

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**  
**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ВОДНОГО ГОСПОДАРСТВА**  
**ТА ПРИРОДОКОРИСТУВАННЯ**

**Навчально-науковий інститут автоматики, кібернетики та  
обчислювальної техніки**

"До захисту допущена"

Зав. кафедри комп'ютерних наук та  
прикладної математики

\_\_\_\_\_ 20\_\_ р.  
«\_\_» \_\_\_\_\_

**КВАЛІФІКАЦІЙНА РОБОТА**

**Настільний застосунок для фізичної особи-підприємця з  
використанням технологій MRP-2 систем**

Виконав: **Федун Назарій Олегович**

\_\_\_\_\_  
(прізвище, ім'я, по батькові)

(підпис)

група КН-41

Керівник: **доцент, к.п.н. Рощенюк А.М.**

\_\_\_\_\_  
(науковий ступінь, вчене звання, посада, прізвище, ініціали)



(підпис)

*Національний університет водного господарства та  
природокористування*

НН інститут **автоматики, кібернетики та обчислювальної техніки**  
Кафедра **комп'ютерних наук та прикладної математики** Освітньо-  
кваліфікаційний рівень **бакалавр**

Галузь знань 12 «Інформаційні технології»

Спеціальність 122 «Комп'ютерні науки»

**ЗАТВЕРДЖУЮ**  
**Завідувач кафедри**

“ \_\_\_\_\_ ” \_\_\_\_\_ 20\_\_ року

**З А В Д А Н Н Я**  
**НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ**

Федуна Назарія Олеговича

(прізвище, ім'я, по батькові)

1. Тема роботи Розробка настільного застосунку для фізичної особи  
підприємця з використанням MRP-2 систем

керівник роботи Рощенюк А.М., к. пед. н., доцент

( прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від "26" травня 2023  
року

№ 449

2. Строк подання студентом роботи 31.05.2023

3. Вихідні дані до роботи технології, бібліотеки та пакети для розробки  
настільного застосунку для фізичної особи-підприємця з використанням MRP-2  
систем.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які  
потрібно розробити) розробити настільний застосунок для фізичної особи-  
підприємця з використанням MRP-2 системи.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових  
креслень) мультимедійна презентація.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Розділ 1	доцент Роценюк А.М.	7.11.22	7.11.22
Розділ 2	доцент Роценюк А.М.	12.12.23	12.12.23
Розділ 3	доцент Роценюк А.М.	7.03.23	7.03.23

7. Дата видачі завдання 01.11.2022

### КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Вивчення літератури за обраної тематикою	01.11.2022 – 11.11.2022	виконав
2.	Аналіз предметної області	11.11.2022 – 16.12.2022	виконав
3.	Розробка алгоритму	16.12.2022 – 20.01.2023	виконав
4.	Проектування бази даних проєкта	20.01.2023 – 07.03.2023	виконав
5.	Розробка веб-серверу додатку	07.03.2023 – 7.04.2023	виконав
6.	Розробка алгоритму клієнтської частини додатку	07.04.2023 – 21.04.2023	виконав
7.	Налагодження фронтенд частини під настільний застосунок	21.04.2023 – 15.05.2023	виконав
8.	Підготовка звіту квал. роботи	15.05.2023 – 24.05.2023	виконав
9.	Підготовка до виступу	24.05.2023 - 30.05.2023	виконав

Студент

---



---

( підпис )

(прізвище та ініціали)



Керівник роботи

---



---

( підпис )

(прізвище та ініціали)

## ЗМІСТ

РЕФЕРАТ	5
ВСТУП	6
РОЗДІЛ 1. ДОСЛІДЖЕННЯ ПРОБЛЕМИ ЗМІСТУ ВИКОРИСТОВУВАНИХ ЗАСТОСУНКІВ З ВИКОРИСТАННЯМ ТЕХНОЛОГІЙ MRP-2 СИСТЕМ	8
1.1. Недоліки	8
1.2. Переваги даного проекту	12
1.3. MRP-2 система проекту	13
РОЗДІЛ 2. АНАЛІЗ СКЛАДОВИХ	16
2.1. База даних додатку	16
2.2. Створення серверної частини проекту	17
2.3. Створення бази даних проекту	21
2.4. Ініціалізація таблиць для БД	23
2.5. Написання логіки програми	25
2.6. Написання НТТР запитів серверу	28
2.7. Клієнтська частина проекту	31
2.8. Створення фронтенду для додатку	36
2.9. Десктопна частина проекту	44
2.10. Підключення клієнтської частини до десктопного варіанту	45
РОЗДІЛ 3. ПРАКТИЧНЕ ВИКОРИСТАННЯ ДОДАТКУ НА КЛІЄНСТЬКІЙ ЧАСТИНІ	47
3.1. Головне меню	47
3.2. Робота з клієнтами	48
3.3. Робота з виробниками та продуктами	52
3.4. Робота зі сховищем	54
3.5. Робота із замовленнями даного проекту	56
ВИСНОВКИ	59
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	61
ДОДАТКИ	62

## РЕФЕРАТ

Кваліфікаційна робота: 69 с., 35 рисунки, 3 додатки, 10 джерел.

**Метою кваліфікаційної роботи** є дослідження, створення та аналіз єдиного програмного середовища, яке повинно надати приватним особам-підприємцям можливість автоматизувати ряд бізнес-процесів для особистих бізнес-проектів.

**Об'єкт дослідження** – процес створення настільного застосунку для фізичної особи-підприємця з використанням MRP-2 систем.

**Предметом дослідження** – методи та засоби для роботи з настільним застосунком для фізичних осіб-підприємців з використанням MRP-2 систем за допомогою React для написання клієнтської частини, Electron для адаптування клієнтської частини у настільному застосунку та ASP.NET Core Web API для створення серверної частини та логіки проекту.

**Методи дослідження** – технології MRP-2 системи, React, ASP.NET Core Web API та Microsoft SQL Server Management Studio (SSMS).

Розробка настільного застосунку на базі технологій MRP-2 для фізичних осіб-підприємців має велику актуальність в сучасних умовах підприємницької діяльності. Вона надасть простий і зручний інструмент для планування виробничих потреб, оптимізації запасів та контролю виробництва.

**Ключові слова:** ДЕСКТОП, ДОДАТОК, TYPESCRIPT, NODEJS, .NET, БАЗА ДАНИХ.

## ВСТУП

У сучасному світі, де бізнес-середовище постійно змінюється і стає все більш конкурентним, фізичні особи-підприємці, які розпочали власний бізнес, потребують ефективних інструментів для управління своїми операціями. Одним з ключових аспектів успішної діяльності є ефективне планування та управління ресурсами, що включає в себе контроль запасів, виробництво, закупівлю та управління замовленнями.

У цьому контексті, розробка настільного застосунку для фізичної особи-підприємця з використанням технологій MRP-2 систем (Material Requirements Planning - 2) стає актуальною та значущою темою. MRP-2 система є високопродуктивним інструментом управління виробництвом та ресурсами, який дозволяє оптимізувати процеси планування, контролю запасів та виробництва.

Нами запропонована робота розробки настільного застосунку, який буде надавати фізичним особам-підприємцям зручні та потужні інструменти для ефективного планування та управління ресурсами в їх бізнесі. Використання технологій MRP-2 систем дозволить автоматизувати багато аспектів управління, забезпечуючи точність прогнозування, зменшення втрат та покращення продуктивності.

У роботі буде проведений аналіз та вибір технологій, описано архітектуру та функціонал розроблюваного застосунку, а також нами буде проведено його випробування та оцінка результатів. Очікується, що розроблений настільний застосунок стане корисним інструментом для фізичних осіб-підприємців, що дозволить їм ефективно керувати своїми ресурсами та покращити свою конкурентоспроможність на ринку.

Для виконання поставлених цілей, нами було сформульовано наступні завдання дослідження:

- дослідження схожих додатків та веб-сайтів;
- аналіз функціональності, дизайну та взаємодії з користувачами додатків, створених на основі технології MRP-2 систем;

- опрацювання ідеї, враховуючи виявлені особливості та потреби цільової аудиторії;
- здійснення розробки програми з використанням передових технологій;
- оцінка ефективності та функціональності створеного застосунку.

Практичне значення кваліфікаційної роботи полягає в створенні ефективного інструменту для фізичних осіб-підприємців, що дозволить спростити управління бізнесом та покращити його продуктивність. Цей настільний застосунок забезпечить оптимізацію виробничих процесів, планування виробничих потреб, керування запасами та контроль виробництва. Впровадження даного застосунку дозволить фізичним особам-підприємцям підвищити ефективність бізнесу, знизити ризики, покращити облік фінансових показників та зосередитися на стратегічному розвитку. Такий інструмент стане незамінним помічником у плануванні та прийнятті управлінських рішень, сприятиме зростанню конкурентоспроможності та забезпечить більш ефективне використання ресурсів.

# РОЗДІЛ 1. ДОСЛІДЖЕННЯ ПРОБЛЕМИ ЗМІСТУ ВИКОРИСТОВУВАНИХ ЗАСТОСУНКІВ З ВИКОРИСТАННЯМ ТЕХНОЛОГІЙ MRP-2 СИСТЕМ

## 1.1. Недоліки

За останні роки фізичні особи-підприємці (ФОПи) стали важливою складовою сектора малого та середнього бізнесів. Завдяки своїй гнучкості та широкому спектру діяльності, вони впливають на економічний розвиток та створення нових робочих місць. Однак, для успішного ведення свого бізнесу ФОПам потрібні ефективні інструменти, зокрема настільні застосунки, які враховують їхні специфічні потреби та допомагають керувати бізнес-процесами.

Розуміючи цю потребу ФОПів в інструментах для оптимізації своєї роботи, ми зосередилися на розробці рішення, яке задовольняло б їхні потреби та пропонувало ефективні функціональні можливості. Цей проект є результатом нашої пристрасної роботи та відданості наданню ФОПам інструменту, який сприятиме їхньому успіху та зростанню бізнесу.

Нами було проведено ретельний дослід потреб та вимог ФОПів, аналізуючи ринок та вивчаючи законодавство, пов'язане з приватними особами-підприємцями. На основі цих досліджень, нами було обрано розробляти саме даний проект.

У сучасному світі, де технології швидко розвиваються, наявність потужних та функціональних настільних додатків стає ключовим фактором для ефективного управління ФОПами. На сьогоднішній день існують деякі додатки, які призначені для фізичних осіб-підприємців, однак вони мають свої недоліки, такі як застарілий дизайн, незадовільний функціонал, помилки та відсутність підтримки. Ці обмеження призводять до незадоволення користувачів та ускладнюють їхню роботу.

Серед застосунків, які мають вищезазначені недоліки нами можуть бути визначені наступні:

- OutdatedFinances;



- BuggyBooks;
- InefficientManager;

Якщо ж детальніше розглядати програму "OutdatedFinances", можна зазначити, що вона має наступні конкретні недоліки, які варто проаналізувати, наприклад:

- застарілий інтерфейс: один з найбільших недоліків програми "OutdatedFinances" полягає в її застарілому та неінтуїтивному інтерфейсі. Користувачам може бути важко зорієнтуватися та виконувати потрібні дії. Недостатня ергономіка та складність взаємодії можуть призвести до зниження продуктивності та погіршення користувацького досвіду.
- обмежені функціональні можливості: "OutdatedFinances" може бути обмеженою у своїх функціональних можливостях порівняно з сучасними програмами управління фінансами. Недостатній набір функцій може ускладнити виконання деяких завдань, особливо для більш складних бізнес-потреб.
- відсутність інтеграції з іншими системами: програма "OutdatedFinances" може не мати можливостей для інтеграції з іншими бізнес-системами, такими як CRM (система управління взаємозв'язками з клієнтами) або електронна комерція. Це може створювати проблеми з обміном даними та виконанням автоматичних операцій.
- недостатня підтримка та оновлення: якщо програма "OutdatedFinances" не отримує регулярних оновлень і покращень, це може призвести до вразливостей безпеки, проблем з сумісністю зі сучасними операційними системами та іншими програмами. Недостатня підтримка може також вплинути на доступність допомоги та вирішення проблем.

- відсутність мобільної версії: у світі зростаючої мобільності та використання смартфонів та планшетів, відсутність мобільної версії програми "OutdatedFinances" може бути значним недоліком. Користувачам може бути не зручно та обмежено використовувати програму лише на стаціонарному комп'ютері.

