимо до частини розробки.

Тож спочатку зробимо першочергові налаштування. Для цього нам знадобиться спочатку підключити базу даних в файлі *appsettings.json*. Для цього ми повинні прописати сервер для бази даних (в нашому випадку це локальне

підключення до нашого ж комп’ютера) та ім’я бази даних (саме те яке ми хочемо, щоб надалося при створені міграції)

```
{
  "ConnectionStrings": {
    "connectionString": "Server=DESKTOP-FQC2FCR\\SQLEXPRESS;Database=InventoryDB;Trusted_Connection=True;MultipleActiveResultSets=true"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*"
}
```

Рис. 3.6 Приклад вигляду файлу appsetings.json

Джерело: створено автором

Після того як ми зв’язали з нашим сервером, одразу підключимо всі потрібні бібліотеки для нашого проєкту. Для цього нам потрібно натиснути ПКМ на один з наших проєктів та обрати розділ *Manage NuGet Packages*, як на фото:

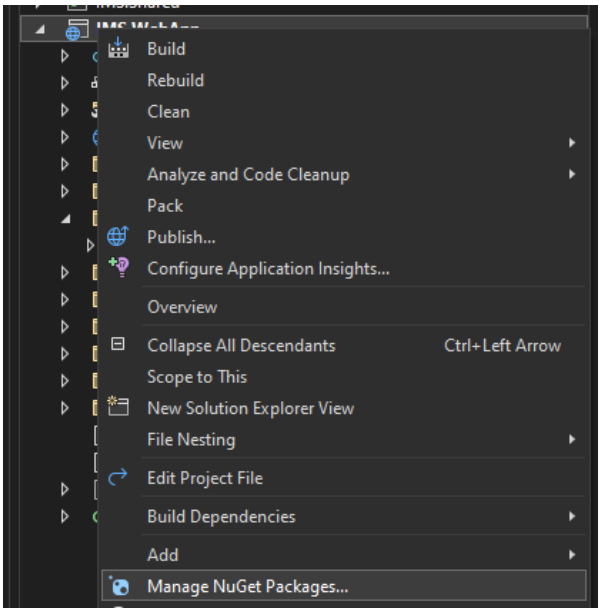

Рис. 3.7 Керування пакетами бібліотек NuGet

Джерело: створено автором

А що таке NuGet Packages:

NuGet packages у Visual Studio є інструментом для управління бібліотеками та залежностями у .NET проектах. Вони дозволяють розробникам легко додавати, оновлювати та видаляти зовнішні бібліотеки у своїх проектах. NuGet забезпечує централізоване сховище, з якого можна завантажити необхідні пакети, а також підтримує автоматичне вирішення залежностей між пакетами, що значно спрощує процес інтеграції зовнішнього коду. Використовуючи NuGet, розробники можуть

зосередитися на написанні власного коду, замість того щоб витрачати час на ручне керування бібліотеками та їхніми версіями.

Для нашого проєкту потрібно підключити наступні:

1. Microsoft.AspNet.Identity.Core
2. Microsoft.AspNetCore.Identity
3. Microsoft.AspNetCore.Identity.EntityFrameworkCore
4. Microsoft.EntityFrameworkCore
5. Microsoft.EntityFrameworkCore.Design
6. Microsoft.EntityFrameworkCore.SqlServer
7. Microsoft.EntityFrameworkCore.Tools
8. MudBlazor

Після цих дій перейдемо до створення класів. Тож в проєкті який в мене називається як IMS.Shared створюємо три класи, а саме:

- Створено класи Recipient, Tools, Inventory

```
public class Tools
{
    1 reference
    public int ToolsId { get; set; }
    5 references
    public string TypeName { get; set; }
    4 references
    public string CompanyName { get; set; }
    4 references
    public int InventoryNumber { get; set; }
    4 references
    public int PurchasePrice { get; set; }
    4 references
    public string DateTransfer { get; set; }
    4 references
    public string Conditions {  get; set; }
    3 references
    public int InventoryId { get; set; }
    1 reference
    public Inventory Inventory { get; set; }
    1 reference
    public Recipient Recipient {  get; set; }
    3 references
    public int RecipientId { get; set;}

}
```

Рис. 3.8 Приклад класу Tools

Джерело: створено автором

Далі нам потрібно створити Інтерфейси. Але спочатку, давайте розберемо, що таке інтерфейси

Інтерфейси в ASP.NET визначають контракти для класів, які реалізують ці інтерфейси. Вони задають методи, властивості та події, які клас повинен реалізувати,

забезпечуючи стандартизовану взаємодію між різними компонентами програми. Це сприяє модульності, повторному використанню коду та полегшує тестування.

Створимо окрему папку в проєкті IMS.WebAPP

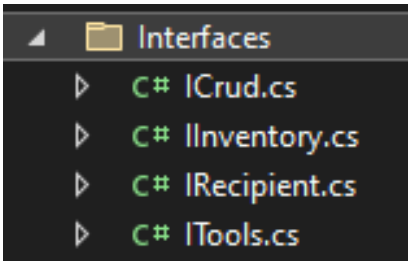

Рис. 3.9 Приклад папки Interfaces з інтерфейсами

Джерело: створено автором

В ній створимо 4 інтерфейси.