Ураховуючи ці недоліки, розробка та використання сучасних програм управління фінансами з урахуванням попередніх зазначених аспектів може сприяти покращенню продуктивності, користувацького досвіду та ефективності управління фінансами для фізичних осіб-підприємців.

Програма "BuggyBooks" також має деякі недоліки, до прикладу:

- нестабільність та помилки: один з найбільших недоліків програми "BuggyBooks" полягає у її нестабільності та постійній появі помилок. Це може призводити до непередбачуваних збоїв та втрати даних. Нестабільна робота програми може значно підірвати довіру користувачів та призвести до негативного враження від використання.
- відсутність важливих функцій: "BuggyBooks" може бути обмеженою у своїх функціональних можливостях порівняно з конкурентними програмами управління книжковою бібліотекою. Відсутність ключових функцій, таких як ефективне каталогізування книг, пошук за авторами або жанрами, може ускладнити організацію та пошук книжкових матеріалів.
- поганий інтерфейс та користувацький досвід: "BuggyBooks" може мати неінтуїтивний та незручний інтерфейс, що робить його важким у використанні для користувачів. Відсутність логічної структури та навігаційних можливостей може призводити до заплутання та погіршення загального користувацького досвіду.
- обмежена підтримка та оновлення: якщо програма "BuggyBooks" не отримує регулярних оновлень та покращень, це може призвести до накопичення вразливостей безпеки та проблем зі сумісністю зі

сучасними операційними системами. Недостатня підтримка може також вплинути на вирішення проблем та надання допомоги користувачам.

- відсутність мобільної версії: у світі, де мобільні пристрої стають все більш популярними, відсутність мобільної версії програми "BuggyBooks" може бути значним недоліком. Користувачам може бути незручно та обмежено використовувати програму лише на стаціонарних комп'ютерах.

Ураховуючи нами дані недоліки, розробка та використання сучасних програм управління книжковою бібліотекою з урахуванням попередніх зазначених аспектів може сприяти покращенню стабільності, функціональності та користувацького досвіду для користувачів "BuggyBooks".

У свою чергу програма "InefficientManager" має наступні недоліки:

- Недостатня ефективність управління: Один з головних недоліків програми "InefficientManager" полягає в її недостатній ефективності управління. Програма може не забезпечувати достатніх інструментів та функціональності для ефективного планування, організації та моніторингу проектів та завдань. Це може призводити до збільшення витрат часу та зусиль для досягнення бізнес-цілей.
- Відсутність аналітичних можливостей: "InefficientManager" може бути обмеженою у своїх аналітичних можливостях. Відсутність інструментів для збору та аналізу даних може ускладнити прийняття управлінських рішень на основі об'єктивної інформації. Відсутність графіків, діаграм та інших засобів візуалізації може ускладнити розуміння стану проектів та завдань.
- Поганий дизайн інтерфейсу: "InefficientManager" може мати неінтуїтивний та заплутаний дизайн інтерфейсу. Неструктурований розміщення елементів, погана взаємодія та незрозумілі символи можуть призводити до збентеження користувачів та ускладнювати використання програми. Недостатня увага до користувацького

досвіду може призвести до незадоволення та відмови від використання програми.

- Недостатня підтримка та оновлення: Якщо програма "InefficientManager" не отримує регулярних оновлень та покращень, це може призвести до виникнення проблем безпеки, несумісності зі сучасними операційними системами та іншими програмними продуктами. Недостатня підтримка може також ускладнити вирішення проблем та отримання необхідної допомоги з боку розробників.
- Відсутність мобільної версії: У сучасному світі мобільні пристрої стають все більш популярними, тому відсутність мобільної версії програми "InefficientManager" може бути значним недоліком. Користувачам може бути не зручно та обмежено використовувати програму лише на стаціонарних комп'ютерах, що обмежує їхню доступність та зручність використання.

Ураховуючи нами данні недоліки, розробка та використання ефективних програм управління з урахуванням попередніх зазначених аспектів може сприяти покращенню продуктивності, управління проектами та користувацького досвіду для бізнес-користувачів.

## **1.2. Переваги даного проекту**

Враховані нами недоліки раніше згаданих додатків, розроблений настільний застосунок для ФОПів на базі технологій MRP-2 систем має кілька вагомих переваг.

- Сучасний та естетичний дизайн: застосунок має сучасний та привабливий інтерфейс, який забезпечує комфортну та зручну роботу з програмою. Він відповідає останнім трендам у дизайні і забезпечує приємний користувацький досвід.
- Розширений функціонал: додаток надає більш широкий спектр функцій, необхідних для ефективного управління бізнес-процесами

ФОПів. Він включає в себе модулі для фінансового обліку, складського управління, замовлень та управління проектами, що забезпечує комплексний підхід до керування бізнесом.

- Надійність та підтримка: програма пройшла ретельне тестування нами та має низький рівень помилок. Крім того, надається постійна підтримка користувачів та регулярні оновлення, що гарантує безперебійну роботу та вирішення можливих проблем.

Розроблений нами настільний застосунок для ФОПів на базі технологій MRP-2 системи має кілька вагомих переваг. Він відповідає поточним вимогам і потребам ФОПів, надаючи сучасний та естетичний дизайн, розширений функціонал та надійність. Додаток забезпечує комплексний підхід до управління бізнес-процесами, включаючи фінансовий облік, складське управління, замовлення та управління проектами. Крім того, надається постійна підтримка користувачів та регулярні оновлення, що гарантує безперебійну роботу та вирішення можливих проблем. У висновку, розроблений мною настільний застосунок для ФОПів на базі технологій MRP-2 систем перевершує існуючі погані додатки за своїми функціональними можливостями, сучасним дизайном та надійністю. Він надає ФОПам ефективний інструмент для успішного управління їхнім бізнесом та допомагає досягти кращих результатів.

### **1.3. MRP-2 система проекту**

Для продуктивної роботи нами була використана MRP-2 система (Manufacturing Resource Planning 2). Це розширена версія системи планування ресурсів виробництва, яка використовується в операційному менеджменті та виробничому плануванні. MRP-2 включає в себе не тільки планування матеріалів, але й інші важливі аспекти виробництва, такі як планування потужностей, планування робочої сили, планування фінансових ресурсів тощо.

Основна мета MRP-2 полягає в тому, щоб забезпечити оптимальне використання ресурсів виробництва, зменшити запаси, знизити час виконання

замовлень та покращити ефективність виробничих процесів. Вона дозволяє компаніям більш точно прогнозувати потреби в матеріалах, розподіляти робочу силу, планувати виробничі операції та контролювати виконання планів.

MRP-2 система використовує комп'ютеризовану систему управління, що базується на зборі та аналізі даних про матеріали, обладнання, робочу силу, фінанси та інші ресурси компанії. За допомогою MRP-2 можна планувати виробництво на основі замовлень, прогнозувати потреби в матеріалах, визначити потрібну кількість робочої сили та знаходити оптимальний графік виробництва.

Основні переваги MRP-2 системи включають автоматизацію та оптимізацію виробничого планування, зниження запасів, покращення доставки та виконання замовлень, збільшення ефективності виробництва та зниження витрат. Вона дозволяє компаніям більш ефективно керувати ресурсами та підвищувати конкурентоспроможність на ринку.

У нашому проєкті MRP-2 система може бути корисною для оптимізації виробничих процесів та керування ресурсами під час розробки проєкту:

- планування виробництва;
- управління запасами;
- планування використання ресурсів.

Дана робота була виконана нами у програмі «ProjectLibre». Це безкоштовне відкрите програмне забезпечення для управління проєктами, яке надає інструменти для планування, розкладування та керування проєктами. Воно є альтернативою комерційним програмам, таким як Microsoft Project, і надає можливості для створення графіків, ресурсного управління, відстеження прогресу проєкту, аналізу ризиків та багато іншого. ProjectLibre підтримує різні формати файлів, що дозволяє обмінюватися даними з іншими програмами для управління проєктами. Він є потужним інструментом для професіоналів у галузі проєктного управління та дозволяє ефективно планувати та керувати проєктами будь-якого розміру і складності.

Перед початком самого додатку нами було розблене поетапне планування розробки доного проекту, розписуючи дати та час початку та завершення завдань, хто виконує певну задачу, та які матеріали потрібні були для цього. У ProjectLibre це все можливо (див. рис. 1.1).

	Name	Duration	Start	Finish	Predecessors	Resource Names
1	Серверна частина	19 days?	4/15/23 8:00 AM	5/11/23 5:00 PM		
2	Interface частина	4 days?	4/15/23 8:00 AM	4/20/23 5:00 PM		
3	Підключення БД	1 day?	4/15/23 8:00 AM	4/17/23 5:00 PM		Backend-Developer
4	створення моделей	1 day?	4/18/23 8:00 AM	4/18/23 5:00 PM	3	Backend-Developer
5	створення конфігурацій для моделей	2 days?	4/18/23 8:00 AM	4/19/23 5:00 PM	3	Backend-Developer
6	створення репозиторію для взаємодії з БД	1 day?	4/20/23 8:00 AM	4/20/23 5:00 PM	5	Backend-Developer
7	Core частина	13 days?	4/21/23 8:00 AM	5/9/23 5:00 PM	2;6	
8	Створення DTOs	1 day?	4/21/23 8:00 AM	4/21/23 5:00 PM	6	Backend-Developer
9	Створення інтерфейсів для сервісів	1 day?	4/24/23 8:00 AM	4/24/23 5:00 PM	8	Backend-Developer
10	Створення сервісів	5 days?	4/24/23 8:00 AM	4/28/23 5:00 PM	8	Backend-Developer
11	Створення automapping	1 day?	5/1/23 8:00 AM	5/1/23 5:00 PM	9;10	Backend-Developer
12	створення валідації	1 day?	5/2/23 8:00 AM	5/2/23 5:00 PM	11	Backend-Developer
13	прописання функціоналу	5 days?	5/3/23 8:00 AM	5/9/23 5:00 PM	11;12	Backend-Developer
14	API частина	2 days?	5/10/23 8:00 AM	5/11/23 5:00 PM	7;13	
15	Створення контролерів	1 day?	5/10/23 8:00 AM	5/10/23 5:00 PM	13	Backend-Developer
16	Створення http запитів	1 day?	5/11/23 8:00 AM	5/11/23 5:00 PM	15	Backend-Developer
17	Дизайн	3 days?	5/12/23 8:00 AM	5/16/23 5:00 PM	1	
18	Створення нативного дизайну для фронту	3 days?	5/12/23 8:00 AM	5/16/23 5:00 PM	1	Designer
19	Клієнтська частина	12 days?	5/17/23 8:00 AM	6/1/23 5:00 PM	17;1	
20	підключення пакетів	1 day?	5/17/23 8:00 AM	5/17/23 5:00 PM	18	Frontend-Developer
21	створення головної сторінки	2 days?	5/18/23 8:00 AM	5/19/23 5:00 PM	20	Frontend-Developer
22	Створення інших сторінок для перегляду даних	1 day?	5/22/23 8:00 AM	5/22/23 5:00 PM	21	Frontend-Developer
23	створення маршрутизації	2 days?	5/22/23 8:00 AM	5/23/23 5:00 PM	21	Frontend-Developer
24	Створення сторінок для створення нових даних	2 days?	5/24/23 8:00 AM	5/25/23 5:00 PM	23;22	Frontend-Developer
25	Створення сторінок для редагування	2 days?	5/26/23 8:00 AM	5/29/23 5:00 PM	24	Frontend-Developer
26	Створення видалення даних з БД	1 day?	5/30/23 8:00 AM	5/30/23 5:00 PM	25	Frontend-Developer
27	Створення валідації даних	2 days?	5/31/23 8:00 AM	6/1/23 5:00 PM	26	Frontend-Developer
28	Тестування та виправлення помилок	12 days?	6/2/23 8:00 AM	6/19/23 5:00 PM	1;17;19	Frontend-Developer;Backen...

Рис. 1.1. Список планування проекту

Після детального процесу планування, передбаченої в ході дослідження, наведеного вище, реалізовано автоматичне створення тижневої діаграми, яка відображає послідовність подій та завдань, необхідних для успішного виконання проекту (див. Рис. 1.2).



Рис. 1.2. Тижнева діаграма планування

## РОЗДІЛ 2. АНАЛІЗ СКЛАДОВИХ

### 2.1. База даних додатку

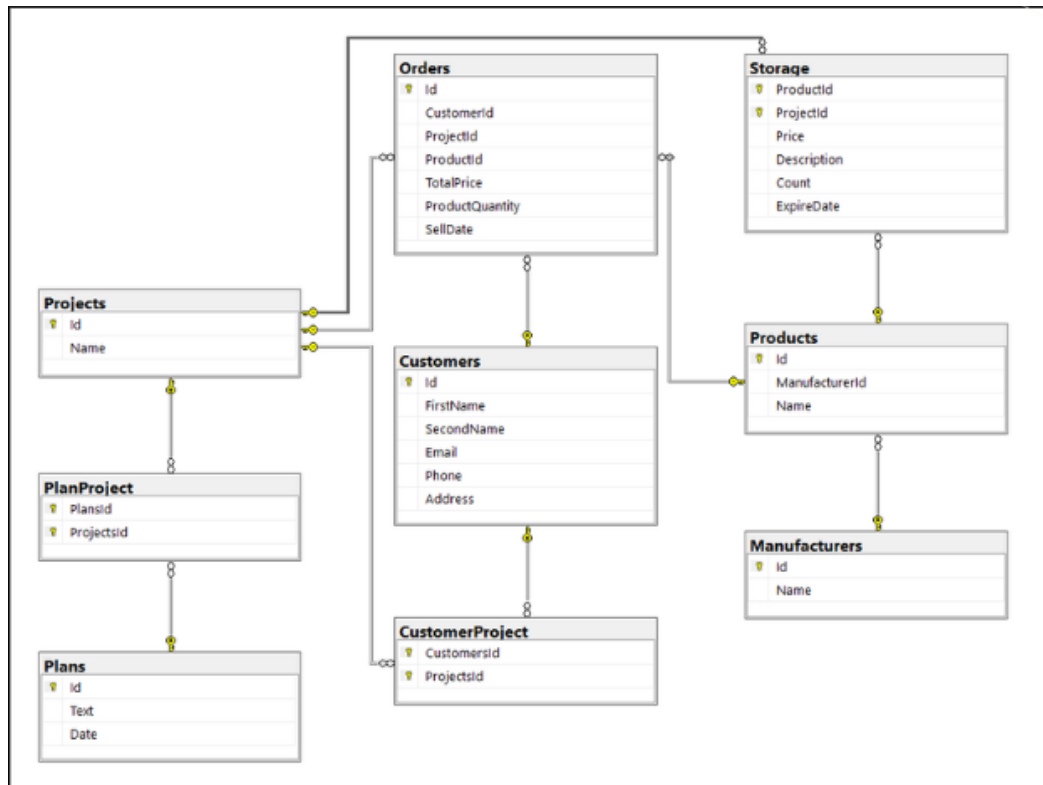
Робота з базою даних виконувалась нами у програмі «Microsoft SQL Server Management Studio» (SSMS). Вона є інтегрованим середовищем для управління та розробки баз даних на Microsoft SQL Server. Дана програма забезпечує зручний графічний інтерфейс, який дозволяє керувати базами даних, виконувати SQL-запити, налаштовувати сервер і моніторити його стан. SSMS також надає широкий спектр інструментів для розробки баз даних, включаючи можливість створювати таблиці, процедури, функції та забезпечувати безпеку даних. Завдяки своїм функціональним можливостям, SSMS є важливим інструментом для роботи з Microsoft SQL Server і допомагає забезпечити ефективне управління базами даних та розробку відповідних рішень.

У ході ретельного дослідження теми кваліфікаційної роботи нами був доканий важливий висновок, що створення детальної та комплексної бази даних є необхідним кроком для максимальної оптимізації робочих процесів та збереження важливої інформації. На основі цього висновку, нами була зосереджена розробка низки таблиць бази даних, які взаємодіють між собою за допомогою спільних ключів та різних типів зв'язків, таких як:

- один до багатьох;
- багато до одного;
- один до одного.

Даний підхід дозволив нам створити потужну та гнучку структуру бази даних, яка забезпечила нам ефективну обробку за збереження інформації, сприяючи швидкому доступу до даних та покращенню робочих процесів (див. рис. 2.1).

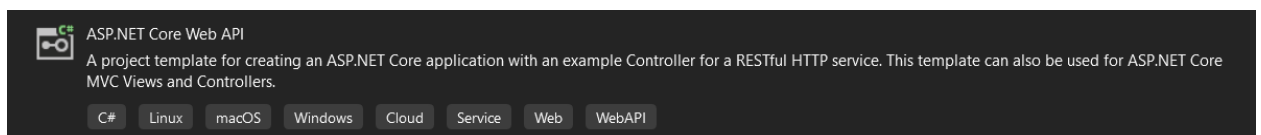




**Рис. 2.1.** Діаграма бази даних проєкту

## 2.2. Створення серверної частини проєкту

У ході роботи нами була створена логіка, БД та таблиці до бази даних у **ASP.NET Core Web API** [6] (див. рис. 2.2) у програмі **Visual Studio 2022**.



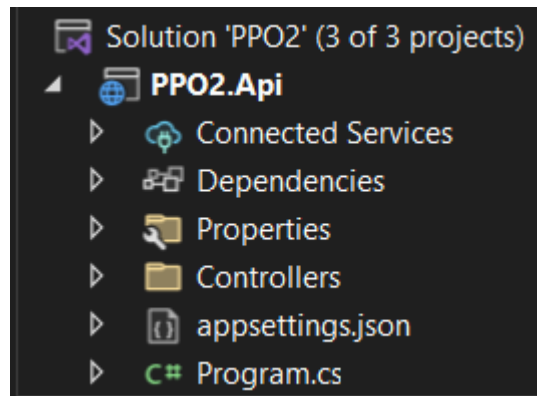
**Рис. 2.2.** Проєкт серверної частини

Бекенд частину проєкту нами було розділено на три основні частини для структуризації коду:

1. Основна або Центральна частина (*Core*) – в ній описується робота з моделями, сервісами, валідація та перетворення даних.
2. Інфраструктурна частина (*Infrastructure*) – робота безпосередньо з базою даних: створення її, ініціалізація, міграції ітд.

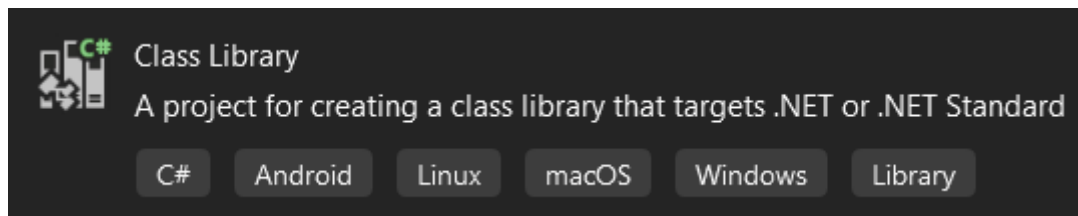
3. Частина програмного інтерфейсу (*API*) – стартап проект, який буде запускатись нами при старті програми. У ньому створюються контролери, різноманітні підключення до сервісів і завантаження додаткових пакетів.

Після створення нами нашого проекту, ми маємо наш стартап підпроект, тобто API частина (див. рис. 2.3).



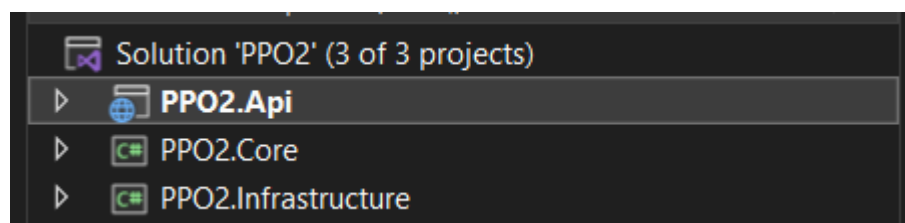
**Рис. 2.3. Початковий вигляд проекту після створення**

Для створення інших підпроектів нами використовувався ClassLibrary проект (див. рис. 2.3).



**Рис. 2.4. ClassLibrary проект**

Після створення нами усіх підпроектів – кінцевий вигляд нашої програми буде, як показано на рис. 2.5.



**Рис. 2.5. Кінцевий вигляд проектів серверної частини**

Для того, щоб реалізувати проект повноцінно потрібно було завантижити ряд NuGet пакетів, щоб використовувати певний функціонал для різних цілей.

Список пакетів:

- Microsoft.EntityFrameworkCore – забезпечує доступ та взаємодію з базою даних з використанням ORM (Object-Relational Mapping) підходу.
- Microsoft.EntityFrameworkCore.Tools – допомагає зручно працювати з Entity Framework Core, спрощуючи процес розробки, міграції та керування базами даних.
- Microsoft.EntityFrameworkCore.Design – надає підтримку для створення та редагування моделей даних, виконання міграцій та генерації коду. Приклад використання даного пакету: *Update-Database; Add Migration "migration\_text"*
- Microsoft.EntityFrameworkCore.SqlServer – надає підтримку для використання СУБД Microsoft SQL Server з фреймворком Entity Framework Core. Цей пакет дозволяє створювати та взаємодіяти з базами даних SQL Server з використанням ORM (Object-Relational Mapping) підходу.
- Ardalis.Specification – надає інструменти для розробки і застосування специфікацій в програмних проектах. Специфікації є шаблонами або правилами, за допомогою яких можна визначити, які об'єкти відповідають певним критеріям або умовам. Цей пакет дозволяє розробникам легко створювати, комбінувати та використовувати специфікації для фільтрації та сортування даних в додатках. Він спрощує реалізацію складних запитів до даних, полегшує їх тестування та підтримку, і допомагає підтримувати чистоту коду та розділення відповідальностей.
- AutoMapper [1] – інструмент для автоматичного відображення (mapping) даних між об'єктами різних типів. Він дозволяє автоматизувати процес перенесення даних з одного об'єкта в інший,

забезпечуючи зручний та ефективний спосіб копіювання значень властивостей. AutoMapper використовується для скорочення часу та зусиль, необхідних для вручного написання коду копіювання даних, зменшення можливості помилок та полегшення розробки програм, які працюють зі складними моделями даних. Приклад використання маппінгу: *CreateMap<Project, ProjectCreateDto>().ReverseMap();*

- FluentValidation [3] – надає засоби для валідації даних в програмних проєктах. Він дозволяє легко визначати правила валідації для моделей даних і забезпечує гнучкість та розширюваність при валідації. За допомогою FluentValidation можна встановлювати правила перевірки на основі типів даних, використовувати вбудовані та користувацькі валідатори, проводити перевірку на рівні властивостей, а також на рівні об'єктів і зв'язків між ними. Цей пакет спрощує процес валідації даних і допомагає забезпечити їхню цілісність та правильність. Приклад використання валідації: *RuleFor(c => c.Name).NotEmpty().MaximumLength(maximumLength: 55);*
- FluentValidation.AspNetCore [3] – надає підтримку для інтеграції фреймворка FluentValidation з ASP.NET Core. FluentValidation дозволяє зручно валідувати дані та моделі у веб-додатках. Цей пакет розширює функціональність ASP.NET Core, дозволяючи використовувати FluentValidation для валідації вхідних даних, переданих у запитах, а також для валідації моделей перед їх збереженням. Використання FluentValidation.AspNetCore спрощує реалізацію валідації в ASP.NET Core додатках допомагає забезпечити правильність та надійність вхідних даних.

### 2.3. Створення бази даних проєкту

Для створення бази даних нами було завантажено чотири NuGet пакети:

1. Microsoft.EntityFrameworkCore;
2. Microsoft.EntityFrameworkCore.Tools;
3. Microsoft.EntityFrameworkCore.Design;
4. Microsoft.EntityFrameworkCore.SqlServer

Нами було створено клас «DataContext» у папці «Data», який відповідає за підключення до самої БД.