- ICrud
- IInventory
- IRecipient
- ITools

```
namespace IMS.WebApp.Interfaces
{
    1 reference
    public interface ICrud<T> where T : class
    {
        4 references
        bool Add(T entity);
        4 references
        bool Update(T entity);
        4 references
        bool Delete(T entity);
        1 reference
        T GetById(int id);
        13 references
        List<T> GetAll();
    }
```

Рис. 3.10 Приклад інтерфейсу ICrud

Джерело: створено автором

Тепер розберемо для чого потрібен кожен з інтерфейсів:

1. ICrud<T>:

Інтерфейс ICrud<T> визначає стандартні операції для керування об'єктами типу T. Це інтерфейс для створення, читання, оновлення та видалення (CRUD) об'єктів. Він містить методи для додавання, оновлення, видалення об'єктів, отримання об'єкта за ідентифікатором та отримання всіх об'єктів. Використовується для забезпечення базових CRUD-операцій у різних типах класів.

2. IInventory:

Інтерфейс IInventory визначає контракт для компонентів, які пов'язані з управлінням інвентарем. Хоча зараз він порожній, його можна розширити методами, що стосуються інвентаризації, наприклад, додавання товарів, відстеження кількості, отримання інформації про запаси і т.д.

3. IRecipient:

Інтерфейс IRecipient визначає контракт для компонентів, які представляють отримувачів. Як і IInventory, він наразі порожній, але може бути розширений методами, що стосуються управління отримувачами, наприклад, додавання нових отримувачів, оновлення інформації про них, видалення та інше.

4. ITools:

Інтерфейс ITools визначає контракт для інструментів або утиліт, які можуть бути використані в додатку. Хоча зараз він порожній, можна додати методи для різних допоміжних операцій, що полегшують основні бізнес-процеси або забезпечують додаткову функціональність.

Ці інтерфейси допомагають структурувати код, роблять його більш організованим і дозволяють легко змінювати та розширювати функціонал без порушення існуючих компонентів.

Далі трохи зупинимося на роботі зі службами. Адже саме вони реалізовують роботу з великою частиною розробленою для backend-ду

Тож спочатку створимо папку в проєкті IMS.WebApp

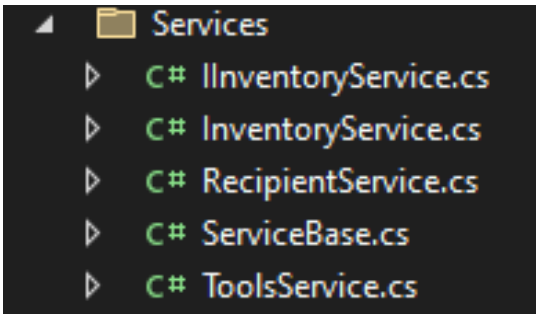

Рис. 3.11 Приклад папки Services

Джерело: створено автором

І тепер зупинимося трохи на розборі коду та пояснення як це працює та для чого.

Код який ми розбиремо, знаходиться в ДОДАТКУ

Давайте розберемо основні елементи цієї служби:

Клас ServiceBase<T>

-ServiceBase<T> — це шаблоний клас-сервіс, який реалізує інтерфейс ICrud<T>. Шаблонний параметр `T` вказує на тип об'єктів, з якими ця служба буде працювати.

Конструктор

- public ServiceBase(ApplicationDbContext context) — конструктор, який отримує екземпляр контексту бази даних (`ApplicationDbContext`). Цей контекст дозволяє службі взаємодіяти з базою даних.

Властивість Entities

- protected IQueryable<T> Entities { get => context.Set<T>(); } — це властивість, яка надає доступ до колекції об'єктів типу `T` у контексті бази даних. Вона повертає IQueryable<T>, що дозволяє виконувати складані запити до бази даних.

Операції CRUD

- Add(T entity), Delete(T entity), GetAll(), GetById(int id), Update(T entity) — ці методи реалізують стандартні операції створення, читання, оновлення та видалення даних в базі даних для об'єктів типу `T`.

- Add — додає новий об'єкт `T` до бази даних.

- Delete — видаляє об'єкт `T` з бази даних.

- GetAll — повертає список всіх об'єктів типу `T`.

- GetById — повертає об'єкт типу `T` за його ідентифікатором.

- Update — оновлює існуючий об'єкт `T` в базі даних.

Обробка винятків

- У кожному методі CRUD є конструкція `try-catch`, яка спробує виконати операцію з базою даних. Якщо виникне помилка, метод поверне `false`.

Ця служба надає загальний шаблон для взаємодії з базою даних та виконання операцій CRUD для будь-якого типу об'єктів в ASP.NET або ASP.NET Core додатках. Вона може бути розширена або використана як базовий блок для створення специфічних сервісів для конкретних типів даних у вашому додатку.

Далі перейдемо до створення та перевірки роботи цього в браузері. Але перед тим ми проведемо міграцію та створимо тестові сторінки, перед переходом до розробки клієнтської частини та інтеграції дизайну.

Тож, що таке міграція та як її провести?

Міграція у контексті баз даних та Entity Framework Core — це процес управління змінами в схемі бази даних за допомогою коду. Вона дозволяє відстежувати зміни в моделі даних і автоматично застосовувати ці зміни до бази даних. Це особливо корисно для підтримки синхронізації між кодом програми та базою даних протягом розробки.

Кроки для проведення міграції:

- Створити клас контексту
    - Для цього в проєкті IMS.WebApp створимо папку **Data**, а в ній створимо клас DataContext.cs. Код даного класу можна переглянути в ДОДАТКУ.
- Додати початкову міграцію:
    - Для цього використаєил консоль диспетчера пакетів створення початкової міграції. І в ній напишемо наступну команду: Add-Migration **та назву міграції**
    - Це створить файли міграції, що включають початковий набір таблиць і стовпців, які потрібно створити в базі даних.

- Застосувати міграції до бази даних:
    - Після створення міграцій, застосуємо їх до бази даних за допомогою наступної команди: Update-Database

Далі перейдемо до папки Pages та змінимо в ній декілька сторінок, щоб протестувати роботу.