Даний клас успадковує Entity Framework Core об'єкт «DbContext» та два конструктури для валідного запуску об'єкту.

```
public class DataContext : DbContext
{
    public DataContext() : base() { }
    public DataContext(DbContextOptions<DataContext> options) :
base(options) {}
}
```

Для того, щоб створити базу даних, нами був створений рядок підключення до неї. Для цього у API частині у appsettings.json файлі нами був прописаний спеціальний рядок.

```
ConnectionStrings": {
    "DefaultConnection":
"Server=.;Database=PP02;Trusted_Connection=true;Integrated
Security=True;MultipleActiveResultSets=True;TrustServerCertificate=true;"
}
```

У Infrastructure частині нами було створено ServiceExtension.cs файл, в якому нами були зроблені різні підключення певного підпроєкту. Був створений статичний клас з функцією «AddDbContext», яка відповідає за підключення до БД за допомогою рядка, який прописаний у appsettings.json.

```

public static class ServiceExtensions
{
    public static void AddDbContext(this IServiceCollection services, string
connectionString)
    {
        services.AddDbContext<DataContext>(options=>options.UseSqlServer(connectio
nString));
    }
}

```

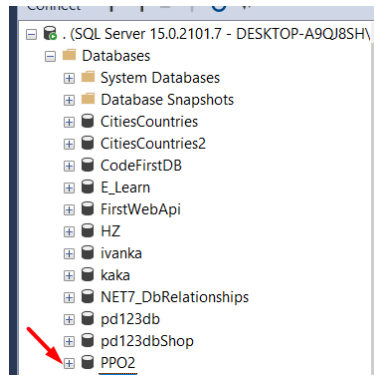
Дана функція нами викликається у API частині у файлі Program.cs. Для того, щоб доступитись до функціоналу іншого проекту, нами було підключено Infrastructure підпроект до API підпроекту натиснувши на нього ПКМ → Add → Project Reference та ставимо галочку на Infrastructure проект. Після цього у файлі Program.cs нами було підключено нашу базу даних.

```

stringconnectionString=builder.Configuration.GetConnectionString("Default
Connection"); // get connection string from appsettings.json
builder.Services.AddDbContext(connectionString); // connect to our DB

```

Далі в Package Manager Console нами було обрано наш Infrastructure проект у Default Project та пишемо команду «Add-Migration». Після чого успішно була створена міграція створення нашої БД. Створена папка «Migrations» у проекті Infrastructure, у якій в подальшому будуть зберігатись усі створені міграції. Щоб прийняти нашу міграцію і занести зміни – нами була прописана команда «Update-Database» після чого створиться наша база даних на локальному пристрої (див. рис. 2.6).



**Рис. 2.6. БД на локальному пристрої**

## 2.4. Ініціалізація таблиць для БД

Для створення таблиць у базі даних, нами були створені Entity моделі. Це класи, які відображають наші таблиці у базі даних. Отже, нами була створена папка «Entities» у якій зберігаються усі моделі.

```
[Table("Projects")]
```

```
public class Project : IEntity
```

```
{
```

```
    public int Id { get; set; }
```

```
    [StringLength(maximumLength: 50)]
```

```
    public string Name { get; set; } = string.Empty;
```

```
    //[JsonIgnore]
```

```
    public List<Customer> Customers { get; set; } = new List<Customer>();
```

```
    [JsonIgnore]
```

```
    public List<Storage> Storage { get; set; } = new List<Storage>();
```

```
    //[JsonIgnore]
```

```
    public List<Plan> Plans { get; set; } = new List<Plan>();
```

```
    //[JsonIgnore]
```

```
    public List<Order> Orders { get; set; } = new List<Order>();
```

```
}
```

Наші моделі були підключені нами до «DataContext.cs».

```
public DbSet<Project> Projects { get; set; }
```

```
public DbSet<Customer> Customers { get; set; }
```

```

public DbSet<Manufacturer> Manufacturers { get; set; }
public DbSet<Product> Products { get; set; }
public DbSet<Storage> Storage { get; set; }
public DbSet<Order> Orders { get; set; }

```

Після створення моделей, нами були створені додаткові правила для них. У Infrastructure частині нами було створено папку «Configurations», а в ній вже безпосередньо клас «ProjectConfiguration.cs» в якому будуть додаткові правила для таблиці з проектами.

```

internal class ProjectConfiguration : IEntityConfiguration<Project>
{
    public void Configure(EntityTypeBuilder<Project> builder)
    {
        builder.HasIndex(p => p.Name).IsUnique();
    }
}

```

Щоб підключити додаткові конфігурації для моделей – нами було їх підключено до «DataContext» у захищеній переписаній функції «OnModelCreating», яка отримується при наслідуванні «DbContext».

```

protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    base.OnModelCreating(modelBuilder);
    modelBuilder.ApplyConfiguration(new ProjectConfiguration());
    modelBuilder.ApplyConfiguration(new CustomerConfiguration());
    modelBuilder.ApplyConfiguration(new ManufacturerConfiguration());
    modelBuilder.ApplyConfiguration(new ProductConfiguration());
}

```



Після підключення бази даних, створення моделей та їх конфігурацій, нами була успішно створена база даних на локальному пристрої з використанням міграції, щоб занести дані і БД.

## 2.5. Написання логіки програми

Функціонал і логіка програми нами була написана у «Core» частині. Для початку нами були створені DTO (Data Transfer Object). Це патерн проектування, який використовується для передачі даних між компонентами або системами. DTO є об'єктом, який містить дані із джерела (наприклад, бази даних) та передає їх іншим компонентам або системам для виконання певних операцій.

Нами була створена папка «DTOs» → «ProjectDto», а в ній клас «ProjectCreateDto.cs». У даному класі нами були створені усі необхідні поля, які мусять бути надіслані у базу даних при створенні проекту.

```
public class ProjectCreateDto
{
    public string Name { get; set; } = string.Empty;
}
```

Після створення об'єкта в якому будуть зберігатись дані – нами був написаний функціонал для відправки цих даних у саму БД безпосередньо. Для цього нами була створена папка «Interfaces» а в ній безпосередньо інтерфейс під назвою «IProjectService.cs» в якому нами було написано оголошення методів, які будуть прописані в нашому сервісі.

```
public class ProjectCreateDto
{
    public string Name { get; set; } = string.Empty;
}

public interface IProjectService
{
```

```

    Task<ServiceResponse> GetAllAsync();
    Task<ServiceResponse> CreateAsync(ProjectCreateDto projectDto);
    Task<ServiceResponse> UpdateAsync(ProjectUpdateDto projectDto, int id);
    Task<ServiceResponse> DeleteAsync(int id);
    Task<ServiceResponse> GetById(int id);
}

```

Далі нами було створено нову папку «Services» у Core частині, у якій знаходяться усі сервіси з унаслідуванням певних інтерфесій для реалізації їх методів. У подальшій роботі нами був створени клас «ProjectService.cs» у якому прописано логіку додавання нового проекту, видалення, оновлення, пошук ітд.

```

    public async Task<ServiceResponse> CreateAsync(ProjectCreateDto
projectDto)
    {
        var mappedProject = _mapper.Map<Project>(projectDto);
        try
        {
            await _projectRepo.Insert(mappedProject);
            await _projectRepo.Save();
            return new ServiceResponse
            {
                Success = true,
                Message = "Project has been created successfully",
                Payload = mappedProject
            };
        }
        catch (Exception ex)
        {
            return new ServiceResponse

```

```

        {
            Success = false,
            Message = ex.InnerException.ToString()
        };
    }
}

```

У даному методі виконується логіка створення нового проекту. Нами використовується перетворення об'єкту типу `ProjectCreateDto` у `Project` тип, щоб відправити коректний тип даних у БД. Для цього нами було створено папку «AutoMapper» а в ній клас «AutoMapperProjectProfile.cs» в якому нами було прописано який тип класу може перетворюватись на інший тип. Це значно скорочує час написання коду та оптимізовує його.

```

public class AutoMapperProjectProfile : Profile
{
    public AutoMapperProjectProfile()
    {
        CreateMap<Project, ProjectCreateDto>().ReverseMap();
        CreateMap<Project, ProjectUpdateDto>().ReverseMap();
        CreateMap<Project, ProjectDto>().ReverseMap();
    }
}

```

Також у методі нами повертається «ServiceResponse». Це окремий особисто створений клас, щоб відправляти дані як відповідь сервера.

```

public class ServiceResponse
{
    public bool Success { get; set; } // result of operation
    public string Message { get; set; } = string.Empty; // message of the result
    public object? Payload { get; set; } // received result's object
}

```

```

        public IEnumerable<string>? Errors { get; set; } // list of errors if the
result is failed
    }

```

## 2.6. Написання HTTP запитів серверу

У API частині нами було створено контролер у папці «Controllers» «ProjectController.cs». У ньому нами було створено http запит post для створення проекту, put для оновлення даних проекту, delete видалення проекту за унікальним ключом, get отримання списку усіх проектів у базі даних.

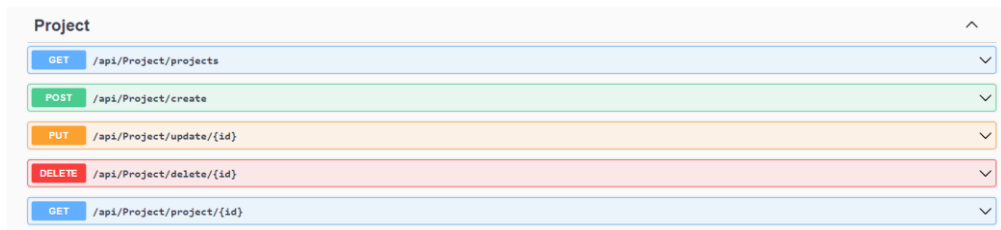
```

    [HttpPost("create")]
    public async Task<IActionResult> Create([FromBody]
ProjectCreateDto request)
    {
        var validation = new ProjectCreateValidation();
        var validationResult = await validation.ValidateAsync(request);
        if (validationResult.IsValid)
        {
            var result = await _projectService.CreateAsync(request);
            return Ok(result);
        }
        return BadRequest(validationResult.Errors);
    }

```

Після ініціювання проекту, відбувається автоматичне відкриття стандартного Swagger-інтерфейсу веб-браузера, який забезпечує доступ до важливої інформації щодо API-запитів. Цей Swagger-інтерфейс включає повний перелік доступних запитів, їх URL-адреси та параметри, які необхідно ввести для успішного виконання запитів на цьому сервері. Цей інтерфейс надає можливість виконувати різні запити до сервера та взаємодіяти з ним шляхом введення відповідних даних (див. рис. 2.7).

Слід зазначити, що Swagger-інтерфейс забезпечує зручне та структуроване представлення можливостей сервера, надаючи користувачам ясне уявлення про доступні запити та їх параметри. Це сприяє зручній взаємодії з API-сервером та дозволяє здійснювати потрібні операції з використанням наявних ресурсів. У цілому, використання Swagger-інтерфейсу підвищує зручність та ефективність роботи з API-сервером, дозволяючи здійснювати різноманітні дії та маніпуляції з даними.

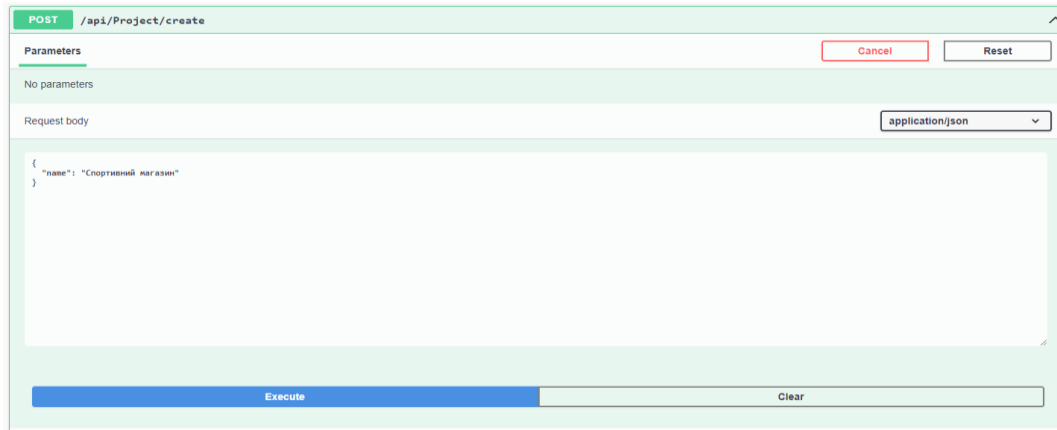


**Рис. 2.7. Вигляд HTTP запитів у Swagger**

Продемонструємо здійснення відправки запиту на створення проекту спортивного магазину який може бути переглянутий на рис. 2.8. Даний запит передбачає передачу відповідних даних, які ідентифікують цей проект та надають йому назву «Спортивний магазин». Запит включає необхідну інформацію про параметри проекту, включаючи опис, цілі, обсяг робіт та інші характеристики, що є важливими для коректного створення та ідентифікації проекту.

Цей процес передбачає використання відповідного інтерфейсу або системи керування проектами, яка забезпечує можливість відправки запиту та обробки отриманих даних. Запит на створення проекту «Спортивний магазин» має на меті ініціювати процес створення проекту згідно з вказаними характеристиками та вимогами.