Наступні декілька фотографій буду показувати роботу сторінок:

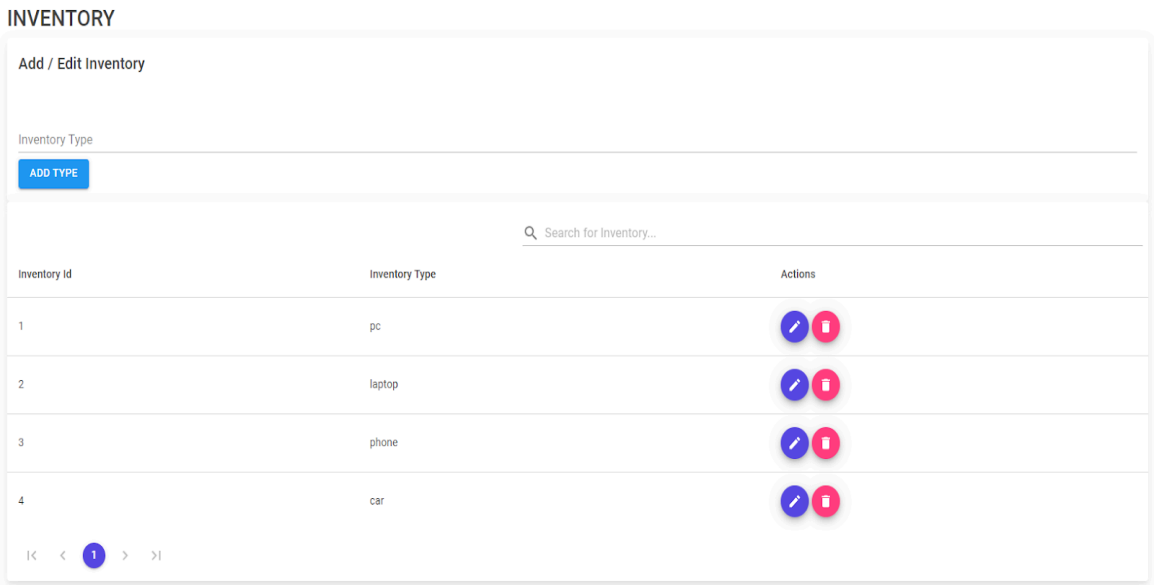

Рис. 3.12 Реалізація сторінки Inventories

Джерело: створено автором

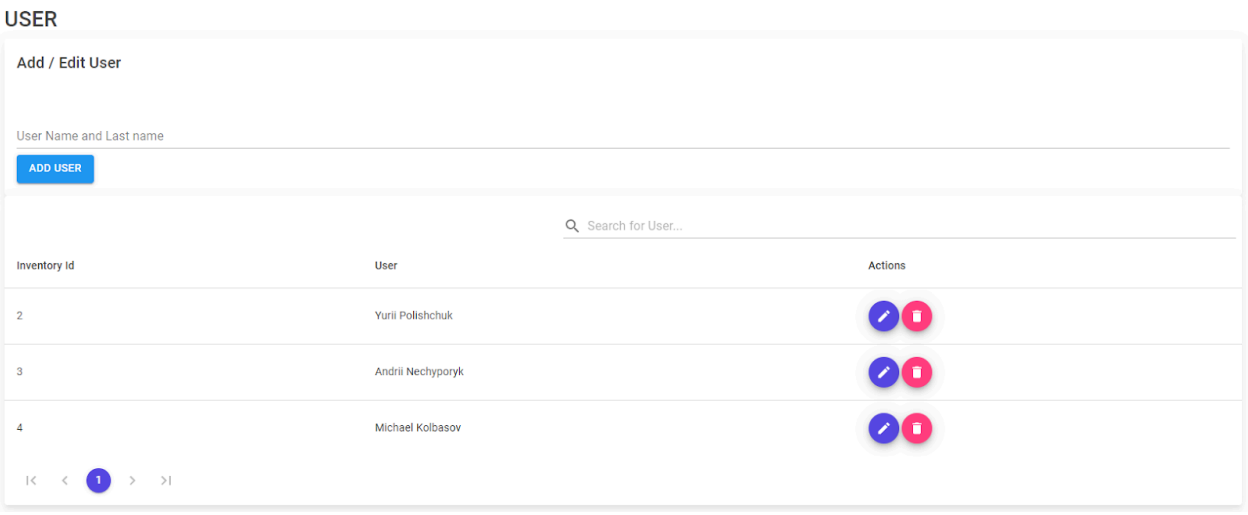

Рис. 3.13 Реалізація сторінки Recipient

Джерело: створено автором

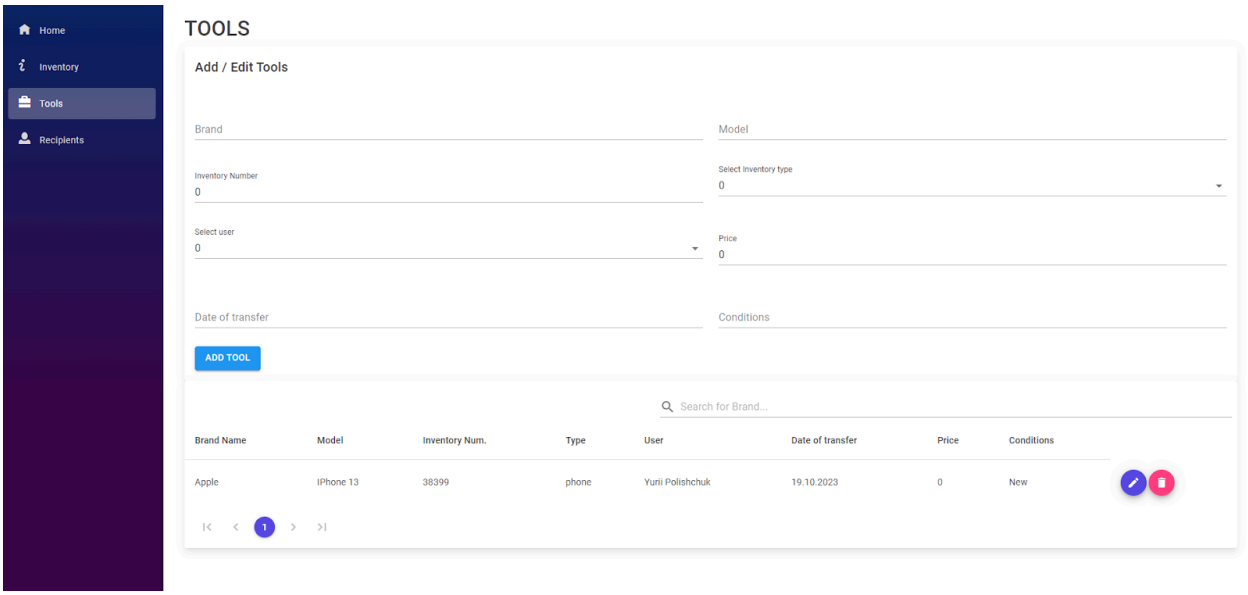

Рис. 3.14 Реалізація сторінки Tools

Джерело: створено автором

Наступним етапом став перехід до розробки клієнтської частини.

2.3 Сервіси клієнтської частини

Спочатку зупинимося на наступних пунктах:

- Що таке інтеграція готового дизайну в проєкт
- Який дизайн було обрано на сьогоднішній момент
- Чому було обрано інтегрувати дизайн, а не створити власний
- Які інструменти використовувалися для цього
- Які інші варіанти створення клієнтської частини

Що таке інтеграція готового дизайну в проєкт

Тож розпочнемо аналіз та пояснення з що таке інтеграція готового дизайну в проєкт:

Інтеграція готового дизайну в проєкт — це процес, при якому готовий макет чи шаблон веб-дизайну адаптується та впроваджується в реальний веб-застосунок або веб-сайт. Це передбачає перенесення візуальних елементів, стилів і структури з дизайну у функціональний код, який буде використовуватися в проєкті.

Процес інтеграції починається з аналізу та розуміння дизайну, який може бути створений у вигляді файлів HTML/CSS готових шаблонів з різних джерел. Після цього дизайн розбивається на окремі компоненти та елементи, такі як заголовки, підзаголовки, навігаційні панелі, кнопки, таблиці тощо. Кожен з цих елементів перекладається у відповідний код. HTML використовується для структурування контенту, CSS для стилізації, а JavaScript додавання інтерактивності та динамічних функцій. Важливим кроком є підключення стилів і скриптів до основної структури проєкту, щоб дизайн правильно відображався на різних сторінках і пристроях.

Також інтеграція може включати адаптацію дизайну для різних роздільних здатностей екранів, забезпечуючи його гнучкість на різних девайсах. Це гарантує, що сайт буде виглядати добре як на настільних комп'ютерах й на мобільних пристроях.

Інтеграція дизайну сайту в веб-застосунок включала кілька ключових етапів, від зміни головного макету до додавання нових сторінок із функціоналом керування даними. Ось як я підійшов до цього процесу:

1. Зміна головного Layout