Такий запит є важливою частиною процесу управління проектами, який дозволяє ідентифікувати та створювати проекти з розробкою специфікацій та постановкою завдань. Запит на створення проекту «Спортивний магазин» виконує функцію початкової точки для подальшого планування та виконання робіт, пов'язаних з реалізацією даного проекту.

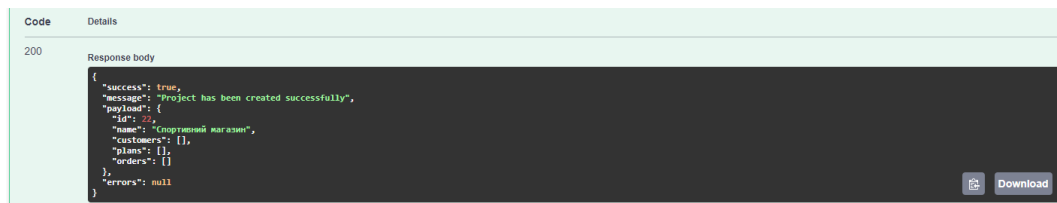


**Рис. 2.8. Відправлення даних на сервер**

За умови відсутності відповідного запису про даний проект у базі даних нами передбачається отримання позитивного результату (див. рис. 2.9). Це означає, що відсутність проекту у базі даних є бажаною умовою, яка сприяє досягненню очікуваного результату.

Протилежно, у випадку наявності відповідного запису про даний проект у базі даних нами передбачається отримання негативного результату (див. рис. 2.10). Цей негативний результат свідчить про несумісність чи небажаність наявності запису про даний проект у базі даних, що відображається у контексті виведеного результату.

У HTTP запитах також використовується валідація даних з використання пакету `FluentValidation`, значення якого було описано вище. Якщо вміст даних не підлягає валідації, то ніякого запиту до бази даних не буде.



**Рис. 2.9. Позитивний результат відправки даних у БД**

```

Code    Details
200     Response body
{
  "success": false,
  "message": "Microsoft.Data.SqlClient.SqlException (0x80131904): Cannot insert duplicate key row in object 'dbo.Projects' with unique index 'IX_Projects_Name'. The duplicate key value is (noop
  ний магазин).\r\n at Microsoft.Data.SqlClient.SqlConnection.OnError(SqlException exception, Boolean breakConnection, Action`1 wrapCloseInAction)\r\n at Microsoft.Data.SqlClient.SqlInternal
  Connection.OnError(SqlException exception, Boolean breakConnection, Action`1 wrapCloseInAction)\r\n at Microsoft.Data.SqlClient.TdsParser.ThrowExceptionAndWarning(TdsParserStateObject stateOb
  j, Boolean callerHasConnectionLock, Boolean asyncClose)\r\n at Microsoft.Data.SqlClient.IdfParser.TryRun(RunBehavior runBehavior, SqlCommand cmdHandler, SqlDataReader dataStream, BulkCopyImpl
  chandler, BulkCopyCommander, TdsParserStateObject stateObj, Boolean dataReady)\r\n at Microsoft.Data.SqlClient.SqlDataReader.TryReadInternal(Boolean ignoreBuffer)\r\n at Microsoft.Data.SqlClient
  t.SqlDataReader.TryReadInternal(Boolean ignoreBuffer, Boolean readMore)\r\n at Microsoft.Data.SqlClient.SqlDataReader.ReadAsyncExecute(Task task, Object state)\r\n at Microsoft.Data.SqlClient.Sql
  DataReader.InvokeAsyncCall[[SqlDataReaderAsyncCallContext`1 context]]\r\n at Microsoft.EntityFrameworkCore.Update.AffectedCountModificationCommandBatch.ConsumeResultAsync(Int32 startCommandIndex, RelationalDataReader reader, CancellationToken cancellationToken)\r\n at Microsoft.EntityFrameworkCore.Update.AffectedCountModificatio
  nCommandBatch.ConsumeAsync(RelationalDataReader reader, CancellationToken cancellationToken)\r\n at Microsoft.Data.SqlClient.SqlClientConnectionId:9b71592-1c61-4f33-964b-f66d72a7fe8\r\n\r\nError Number:2601,State:1,Class:14",
  "payload": null,
  "errors": null
}
  
```

**Рис. 2.10. Негативний результат відправки даних у БД**

Після успішного нами запиту – наш проект спортивного магазину з’явиться у базі даних (див. рис. 2.11).

Id	Name
7	Cafe
21	Cafe1
19	dfrdfdf
1	Grocery shop
5	Hostel 'Arania'
3	Hotel 'Araga'
16	Арсен
18	АТБ
8	Сільпо
13	Сільпо2
22	Спортивний магазин
* NULL	NULL

**Рис. 2.11. Доданий проект у БД**

## 2.7. Клієнтська частина проекту

Фронтенд частина проекту була написана за допомогою програми «React». Це відкрита JavaScript бібліотека, що використовується для розробки динамічних інтерфейсів веб-додатків. Вона дозволяє розбити інтерфейс з використанням незалежних компонентів, якими можна легко управляти та повторно використовувати. React надає нам можливість ефективно оновлювати інтерфейс, який здатний реагувати на зміни даних, що забезпечує швидку розробку та високу продуктивність нашого додатку.

Для кращої структуризації коду ми використовували «TypeScript». Це мова програмування, яка розширенням JavaScript. Вона додає до JavaScript статичну типізацію, що дозволяє визначати типи даних для змінних, параметрів функцій, властивостей об'єктів та інших елементів коду. TypeScript полегшує розробку складних проектів, забезпечуючи переваги статичної типізації, такі як підказки про помилки під час розробки, поліпшена

автодоповнення коду та зрозуміліша документація. Він компілюється до чистого JavaScript і може бути використаний для розробки веб-додатків, серверних застосунків, мобільних додатків та багатьох інших сценаріїв.

Для того, щоб відтворити клієнтську частину так, щоб для клієнту все було зрозуміло і нативно ми завантажили NPM пакети за допомогою Node.js які пришвидшили, покращили роботу розробника, та в результаті призвели до створення якісного фронтенду. Список npm пакетів:

- React Router DOM [7] – це бібліотека, яка надає маршрутизацію (routing) для React-додатків, зокрема для веб-додатків, що використовують браузерний JavaScript. Вона дозволяє визначати шляхи (routes) в додатку і забезпечує перемикання між різними компонентами на основі поточного URL-адреси. React Router DOM надає різні компоненти, такі як `<BrowserRouter>`, `<Route>`, `<Link>`, які спрощують налаштування маршрутів, створення посилань та навігацію в додатку. Ця бібліотека допомагає реалізувати маршрутизацію в React-додатках, що дозволяє створювати багатосторінкові додатки зі зручною навігацією між сторінками.
- Formik [4] – це бібліотека для управління формами в React-додатках. Вона надає зручний спосіб управління станом форми, валідації даних, обробки подій та відправки форми. За допомогою Formik можна легко створювати форми з валідацією, обробляти їх стан та взаємодіяти зі значеннями полів форми. Вона також дозволяє використовувати різні компоненти, які пов'язані з формами, такі як `<Form>`, `<Field>`, `<ErrorMessage>`, що спрощують роботу з формами і зменшують необхідність вручну керувати станом та обробкою подій. Formik є потужним інструментом для розробки інтерактивних та динамічних форм в React-додатках, який полегшує роботу з управлінням даними форми. Приклад використання формік: 

```
const formik = useFormik({initialValues: init, onSubmit: HandleSubmit, validationSchema: ProjectCreateValidationSchema}).
```



- Yup [4] – це бібліотека для валідації даних в JavaScript-додатках. Вона надає простий та декларативний спосіб визначення правил валідації для об'єктів даних. За допомогою Yup можна встановлювати різноманітні правила валідації, такі як перевірка на обов'язкове заповнення поля, перевірка формату електронної пошти, числові обмеження, довжину рядків тощо. Використання Yup разом з іншими бібліотеками, такими як Formik або React Hook Form, дозволяє зручно валідувати та обробляти дані в формах, забезпечуючи надійну та коректну обробку введених користувачем даних. Приклад використання пакету yup: 

```
export const ProjectCreateSchema = yup.object().shape({ name: yup.string().required("Введіть назву проекту") });
```
- Redux – це управління станом (state management) бібліотека для JavaScript-додатків. Вона надає підхід до організації та керування станом додатка, що спрощує управління станом і взаємодію з ним у складних додатках. Приклад використання redux пакету: 

```
export const store = configureStore({ reducer: rootReducer, devTools: true, middleware: [thunk] });
```
- React-Redux – це офіційна бібліотека, яка поєднує дві потужні технології: React і Redux. Вона надає зручний спосіб інтеграції Redux з додатками на основі React, спрощуючи керування станом додатка і забезпечуючи ефективну взаємодію між компонентами. React-Redux дозволяє зв'язати React-компоненти зі станом Redux, дозволяючи компонентам отримувати необхідні дані зі стану і відслідковувати його зміни. Це досягається за допомогою двох основних компонентів - "Provider" і "connect".
- Redux DevTools Extension – це розширення для розробника, яке дозволяє візуалізувати та відстежувати стан і дії в Redux додатках. Воно надає потужний інструментарій для відлагодження та аналізу роботи з Redux. Завдяки Redux DevTools Extension, розробнику стає

легше спостерігати за змінами стану, відстежувати дії, які змінюють стан, і аналізувати взаємодію між компонентами та Redux store.

- **Redux Thunk** – це middleware для Redux, яке дозволяє використовувати асинхронні функції диспетчерів дій (action creators) в Redux. Зазвичай, у Redux дії (actions) є синхронними об'єктами, але за допомогою Redux Thunk можна створювати дії, які є асинхронними функціями. Коли Redux Thunk використовується як middleware, він перехоплює дії, які передаються до Redux store. Якщо дія є функцією замість об'єкта, Redux Thunk виконує цю функцію і передає в неї деякі параметри, такі як диспетчер (dispatch) і функцію для отримання поточного стану (getState). Це дає можливість виконувати асинхронні операції, такі як HTTP-запити до сервера, та управляти потоком дій у Redux. Основна перевага Redux Thunk полягає в тому, що він дозволяє обробляти складні логіки, які потребують асинхронних операцій, без необхідності використовувати інші бібліотеки або middleware. Він спрощує управління асинхронними запитами та дозволяє зберігати логіку обробки даних в одному місці - в диспетчерах дій.
- **@types/react-redux** – є типовим описом для бібліотеки React Redux у середовищі TypeScript. Він надає статичні типи для функцій, компонентів і об'єктів, які використовуються при розробці React Redux додатків з TypeScript.
- **@reduxjs/toolkit** – це офіційний пакет, який надає набір інструментів для спрощення розробки з використанням бібліотеки Redux. Він пропонує зручний та продуктивний підхід до написання коду Redux, забезпечуючи багато корисних функцій та шаблонів.
- **Axios** – це популярна бібліотека JavaScript, яка дозволяє здійснювати HTTP-запити з браузера або з середовища Node.js. Вона надає простий та зрозумілий API для виконання різних типів запитів, таких як GET, POST, PUT, DELETE і багатьох інших. Приклад

використання axios пакету: `const res = axios.post("api/project/create", values);`.

- React-Bootstrap – це набір готових компонентів і стилів для створення користувацького інтерфейсу веб-додатків з використанням бібліотеки React. Він поєднує можливості React з потужним фреймворком Bootstrap, що дозволяє швидко та ефективно розробляти сучасний та респонсивний дизайн.
- Bootstrap – це один з найпопулярніших фреймворків для розробки веб-інтерфейсів. Він надає набір готових компонентів, стилів та скриптів JavaScript, що допомагають швидко і просто створювати сучасні та естетичні веб-сайти. Приклад використання bootstrap пакету: `<h1 className = "text-center mt-3">Hello World</h1>`.
- Classnames – це JavaScript-бібліотека, яка допомагає зручно керувати класами елементів HTML на основі різних умов та даних. Основна функція бібліотеки - це генерація рядків класів на основі переданих параметрів. Ви можете використовувати Classnames для об'єднання різних класів, включення/виключення класів в залежності від умов, роботи з динамічними класами та багато іншого. Приклад використання Classnames пакету: `<input id="name" className={classnames( "form-control", { "is-invalid" : errors.name && touched.name} ) }/>`.
- React Icons – це набір готових векторних іконок, які призначені для використання в React-проектах. Цей набір дозволяє легко вставляти іконки в React-компоненти без необхідності завантажувати окремі зображення або використовувати CSS-стилі. Приклад використання React Icons пакету: `<i classNmae="fa fa-pencil" aria-hidden="true"></i>`.

## 2.8. Створення фронтенду для додатку

Для створення React проекту, нами була прописана команда у консолі: *npm create-react-app my-app --template typescript*. Дана команда створює react проект з використанням типізованої мови TypeScript.

Нами був підключений редакс у папці `src` → `components` → `containers` → `store` → `index.tsx`.

```
import { configureStore } from "@reduxjs/toolkit";
import { applyMiddleware, combineReducers, createStore } from "redux";
import { composeWithDevTools } from "redux-devtools-extension";
import thunk from "redux-thunk";
import { ProjectReducer } from "../../pages/Project/projectReducer";
export const rootReducer = combineReducers({ // імпортуєм наші редюсери
  project: ProjectReducer // підключити наш редюсер, який міститься під
  полем project
});
export const store = configureStore({ // це новий стор, в якому буде
зберігатись все, що в редаксі
  reducer: rootReducer, // набір наших редюсерів, які будуть в редаксі
  devTools: true, // щоб працював дев тулс, щоб в браузері дивитись через
F12
  middleware: [thunk] // щоб міг редакс працювати асинхроно
});
```

Нами був підключений редакс та маршрутизація у кореневому файлі `index.tsx`

```
const root = ReactDOM.createRoot(
  document.getElementById('root') as HTMLElement
);
root.render(
  <Provider store={store}>
    <HashRouter>
```

```

    <App />
  </HashRouter>
</Provider>
);

```

Також нами було здійснено підключення до серверу. Нами було створено файл «.env» у папці src, в якому нами була прописана зміна, яка зберігає домен нашого серверу:

```

REACT_APP_BASE_URL = https://localhost:7014/

```

Нами був створений файл index.tsx у папці «env» в якій нам був створений об'єкт для зберігання доменів.

```

const BASE_URL: string = process.env.REACT_APP_BASE_URL as string;
const APP_ENV = {
  BASE_URL: BASE_URL
}
export { APP_ENV };

```

Також нами був створений файл http\_common.ts у якому створений axios об'єкт через який можна надсилати запити на наш сервер.

```

import axios from 'axios';
import { APP_ENV } from './env'
const http = axios.create({
  baseURL: APP_ENV.BASE_URL,
  headers: {
    "Content-Type": "application/json"
  }
})
export default http;

```

Після, нами була створена папка «pages» у директорії «components» для створення усіх сторінок, які знаходяться на сайті.

У рамках дослідження нами було проведено аналіз та демонстрація операцій, пов'язаних з керуванням користувачами у системі. Показана буде послідовність роботи з користувачами, що включає виведення списку користувачів, видалення та редагування їх даних з використанням різноманітних додаткових npm пакетів.

З метою виведення списку клієнтів нами було застосовано відповідний алгоритм, що дозволяє отримати повний перелік клієнтів, збережених у системі, створення. Даний алгоритм передбачає отримання та відображення важливої інформації про кожного користувача, таку як ім'я, прізвище, електронна адреса та інші характеристики.

Помітною складовою роботи з клієнтами є їх видалення. Для цього нами були застосовані відповідні процедури, які дозволяють ідентифікувати та вилучити вказаного користувача з системи. Цей процес передбачає перевірку прав доступу та відповідних дозволів перед здійсненням видалення.

Крім того, нами буде проілюстровано можливість редагування даних користувачів. Це включає в себе внесення змін до існуючих записів користувачів, таких як зміна імені, прізвища, пошти та інших важливих параметрів. Цей процес вимагає валідації та перевірки даних, а також забезпечення відповідності з встановленими правилами та обмеженнями.

Загалом, проведення роботи з користувачами передбачає послідовну інтеракцію з їх даними, зокрема виведенням списку, видаленням та редагуванням. Ці операції виконуються з метою забезпечення належного керування користувачами та підтримки їхніх потреб у системі.

Для виведення списку користувачів нами було використано axios для отримання даних з бази даних надсилаючи http запит на нашу серверну частину

```
useEffect(() => {  
  console.log(search);
```

```

    http.get<ICustomerSearchResult>(`api/customer/search`, {
      params: search
    })
      .then(resp => {
        const { data } = resp;
        console.log("Server response", data);
        setSearchResult(data);
      }).catch(err => {
        console.log("ERR", err);
      })
  }, [search]);

```

Для візуалізації даних клієнтів з метою досягнення естетичного вигляду, нами було використано масив даних типу «map» та таблицю зі стилізацією Bootstrap фреймворком. Даний тип масиву використовувався для структурованого зберігання та обробки даних користувачів, включаючи їх основні атрибути та характеристики. Таблиця зі стилізацією фреймворку використовувалась для відображення цих даних у зручному та привабливому вигляді.

```

const tableViewList = customers.map((item) => {
  return (
    <tr key={item.id}>
      <th scope="row">{item.id}</th>
      <td>{item.firstName}</td>
      <td>{item.secondName}</td>
      <td>{item.phone}</td>
      <td>{item.email}</td>
      {item.address ? (
        <td>{item.address}</td>
      ) : (

```

```

        <td>-----</td>
    )}
    <td style={{ display: 'flex' }}>
        <Link to={`edit/${item.id}`} className="btn btn-warning"
        >
            Змінити <i className="fa fa-pencil" aria-hidden="true"></i>
        </Link>
        &nbsp;
        &nbsp;
        &nbsp;
        <ModalDelete
            id={item.id}
            text={`Ви дійсно бажаєте видалити користувача
            '${item.firstName} ${item.secondName}'?`}
            deleteFunc={onDeleteCustomer}
        />
    </td>
</tr>
)
})

```

Використання масиву типу «map» забезпечувало нам можливість ефективної обробки даних, їх впорядкування та маніпуляцій. Ми використовували його для створення структурованої колекції даних, де кожному користувачу був призначений унікальний ключ, що дозволяв швидкий доступ до конкретної інформації про кожного користувача.

Таблиця зі стилізацією Bootstrap була використана для створення гнучкого та привабливого інтерфейсу, де дані користувачів були представлені в табличному форматі. Вона надавала можливість керування виглядом таблиці, включаючи вирівнювання, колонки, рядки, а також додаткові



функціональні можливості, які забезпечували покращений візуальний досвід для користувачів.

Використання масиву типу «map» та таблиці зі стилізацією Bootstrap в поєднанні дозволило нам ефективно вивести дані користувачів, надаючи їм зручний та естетично приємний вигляд.

У нашій розробці була впроваджена спеціальна функціональність, що передбачає видалення користувачів з бази даних шляхом використання спеціальної кнопки, розташованої поруч із кожним користувачем. Ця кнопка служить для зберігання унікального ідентифікатора користувача, необхідного для здійснення операції видалення з бази даних.

При натисканні на цю кнопку, активується модальне вікно, реалізоване з використанням React Bootstrap пакету. Це вікно надає можливість користувачеві обрати, чи бажає він виконати дію видалення, чи відмовитися від неї. Такий підхід сприяє забезпеченню контролю над операцією видалення, дозволяючи користувачеві усвідомлено вирішити, чи бажає він виконати дану дію.

Таке використання кнопки та модального вікна сприяє поліпшенню інтерфейсу та користувацького досвіду, забезпечуючи зручність та узгодженість взаємодії з системою. Крім того, використання React Bootstrap пакету надає зручні та ефективні інструменти для реалізації модального вікна, сприяючи розширенню можливостей інтерфейсу та забезпеченню його сучасного та професійного вигляду.

```
<ModalDelete
  id={item.id}
  text={`Ви дійсно бажаєте видалити користувача '${item.firstName}
`${item.secondName}'?`}
  deleteFunc={onDeleteCustomer}/>
```

У процесі розробки продукту нами була розроблена функціональність, що включає створення кнопки над списком користувачів, що перенаправляє

користувача до окремого вікна за допомогою пакету React-Router-Dom для введення даних нового користувача. Вказане вікно використовує механізм валідації введених даних з використанням пакетів Formik та Yup для визначення схеми валідації. У разі успішної валідації, проте виникнення помилки на сервері при створенні користувача, наприклад, через неправильний телефон або вже зайняту електронну пошту, ці помилки повертаються та виводяться на клієнтську сторону з метою інформування користувача про виниклу проблему в процесі роботи.

Такий підхід до обробки помилок у реалізації функціональності продукту є необхідним для забезпечення коректної комунікації між сервером та клієнтом. Виведення помилок на клієнтську сторону дозволяє забезпечити взаємодію користувача з системою та відображення зрозумілої інформації про можливі проблеми, що виникають під час виконання відповідних операцій. Такий підхід сприяє покращенню користувацького досвіду та допомагає користувачеві зрозуміти причини невдалого виконання певних дій під час роботи з продуктом.

```
const onCreateSubmit = async (values: ICustomerCreateItem) => {
  console.log("CREATE VALUES", values);
  http.post("api/customer/create", values)
    .then(resp => {
      console.log("RESP", resp);
      const { payload } = resp.data;
      http.post(`api/CustomerProject/create/customer-
        ${Number(payload.id)}/project-${projectId}`)
        .then(resp => {
          console.log("RESP", resp)
          navigate("../");
        }).catch(err => {
          console.log("ERR", err);
        })
    })
}
```

```

    })
    .catch(err => {
      const { data } = err.response;
      console.log("ERR", data);
      if (data.message.includes("Повнота")) setFieldError("email",
data.message);
      else setFieldError("phone", data.message);
    })
  }
}

```

Візуальний вигляд вікна для редагування виглядає схоже як і вікно для створення користувача, але поля для ведення даних вже заповнені даними даного користувача якого збираємось змінити. Для цього нами створена додаткова кнопка біля кнопки видалення, яка також приймає унікальний ідентифікатор користувача та перекидує на сторінку редагування.

```

<Link to={`edit/${item.id}`} className="btn btn-warning">
  Змінити <i className="fa fa-pencil" aria-hidden="true"></i>
</Link>

```

Для отримання даних користувача, нами був прописаний код, для відправки http запиту на сервер для отримання даних користувача з бази даних за унікальним ключем.

```

useEffect(() => {
  http.get<ICustomerByIdServiceResponse>(`api/customer/customer/${id}`)

  .then(resp => {
    const { payload } = resp.data;
    console.log("RESP", resp);
    setFieldValue("firstName", payload.firstName);
    setFieldValue("secondName", payload.secondName);
  })
}

```

```

    setFieldValue("email", payload.email);
    setFieldValue("phone", payload.phone);
    setFieldValue("address", payload.address);
  }).catch(err => {
    console.log("ERR", err);
  })
}, [])

```

На даній сторінці нами також була передбачена валідація, отримання та виведення помилок із серверу як на сторінці створення користувача.

## 2.9. Десктопна частина проєкту

React переважно використовується для створення веб-сайтів. Але його код можливо інтегрувати в як настільний застосунок за допомогою програми «**Electron**». Це фреймворк для розробки десктопних додатків, який дозволяє використовувати веб-технології, такі як HTML, CSS і JavaScript для створення кросплатформових додатків, які працюють на різних операційних системах, включаючи Windows, macOS і Linux.

Основна ідея Electron в нашій роботі полягає в тому, що веб-технології можна використовувати для розробки не тільки веб-сайтів, але і десктопних додатків. За допомогою Electron, нам надано можливість збудувати додаток, який має власне вікно, інтерфейс користувача, доступ до файлової системи та інших нативних ресурсів операційної системи.

Однією з ключових переваг Electron є його кросплатформовість. Саме воно надає нам можливість розробляти додатки за допомогою Electron на одній операційній системі і вони будуть працювати на інших операційних системах без необхідності великих змін у коді. Це значно спрощує нам розробку додатків для різних платформ.

Крім того, Electron надає нам доступ до нативних API операційних систем, що дозволяє створювати багатofункціональні додатки з потужними

можливостями. Нам також надана можливість використовувати функціонал, який зазвичай доступний тільки для нативних додатків, такий як взаємодія з файловою системою, мережеві запити, сповіщення та багато іншого.

Загалом, Electron відкриває широкі можливості для розробки потужних кросплатформових десктопних додатків з використанням знайомих веб-технологій. Він став популярним вибором для багатьох розробників, які хочуть швидко створити десктопні додатки з використанням своїх веб-навичок.