Я почав із зміни основного макету (`MainLayout.razor`), щоб він відповідав дизайну і стилю нового шаблону, який було вибрано. Це включало інтеграцію CSS і JavaScript файлів шаблону, які були необхідні для відтворення його візуального стилю і поведінки.

2. Реорганізація сторінок

Кожну сторінку застосунку я переписав, використовуючи новий шаблон. Це включало адаптацію структури HTML та використання компонентів Bootstrap 5, які надав шаблон, для забезпечення консистентності UI/UX по всьому застосунку.

3. Додавання нових сторінок

Я створив нові сторінки для додавання, видалення і редагування даних, які користувачі могли керувати через веб-інтерфейс. Кожна сторінка була розроблена для того, щоб забезпечити інтуїтивно зрозумілий і зручний інтерфейс.

4. Навігація із шаблону

Я інтегрував навігацію, яка була частиною шаблону, забезпечивши легке переміщення між сторінками і розділами застосунку. Це включало оновлення компонентів навігації для відображення активних сторінок та підсвічування поточних розділів.

5. Підключення до Backend

Всі нові та існуючі сторінки були підключені до backend частини застосунку, використовуючи запити до API для взаємодії з базою даних. Це дозволило здійснювати CRUD операції з даними безпосередньо з користувацького інтерфейсу.

6. Додавання динамічних таблиць

Я додав таблиці, які динамічно заповнюються даними з бази даних, забезпечивши можливість користувачам бачити актуальну інформацію, а також керувати цими даними через інтерфейс (редагування, видалення тощо).

7. Поліпшення UI/UX

Завершальний етап полягав у вдосконаленні користувацького досвіду, включаючи тестування інтерфейсу на різних пристроях та у різних браузерах, для забезпечення надійності та зручності користування.

Ці кроки дозволили не тільки інтегрувати новий дизайн, але й забезпечити, що застосунок є функціональним, зручним і естетично привабливим для кінцевого користувача.

Який дизайн було обрано на сьогоднішній момент

Витративши купу часу на спроби створити власний дизайн на платформі Figma, я зрозумів, що я не дизайнер і всі мої спроби створити щось своє, перетворюються на повторення готового рішення від Blazor, в якому я змінював кольори та послідовність якихось об’єктів, що я міг зробити це й так, підправивши трохи готового коду. Тому я вирішив спробувати змінити вектор та знайти готовий дизайн та інтегрувати його.