## 2.10. Підключення клієнтської частини до десктопного варіанту

Для створення Electron проекту нам був прописаний ряд консольних команд:

1. `npm i -g @electron-forge/cli`
2. `electron-forge init electron-app`
3. `npm install electron-devtools-installer` → з метою користування `redux` у консолі

Після ініціалізації даного проекту, нами було виконано процес підключення до хосту серверної частини. З метою встановлення зв'язку між сервером та фронтендом, було виконано внесення відповідної конфігурації у файлі «`main.js`» розташованому в папці «`src`». Даний файл відіграє важливу роль у процесі запуску та взаємодії з фронтенд хостом: `mainWindow.loadURL("http://localhost:3000");`

Виконання рядка коду у файлі "main.js" забезпечує успішне підключення до фронтенд хосту, що вимагається для забезпечення правильної роботи серверної частини проекту. Цей крок є необхідним для налагодження зв'язку та обміну даними між сервером та фронтендом, що забезпечує ефективну взаємодію між цими компонентами.

Розташування рядка коду в файлі «`main.js`» демонструє належне організування структури проекту та виконання необхідних дій для підключення до фронтенд хосту. Цей процес є необхідною передумовою для

подальшого взаємодії з фронтом та забезпечення належного функціонування серверної частини проекту.

У контексті нашого дослідження, з метою підключення інструментів Redux до веб-консолі, ми використали певний код, який був вписаний у файл `src` → `main.js`. Цей код включав необхідні інструкції та налаштування, які дозволяють належно підключити та інтегрувати Redux з веб-консоллю.

```
const installExtension = require("electron-devtools-installer").default;
const { REDUX_DEVTOOLS } = require("electron-devtools-installer");
app.whenReady().then(() => {
  installExtension(REDUX_DEVTOOLS).then((name)=>console.log(`Added Extension: ${name}`)).catch((err)=>console.log("An error occurred: ", err));
});
```

Впровадження даного коду у файлі `src` → `main.js` забезпечує необхідні дії для налагодження зв'язку між Redux-інструментами та веб-консоллю. Це включає налаштування потрібних залежностей, ініціалізацію Redux-сторю та його підключення до веб-консолі. Така інтеграція забезпечує можливість використання функціоналу Redux, зокрема, моніторингу стану додатку та відлагодження дій, здійснюваних у веб-консолі.

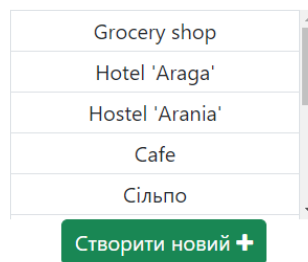
Додатково, варто відзначити, що код, внесений у файл `src` → `main.js`, відображає важливий крок у процесі інтеграції Redux з веб-консоллю, який дозволяє забезпечити ефективне використання інструментів Redux для веб-розробки та полегшує відладку та керування станом додатку у веб-середовищі.

## РОЗДІЛ 3. ПРАКТИЧНЕ ВИКОРИСТАННЯ ДОДАТКУ НА КЛІЄНТСЬКІЙ ЧАСТИНІ

### 3.1. Гловне меню

При першому відкритті додатку користувачу надається можливість обрати проект в якому він планує вносити зміни або ж створити новий (див. рис. 3.1).

#### Оберіть Проект



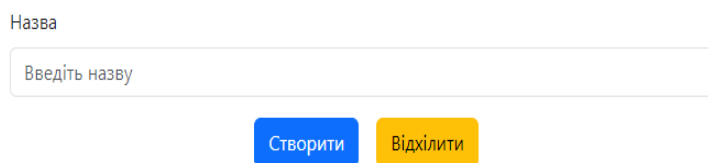
Grocery shop
Hotel 'Araga'
Hostel 'Arania'
Cafe
Сільпо

Створити новий +

**Рис. 3.1. Вибір проекту**

При натиску на кнопку створення нового проекту, фізичній особі-підприємцю відкривається нове вікно де потрібно ввести назву проекту (див. рис. 3.2).

#### Створення Проекту

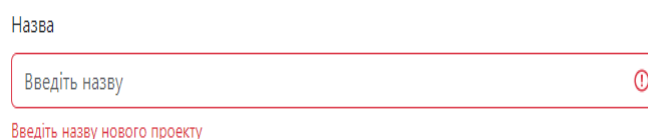


Назва

Створити Відхилити

**Рис. 3.2. Створення проекту**

При створенні проекту передбачена валідація та отримання помилки від сервера, якщо назва проекту вже існує (див. рис. 3.3).

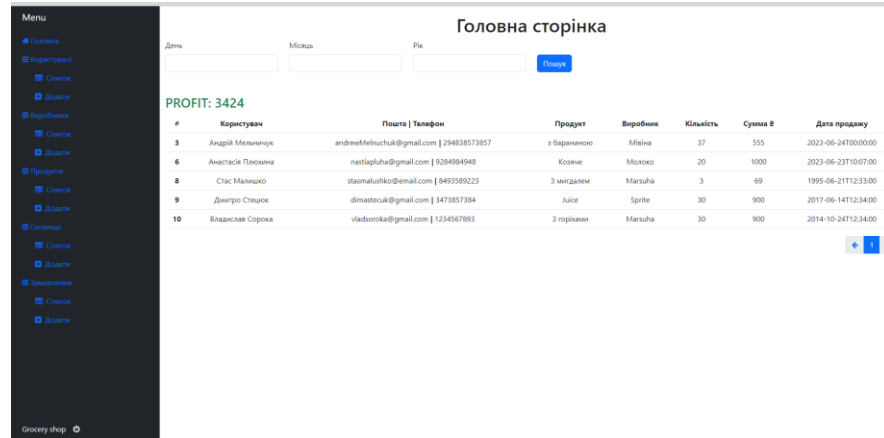


Назва

Введіть назву нового проекту

**Рис. 3.3. Вигляд валідації назви проекту**

При успішному створенні проекту – користувачу відкриється головне меню з даними списком проектів. Після вибору проекту, в якому збирається працювати ФОП – його відразу перенаправить на головну сторінку додатка (див. рис. 3.4).



**Рис. 3.4. Гловна сторінка**

Зліва користувачу відображається бокова панель, яка надає можливість легко зрозуміти, який функціонал йому передбачений у даній програмі: список користувачів, продуктів, виробників, сховища, замовлень та їх створення. У лівому нижньому куту відображена назва поточного проекту та кнопка виходу із нього. При натиску на дану кнопку – користувача перенаправить на сторінку з вибором проекту.

На головній сторінці (див. рис. 3.4) відображається список замовлень за весь час. На сторінці, нами також передбачено пагінація, пошук по даті та вивід загального заробітку. При пошуку по певній даті – оновлюється список замовлень та загальний заробіток. Тим самим це значно полегшує фізичній особі-підприємцю керувати свій заробіток та аналізувати подальшу роботу зі своїм проектом чи бізнесом.

### 3.2 Робота з клієнтами

При переході на список клієнтів – ФОПа відразу ж зустрічає їх список. Також нами була передбачена пагінація списку, пошук по імені, прізвищі,



пошти, телефону та адресі, додавання, видалення та редагування клієнту (див. рис. 3.5).

**Список клієнтів**

Ім'я:  Пошук по назві | Фамілія:  Пошук по фамілії | Пошта:  Пошук по пошті | Телефон:  Пошук по телефону | Адреса:  Пошук по адресі |

#	Ім'я	Фамілія	Телефон	Пошта	Адреса		
19	Маляр	Коцюба	1234267891	malarokocuba@gmail.com	-----	<input type="button" value="Змінити"/>	<input type="button" value="Видалити"/>
21	Андрій	Коваль	395739108983	andrewkoyal@gmail.com	-----	<input type="button" value="Змінити"/>	<input type="button" value="Видалити"/>
22	Владислав	Сорока	1234567893	vladsoroka@gmail.com	Вол Див 25	<input type="button" value="Змінити"/>	<input type="button" value="Видалити"/>
24	Андрій	Мельничук	294838573857	andrewMelnychuk@gmail.com	Шухевича 25	<input type="button" value="Змінити"/>	<input type="button" value="Видалити"/>
25	Дмитро	Стецюк	3473857384	dimastecuk@gmail.com	-----	<input type="button" value="Змінити"/>	<input type="button" value="Видалити"/>
26	Анастасія	Плюхина	9284984948	nastapluha@gmail.com	-----	<input type="button" value="Змінити"/>	<input type="button" value="Видалити"/>
27	Стас	Малишко	8493589223	stasmalushko@email.com	-----	<input type="button" value="Змінити"/>	<input type="button" value="Видалити"/>
28	Катя	Малиновська	538938493849	katymalynowska@gmail.com	-----	<input type="button" value="Змінити"/>	<input type="button" value="Видалити"/>

**Рис. 3.5. Список клієнтів**

При натиску на кнопку додавання нового клієнту – користувача перенаправляє на сторінку зі створенням клієнта (див. рис. 3.6).

**Створення Клієнта**

Ім'я \*  Фамілія \*

Пошта \*

Телефон \*

Адреса

**Рис. 3.6. Сторінка створення клієнта**

На даній сторінці позначено червоною зіркою поля, які є важливими для введення. При невірному веденні даних – користувач отримує список помилок які чітко дають зрозумів, що саме було зроблено не так (див. рис. 3.7).

## Створення Клієнта

Ім'я \*  Введіть ім'я

Фімілія \*  Введіть фамілію

Пошта \*  Введіть пошту

Телефон \*  Введіть телефон

Адреса

**Рис. 3.7. Валідація створення клієнта**

При успішній відправці даних – клієнт створюється у базі даних та буде виведений у списку клієнтів на фронтенд частині. Самого користувача перенаправляє до списку клієнтів.

При натиску на кнопку редагування існуючого клієнта – користувача перенаправляє на сторінку із редагуванням клієнта. Поля заповнені актуальною інформацією даного клієнта (див. рис. 3.9).

## Редагування користувача

Ім'я \*

Фімілія \*

Пошта \*

Телефон \*

Адреса

**Рис. 3.9. Вікно редагування клієнта**

На сторінці присутня валідація даних. Якщо одне з важливих полей буде пусте, або ж пошта чи телефон зайняті іншими клієнтами – буде отримана певна помилка для користувача (див. рис. 3.10).

### Редагування користувача

Ім'я\*  
Введіть ім'я

Фімілія\*  
Введіть Фамілію

Пошта\*  
katymalynowska@gmail.com

Телефон\*  
538938493849

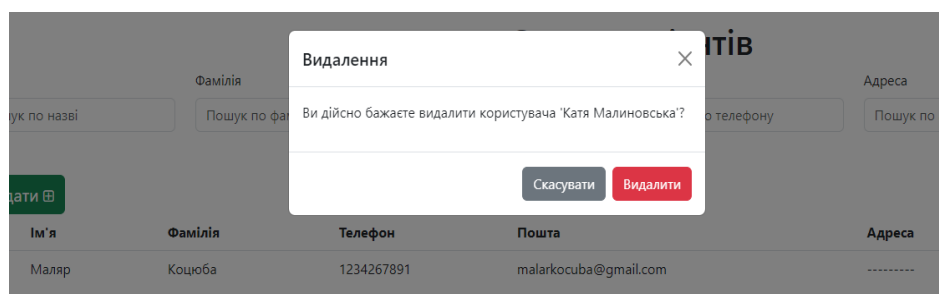
Адреса  
Введіть адреса

Змінити Відхилити

**Рис. 3.10. Валідація редагування клієнта**

Після успішного редагування клієнта – користувача перенаправляє на список клієнтів, де той клієнт в якого вносили зміни буде з актуальною зміненою інформацією про себе.

При натисканні на кнопку видалення клієнта, наша програма відкриває модальне вікно (див. рис. 3.8). У якому чітко, коротко та зрозуміло вказана інформація кого саме користувач збирається видалити та запитує чи збирається він видалити саме цього клієнта. У ФОПа є дві кнопки. Одна з яких відповідає за відміну видалення, у разі чого модальне вікно закривається та доступ до іншого функціоналу на сайті стає знову доступним. Інша кнопка відповідає за видалення користувача.



**Рис. 3.8. Модальне вікно по видаленню клієнта**

Під час натиску на дану кнопку – клієнт відразу ж видаляється з бази даних автоматично, та список на фронтенді моментально оновлюється, тим самим даючи ФОПу бачити актуальну інформацію стосовно своїх клієнтів на даному проекті.