За основу було взято макет **Soft UI**[23]. Він задовільняв всі мої потреби та мав гарні готові елементи.

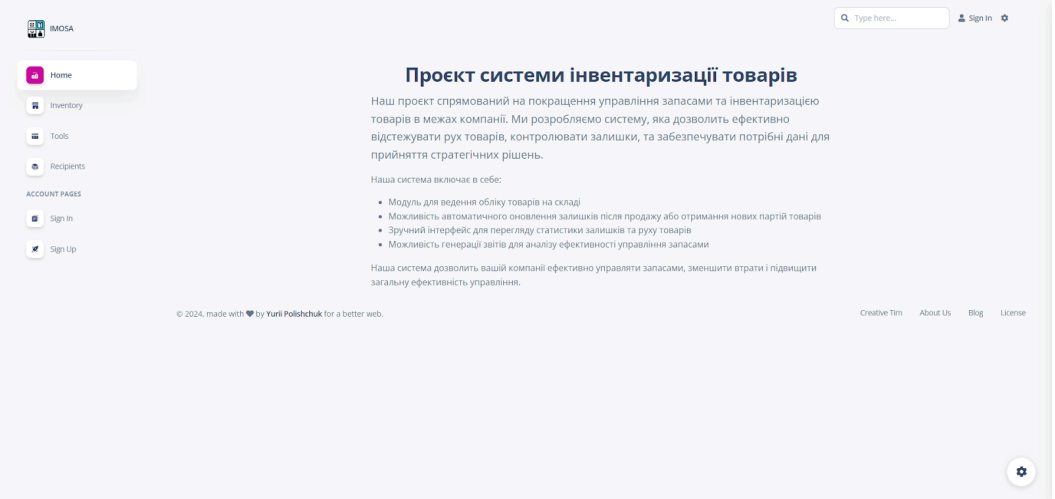

Рис. 3.15 Приклад дизайну

Джерело: створено автором

Які інструменти використовувалися для цього

Для того, щоб інтегрувати дизайн було прийняте рішення працювати й далі в Visual Studio та Blazor, й інтегрувати код на пряму, аніж робити розробку клієнтської частина VS Code та React.

Тепер пояснюю чому. Все тому що над таким проєктом я працював сам. Я мав мало досвіду роботи на React та мало знань в сфері роботи з JavaScript. А інтеграція за допомогою готових частив в Blazor була мені близька та мала частину переваг. А саме такі як:

- Легко змінити дизайн в будь-який момент
- Легкий в тестуванні
- Гарна взаємодія з backend-ом

Щоб досягнути максимального успіху роботи клієнтської частини з серверною частиною, нам потрібно буде створити контролери та повернутися до створеного нами консольного проєкту для репозиторіїв.

Одразу потрібно обговорити різницю між контролерами та репозиторіями, й інтерфейсів та служб

Контролери та репозиторії відрізняються від інтерфейсів та служб своєю призначеністю та контекстом використання.

У даному проєкті контролери та репозиторії використовуються для обробки запитів та роботи з даними відповідно. Контролери приймають запити від клієнтської частини додатка, обробляють їх та повертають відповіді. Репозиторії, у свою чергу, забезпечують доступ до даних та виконують з ними операції, такі як збереження, видалення та оновлення. Це дозволяє структурувати логіку додатка та відокремлювати її від представлення даних.

Інтерфейси та служби використовуються для реалізації бізнес-логіки додатка та виконання специфічних завдань. Вони можуть використовуватися для роботи з даними, обробки запитів та взаємодії з іншими компонентами додатка.

У контексті переходу на роботу з серверною частиною, контролери та репозиторії можуть бути більш зручними в використанні, оскільки вони відповідають за специфічні аспекти роботи додатка. Вони дозволяють відокремлювати логіку додатка від представлення даних та забезпечують більшу гнучкість у розробці та підтримці додатка.

Тож підсумуємо, що нам потрібні контролери, репозиторії, інтерфейси та служби для підтримки додатку. Одні відповідають за створення бізнес логіки та роботи з базою даних, а інші за візуалізацію та обробки даних в базі даних.

Тож давайте переглянемо як виглядає перероблена частина коду однієї з сторінок. Приклад коду в ДОДАТКУ.

Та можемо переглянути сторінки на фото:

Рис. 3.16 Реалізація таблиці на сторінці Inventory

Джерело: створено автором

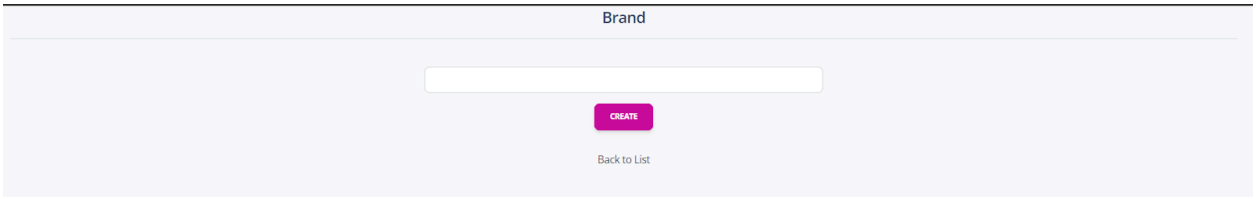

Рис. 3.17 Реалізація сторінки з додаванням

Джерело: створено автором

Також було написано скрипт для друку як сторінки так і окремої таблиці, що досить зручно в бухгатерській справі. Приклад як виглядає вивід до друку на фото:

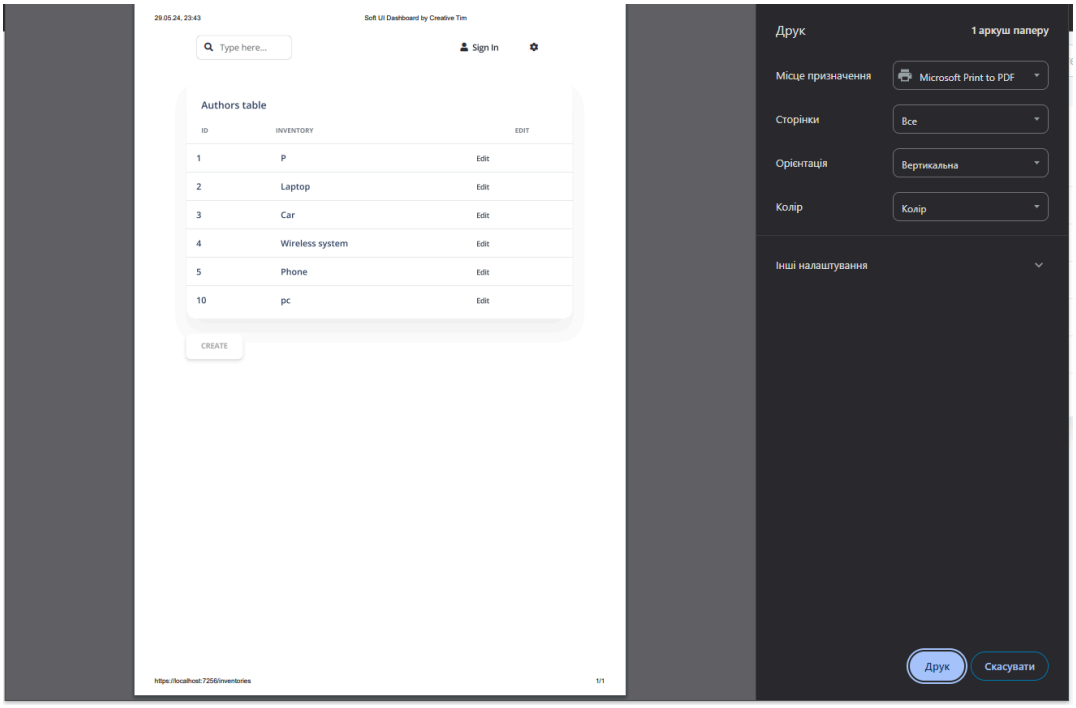

Рис. 3.18 Реалізація сторінки з додаванням

Джерело: створено автором

Також хочеться приділити увагу розробленому логотипу який був створений власноруч за допомогою Figma. Щоб створити такий логотип нам потрібно

встановити квадратні рамки, які містить весь логотип. Використати інструмент прямокутника, щоб створити чотири однакові за розміром квадранти. Кожен квадрант буде містити одну з чотирьох піктограм.

Для піктограми у верхньому лівому куті почнемо з додавання кола та розмістимо його над трьома фігурами листя, розташованими у формі трикутника. Додаймо горизонтальну лінію під цими елементами, щоб завершити дизайн.

У верхньому правому квадранті створимо простий значок книги, намалювавши два прямокутники поруч, злегка нахиленими один до одного, щоб представити відкриті сторінки книги. Використаємо градієнтні заливки для досягнення ефекту затінення.

Для нижнього лівого значка створіимо фігуру з колом для голови, півколом для тіла та двома прямими лініями для ніг. Розмістимо елементи так, щоб утворити спрощену фігуру людини з піднятими руками.

У правому нижньому квадранті намалюємо форму сумки, використовуючи комбінацію трапеції та маленького трикутника зверху для зав’язаної частини. Налаштуємо пропорції відповідно до вигляду значка.

Протягом усього процесу проектування використовужмо інструменти вирівнювання Figma, щоб переконатися, що всі елементи розташовані по центру та рівномірно розташовані. Застосовуємо відповідні заливки та обведення відповідно до колірної схеми значка, зосереджуючись на використанні сталої товщини лінії та палітри кольорів, щоб підтримувати цілісність усіх чотирьох значків.

Рис. 3.19 Емблема сайту

Джерело: створено автором

3.3 Перспективи розвитку та впровадження додатку на ринок веб додатків

Додаток, спрямований на полегшення роботи компаній в інвентаризації основних засобів, може мати значний потенціал на ринку веб-додатків. Ось деякі перспективи розвитку та впровадження такого проєкту:

1. **Оптимізація робочих процесів:**

Додаток, спрямований на інвентаризацію основних засобів, може включати функціональність для автоматизації процесів збору, оновлення та відстеження даних про активи компанії. Це дозволить значно зекономити час та зусилля працівників, які відповідають за управління цими активами.

2. **Інтеграція та сумісність:**

Один із ключових аспектів успішного додатку полягає у можливості інтеграції з існуючими системами компанії та іншими важливими додатками. Це може включати зв'язок з системами управління запасами, обліку, фінансами, що забезпечить більш повне управління активами.

3. **Функціональність звітності та аналізу:**

Власний застосунок дасть можливість простіше інтегрувати програму власного виробництва, що буде дуже зручним так як вся ідеологія і розуміння як це працює і що потрібно саме вам.

4. **Безпека даних:**

Збереження та захист інформації про активи є критично важливим. Застосування кращих практик забезпечення безпеки даних, таких як шифрування, автентифікація, авторизація та резервне копіювання, є обов'язковими.

5. **Доступність та масштабованість:**

Важливою є можливість роботи додатку на різних пристроях та платформах, забезпечуючи доступ до інформації з будь-якої точки. Крім того, додаток повинен бути готовим до масштабування, спроможним витримати зростання обсягів даних та користувачів.

6. **Користувацький досвід та інтерфейс:**

Успіх додатку значно залежить від зручного та інтуїтивно зрозумілого інтерфейсу. Простота використання та забезпечення користувачам необхідного функціоналу будуть важливими аспектами.

7. **Стратегія впровадження та підтримки:**

Важливо мати стратегію впровадження, тестування та підтримки додатку після його запуску. Крім того, реакція на відгуки користувачів та їхні вимоги щодо покращення можливостей додатку також грає важливу роль.

8. **Додавання нового функціоналу:**

За допомогою роботи разом з користувачами, можна додавати новий функціонал та надавати підтримку для юзерів. Однією з можливих нововведень це можу бути додавання можливості створення накладної на переміщення одразу зі списку, щоб можна було це роздрукувати, чи просто мати електронний варіант накладної на переміщення для майбутнього підписання та підтвердження дійсності передачі, та отримання продукту, техніки, чи будь-якої опридаткованого товару.

Якщо успішно впровадити та розвивати додаток для полегшення інвентаризації основних засобів, то від того, наскільки добре він відповість потребам користувачів, забезпечить ефективність та зручність використання, а також наскільки він буде готовий до майбутніх змін та розширень.

Висновки до розділу 3

Під час аналізу технологій та рішень для нашого проекту, ми зробили акцент на виборі найбільш оптимальних інструментів для реалізації технічних вимог та бізнес-цілей. Основу проекту складає многошарова N-Layer архітектура, яка ідеально підходить для розробки веб-застосунків завдяки своїй гнучкості та масштабованості. Ця архітектура дозволяє нам чітко розділити логіку інтерфейсу користувача, бізнес-логіку та шар доступу до даних, що сприяє кращому розумінню коду та спрощує подальше обслуговування програми.

Для реалізації бізнес-логіки ми використовуємо ASP.NET Core, сучасний крос-платформовий фреймворк, який забезпечує високу продуктивність та безпеку. Blazor Server App дозволяє нам створювати інтерактивні веб-інтерфейси з використанням C# замість JavaScript, що є значною перевагою для команди, яка вже добре знайома з C#. Це також дозволяє нам використовувати спільні ресурси та бібліотеки між серверною частиною та клієнтською частиною нашого застосунку, що сприяє більшій консистенції та зменшенню кількості помилок.

Застосування SQL Server як нашої системи управління базами даних гарантує надійність, безпеку та оптимізацію в обробці великих обсягів даних, що є критично важливим для нашого додатка. Entity Framework Core, у свою чергу, використовується для роботи з базою даних на високому рівні абстракції, що дозволяє нам ефективно виконувати CRUD операції без необхідності писати складний SQL код.

Інтеграція розроблених шарів у єдину систему дозволила нам створити стабільний, надійний та легко масштабований веб-застосунок. Весь процес від дизайну інтерфейсу до впровадження бізнес-логіки був спрямований на максимальне задоволення потреб кінцевих користувачів, забезпечуючи їм зручні та інтуїтивно зрозумілі інструменти для ефективної роботи з даними. Такий підхід не тільки спрощує введення нового програмного продукту в експлуатацію, але й значно підвищує ефективність подальшої його підтримки та розвитку.

ВИСНОВКИ

У ході дослідження було визначено, що для досягнення максимальної ефективності та інтеграції веб-застосунку на Blazor WebAssembly з існуючою IT-інфраструктурою компанії найкращим підходом є використання сучасних технологій і архітектурних рішень. Застосування багатошарової N-Layer архітектури забезпечило гнучкість та масштабованість, дозволивши чітко розділити логіку інтерфейсу користувача, бізнес-логіку та шар доступу до даних. Вибір ASP.NET Core для реалізації бізнес-логіки та Blazor Server App для створення інтерактивних веб-інтерфейсів на C# сприяв високій продуктивності, безпеці та консистенції коду, зменшуючи кількість помилок.

Для розробки було обрано Microsoft Visual Studio та Visual Studio Code, які завдяки глибокій інтеграції з ASP.NET та широким можливостям налагодження й управління проектами, забезпечили ефективну розробку та підтримку веб-додатків. Порівняння з альтернативними IDE, такими як Rider та WebStorm, показало, що інструменти від Microsoft краще інтегруються з обраними технологіями та забезпечують більшу зручність для розробників.

Вибір Microsoft SQL Server як системи управління базами даних, у поєднанні з Entity Framework Core, дозволив ефективно виконувати CRUD операції та забезпечив надійність і безпеку в обробці великих обсягів даних. Інтеграція між Visual Studio та SQL Server Management Studio спростила управління базами даних, підвищивши продуктивність розробників.

Загалом, використання ASP.NET Core, Blazor Server App, Visual Studio, Visual Studio Code та Microsoft SQL Server дозволило створити стабільний, надійний і легко масштабований веб-застосунок. Цей підхід забезпечив максимальну ефективність та задоволення потреб кінцевих користувачів, спрощуючи подальшу підтримку та розвиток системи, і підтвердив доцільність обраного технологічного стеку та архітектурних рішень для реалізації проекту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ASP.NET Core. URL: http://surl.li/uaxcr (дата звернення: 10.01.2024 р.).
2. Blazor Server App. URL: http://surl.li/uaxcu (дата звернення: 10.01.2024 р.).
3. SQL Server. URL: http://surl.li/uaxcx (дата звернення: 10.01.2024 р.).
4. Entity Framework Core. URL: http://surl.li/uaxda (дата звернення: 10.01.2024 р.).
5. Visual Studio - продукт Microsoft. URL: http://surl.li/uaxdb (дата звернення: 21.01.2024 р.).
6. Client-ServerArchitecture. URL: http://surl.li/uaxde (дата звернення: 01.02.2024 р.).
7. MudBlazor. URL: http://surl.li/uaxdi (дата звернення: 23.03.2024 р.).
8. Інтерфейс SAP Logon. URL: http://surl.li/uaxdk (дата звернення: 16.02.2024 р.).
9. Інтерфейс Oracle NetSuite. URL: http://surl.li/uaxdm (дата звернення: 16.02.2024 р.).
10. Інтерфейс Microsoft Dynamics 365. URL: http://surl.li/uaxdn (дата звернення: 16.02.2024 р.).
11. Інтерфейс Zoho Inventory. URL: http://surl.li/uaxdq (дата звернення: 16.02.2024 р.).
12. Інтерфейс QuickBooks. URL: http://surl.li/uaxds (дата звернення: 16.02.2024 р.).
13. Приклад N-Layer архітектури. URL: http://surl.li/uaxdt (дата звернення: 01.02.2024 р.).
14. ASP.NET Core Blazor Authentication and Authorization. URL: http://surl.li/uaxdv (дата звернення: 10.03.2024 р.).
15. ASP.NET Core Identity Overview. URL: http://surl.li/uaxdy (дата звернення: 20.03.2024 р.).
16. Інтерфейс 1С. URL: http://surl.li/uaxea (дата звернення: 16.02.2024 р.).
17. Вигляд програми Visual Studio. URL: http://surl.li/uaxee (дата звернення: 24.04.2024 р.).
18. Вигляд програми SSMS. URL: http://surl.li/uaxef (дата звернення: 24.04.2024 р.).

19. Вигляд програми Visual Studio Code. URL: http://surl.li/uaxeg (дата звернення: 24.04.2024 р.).
20. Вигляд програми Figma. URL: http://surl.li/uaxei (дата звернення: 24.04.2024 р.).
21. Вигляд програми Rider. URL: http://surl.li/uaxek (дата звернення: 24.04.2024 р.).
22. Вигляд програми WebStorm. URL: http://surl.li/uaxem (дата звернення: 24.04.2024 р.).
23. Посилання на готовий макет Soft UI. URL: http://surl.li/uaxeo дата звернення: 02.04.2024 р.).
24. Layered (N-layered). URL: http://surl.li/uaxer (дата звернення: 17.03.2024 р.).
25. JavaScript. URL: http://surl.li/uaxfz (дата звернення: 28.04.2024 р.).
26. Веб-застосунок для скорочення посилань. URL: http://surl.li/uaxhr (дата звернення: 24.05.2024 р.).
27. Веб-застосунок для скорочення посилань. URL: http://surl.li/uaxhr (дата звернення: 24.05.2024 р.).
28. Компанія для якої був задум створення веб-застосунку. URL:http://surl.li/uaxii (дата звернення: 25.05.2024 р.).

Додаток А

Служба ServiceBase

Лістинг програми

```
using IMS.WebApp.Interfaces;
namespace IMS.WebApp.Services
{
    public class ServiceBase<T> : ICrud<T> where T : class
    {
        readonly ApplicationDbContext context;
        protected IQueryable<T> Entities { get =>
context.Set<T>(); }
        public ServiceBase(ApplicationDbContext context)
        {
            this.context = context;
        }
        public bool Add(T entity)
        {
            try
            {
                context.Set<T>().Add(entity);
                return context.SaveChanges() != 0;
            }
            catch
            {
                return false;
            }
        }

        public bool Delete(T entity)
        {
            try
            {
                context.Set<T>().Remove(entity);
                return context.SaveChanges() != 0;
```

```
            }
            catch
            {
                return false;
            }
        }
        public List<T> GetAll()
        {
            return context.Set<T>().ToList();
        }
        public T GetById(int id)
        {
            return context.Set<T>().Find(id);
        }

        public bool Update(T entity)
        {
            try
            {
                context.Set<T>().Update(entity);
                return context.SaveChanges() != 0;

            }
            catch
            {
                return false;
            }
        }
    }
}
```

Додаток Б

DataContext

Лістинг програми

```
using IMS.Shared;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;
namespace IMS.WebApp.Data
{
    public class DataContext : IdentityDbContext
    {
        public DataContext(DbContextOptions options) :
base(options)
        {
        }
        public virtual DbSet<Tools> Tools { get; set; }
        public virtual DbSet<Inventory> Inventory { get; set; }
        public virtual DbSet<Recipient> Recipient { get; set; }
    }
}
```

Додаток В

Приклад сторінки додавання

Лістинг програми

```
@page "/add-inventory"
@using IMS.WebApp
@using IMS.Shared
@inject MudBlazor.ISnackbar snackBar
@inject IDialogService DialogService
@inject IJSRuntime JsRuntime
@inject NavigationManager NavigationManager
```

```
@inject InventoryService InventoryService
<div class="content-wrapper pb-0 text-center">
    <h4>Inventory</h4>
    <hr />
    <div style="justify-content:center" class="row">
        <div class="col-md-4">
            <form asp-action="Create">
                <div asp-validation-summary="ModelOnly"
class="text-danger"></div>
                <div class="form-group">
                    <label asp-for="Name"
class="control-label"></label>
                    <input asp-for="Name" class="form-control" />
                    <span asp-validation-for="Name"
class="text-danger"></span>
                </div>
                <div class="form-group">
                    <input type="submit" value="Create"
class="btn btn-primary" />
                </div>
            </form>
        </div>
    </div>

    <div>
        <a href="/">Back to List</a>
    </div>

</div>
```