### 3.3. Робота з виробниками та продуктами

При переході на список виробників – ФОПа відразу ж зустрічає їх список. Також нами була передбачена пагінація списку, пошук по назві, додавання, редагування та видалення виробника (див. рис. 3.11).

**Виробники**

Назва

#	Назва	Змінити ✎	Видалити ✖
1	Pepsi	Змінити ✎	Видалити ✖
17	Coca-cola	Змінити ✎	Видалити ✖
18	Fanta	Змінити ✎	Видалити ✖
19	Mojo-Mojo	Змінити ✎	Видалити ✖
20	Coca-cola zero	Змінити ✎	Видалити ✖
24	Nivea	Змінити ✎	Видалити ✖
25	Snickers	Змінити ✎	Видалити ✖
26	Marsuha	Змінити ✎	Видалити ✖

**Рис. 3.11. Список виробників**

При натиску на кнопку додавання – користувач перенаправляє на сторінку зі створенням нового виробника (див. рис. 3.12).

### Створення Виробника

Назва \*

**Рис. 3.12. Сторінка створення виробника**

На даній сторінці позначено червоною зіркою поля, які є важливими для введення. При невірному введенні даних – користувач отримує список помилок які чітко дають зрозумів, що саме було зроблено не так (див. рис. 3.13).

Назва \*

Pepsi

Назва продукту вже зайнята

**Рис. 3.13. Валідація створення виробника**

При успішній відправці даних – виробник створюється у базі даних та буде виведений у списку виробників на фронтенд частині. Самого користувача перенаправляє до списку виробників.

Функціонал видалення виробника працює аналогічно за видаленням клієнта з бази даних

При натиску на кнопку редагування існуючого виробника – користувача перенаправляє на сторінку із редагуванням. Користувачу надається можливість змінити дане ім'я виробника. Поля заповнені актуальною інформацією даного виробника (див. рис. 3.14).

### Редагування виробника

Назва \*

Pepsi

Змінити Відхилити

**Рис. 3.14. Вікно редагування виробника**

На сторінці присутня валідація даних аналогічна за дизайном валідації редагування клієнта. Якщо одне з важливих полей буде пусте, або ж назва виробника вже зайнята іншими – буде отримана певна помилка для користувача

Після успішного редагування виробника – користувач буде перенаправлений на список виробників, де той виробник в якого вносили зміни буде з актуальною зміненою інформацією про себе.

Список продуктів, їх редагування, видалення та створення працює за аналогією роботи з виробниками.

У списку продуктів відображається їх унікальний ідентифікатор, назва, та назва виробника від якого вони походять (див. рис. 3.15).

**Список Продуктів**

**Додати**

#	Назва	Виробник	<input type="button" value="Додати"/>	<input type="button" value="Видалити"/>
14	Лісе	Sprite	<input type="button" value="Додати"/>	<input type="button" value="Видалити"/>
16	з куркою	Milena	<input type="button" value="Додати"/>	<input type="button" value="Видалити"/>
19	з бараниною	Milena	<input type="button" value="Додати"/>	<input type="button" value="Видалити"/>
24	з кокосом	Snickers	<input type="button" value="Додати"/>	<input type="button" value="Видалити"/>
25	з мигдалем	Snickers	<input type="button" value="Додати"/>	<input type="button" value="Видалити"/>
26	з кокосом	Marsuha	<input type="button" value="Додати"/>	<input type="button" value="Видалити"/>
28	з мигдалем	Marsuha	<input type="button" value="Додати"/>	<input type="button" value="Видалити"/>
29	з горіхами	Marsuha	<input type="button" value="Додати"/>	<input type="button" value="Видалити"/>

**Рис. 3.15. Сторінка списку продуктів**

На сторінці додавання продукту існує два необхідних поля для заповнення: текстове поле для назви продукту та випадаючий список усіх існуючих виробників у базі даних (див. рис. 3.16).

### Створення продукту

Назва

Оберіть Виробника

**Рис. 3.16. Сторінка створення продукту**

На даній сторінці також приступна валідація нічим не відрізняючись від вище згаданих прикладів.

Після успішної відправки даних, продукт створюється у базі даних та відображається у даному списку продуктів

## 3.4. Робота зі сховищем

При переході на список продуктів у сховищі – ФОПа відразу ж зустрічає їх список у якому відображається унікальний ідентифікатор продукту, назва продукту, виробника, опис, ціна за одну штуку, кількість та термін придатності. Також нами була передбачена пагінація списку, пошук по назві продукту та виробника, пошук по опису, кількості та ціні, додавання, видалення та

редагування продукту за схожою логікою що і у вище згаданих пунктах. Також у списку виділено червоним кольором кількість продуктів, якщо вона рівна нулю та термін придатності якщо він уже пройшов (див. рис. 3.17).

**Сховище**

Продукт: 
 Виробник: 
 Опис: 
 Кількість: 
 Ціна:

# Продукту	Продукт	Виробник	Опис	Ціна (в грн)	Кількість	Термін призначеності		
14	Juice	Sprite	Супер крутий сік від спрайта!!!	30	75	2023-08-22T06:45:00	<input type="button" value="Змінити"/>	<input type="button" value="Видалити"/>
15	Звичайний	Sprite	Звичайний спрайт, без добавок	10	300	1995-12-15T06:45:00	<input type="button" value="Змінити"/>	<input type="button" value="Видалити"/>
19	з бараниною	Мівіна	укр мівіна з бараниною	15	0	2023-10-15T06:45:00	<input type="button" value="Змінити"/>	<input type="button" value="Видалити"/>
20	Козяче	Молоко	Дуже смачне молоко козяче	50	0	2023-06-30T06:45:00	<input type="button" value="Змінити"/>	<input type="button" value="Видалити"/>
28	З мигдалем	Marsuha	батончик mars з мигдалем	23	232	2024-06-22T06:45:00	<input type="button" value="Змінити"/>	<input type="button" value="Видалити"/>
29	З горіхами	Marsuha	Батончик від марсуха з горіхами ДУЖЕ СМАЧНО	30	0	2023-06-30T20:20:00	<input type="button" value="Змінити"/>	<input type="button" value="Видалити"/>

**Рис. 3.17. Сторінка списку продуктів у сховищі**

При натиску нами на кнопку додавання нового продукту до сховища – користувача перенаправляє на сторінку зі створенням нового продукту (див. рис. 3.18).

**Сховище - добавлення продукту**

Оберіть Виробника: 
 Оберіть Продукт:

Опис:

Ціна:

Кількість:

Термін придатності:

**Рис. 3.18. Сторінка добавлення продукту до сховища**

На даній сторінці існує два випадających списки по виробникам та продуктам. Потрібно обрати виробника для того, щоб обрати один із продуктів

які відносяться до нього, інакше список продуктів у випадяючому списку буде нулевий. Також присутні всі інші поля які потрібні для створення.

При невірному веденні даних – користувач отримує список помилок які чітко дають зрозуміти, що саме було зроблено не так. Відображення валідації схоже по аналогії на валідацію продуктів.

Функціонал видалення та редагування схожий нами по аналогії роботи з іншими об'єктами проєкту.

### 3.5. Робота із замовленнями даного проєкту

У списку замовлень нами була передбачена пагінація списку, додавання, видалення та редагування продукту (див. рис. 3.19).

У списку відображається унікальний ключ клієнта, його ПІБ, пошта, телефон, продукт та виробник який він придбав, кількість придбаного продукту, загальна сума закупки та дата покупки

**Замовлення**

Додати

#	Користувач	Пошта   Телефон	Продукт	Виробник	Кількість	Сумма ₴	Дата продажу		
3	Андрій Мельничук	andrewMelnychuk@gmail.com   294838573857	з бараниною	Мівіна	37	555	2023-06-24T00:00:00	<span style="background-color: #ffc107; padding: 2px 5px;">Змінити</span>	<span style="background-color: #dc3545; color: white; padding: 2px 5px;">Видалити</span>
6	Анастасія Плохіна	nastiapluha@gmail.com   9284984948	Козяче	Молоко	20	1000	2023-06-23T10:07:00	<span style="background-color: #ffc107; padding: 2px 5px;">Змінити</span>	<span style="background-color: #dc3545; color: white; padding: 2px 5px;">Видалити</span>
8	Стас Малишко	stasmalushko@gmail.com   8493589223	3 мигдалем	Marsuha	3	69	1995-06-21T12:33:00	<span style="background-color: #ffc107; padding: 2px 5px;">Змінити</span>	<span style="background-color: #dc3545; color: white; padding: 2px 5px;">Видалити</span>
9	Дмитро Стецюк	dimastecuk@gmail.com   3473857384	Juice	Sprite	30	900	2017-06-14T12:34:00	<span style="background-color: #ffc107; padding: 2px 5px;">Змінити</span>	<span style="background-color: #dc3545; color: white; padding: 2px 5px;">Видалити</span>
10	Владислав Сорока	vldsoroka@gmail.com   1234567893	3 горіхами	Marsuha	30	900	2014-10-24T12:34:00	<span style="background-color: #ffc107; padding: 2px 5px;">Змінити</span>	<span style="background-color: #dc3545; color: white; padding: 2px 5px;">Видалити</span>

←
1
→

**Рис. 3.19. Список замовлень**

При натиску на кнопку додавання нового замовлення – користувача перенаправляє на сторінку зі створенням нового замовлення (див. рис. 3.20).



## Створення замовлення

Оберіть Клієнта

---

Оберіть Виробника

---

Оберіть Продукт

---

Кількість продукту

0 ≥ 0

Дата продажу

mm/dd/yyyy --:-- --

Створити Відхилити

**Рис. 3.20. Сторінка створення замовлення**

У даній сторінці при створенні замолення передбачені три випадających списки, одне поле для ведення кількості продуктів та дата продажу:

1. випаданий список клієнтів – відображаються лише ті клієнти, що відносяться до даного проекту;
2. випаданий список виробників – відображаються лише ті виробники, що існують у сховищі проекту;
3. випаданий список продуктів – відображається список продуктів певного виробника, що також лише існують у сховищі проекту;
4. поле вибору кількості продуктів – введення кількості продуктів які плануються продатись, якщо кількість буде більша ніж та, що у сховищі виведеться помилка;
5. поле вибору дати продажу.

При невірному веденні даних – користувач отримує список помилок які чітко дають зрозумів, що саме було зроблено не так (див. рис. 3.21).

## Створення замовлення

Оберіть Клієнта

---

Оберіть клієнта

Оберіть Виробника

---

Оберіть Продукт

---

Виберіть виробника для продукту

Оберіть продукт

Кількість продукту

0

Виберіть кількість продуктів (мін. 1)

Дата продажу

mm/dd/yyyy --:-- --

Введіть дату продажу

Створити Відхилити

**Рис. 3.21** Валідація створення замовлення

При успішній відправці даних – замовлення створюється у сховищі бази даних, кількість що була введена видалиться зі сховища та буде виведений у списку сховища на фронтенд частині. Самого користувача перенаправляє до списку замовлень.

При натиску на кнопку редагування існуючого замовлення – користувача перенаправляє на сторінку із редагуванням. Поля заповнені актуальною інформацією даного замовлення (див. рис. 3.22).

## Редагування замовлення

Оберіть Виробника

Андрій Мельничук andrewMelnuchuk@gmail.com

Оберіть Виробника

Мівіна

Оберіть Продукт

з бараниною

Кількість продукту

37

555

Дата продажу

06/24/2023 12:00 AM

Створити Відхилити

**Рис. 3.22** Сторінка редагування замовлення

## ВИСНОВКИ

У кваліфікаційній роботі нами було розроблено потужний алгоритм та інноваційна клієнтська частина, що надають приватному підприємцю зручність та ефективність використання MRP-2 системи під час розробки проекту. Це дозволило нам значно зменшити час виконання завдань та забезпечити високу продуктивність

Перед початком роботи нами було проведено докладний аналіз конкуруючих веб-сайтів та додатків, враховані всі переваги та недоліки. Зроблено вагомий крок у залученні клієнтів на свою сторону шляхом розробки привабливої та функціональної платформи.

У розробці нами були використані передові технології як у серверній, так і у клієнтській частині. Для фронтенду було обрано React разом з потужними інструментами, такими як Redux, Formik, Yup, Classnames, React Icons і React Router Dom. Це дозволило нам створити сучасну та динамічну користувацьку інтерфейсну частину.

Завдяки використанню React, разом з його екосистемою і широким спектром додаткових бібліотек, нами було забезпечено високу розширюваність та перевикористовування коду. Компонентний підхід дозволив нам ефективно організувати структуру проекту, розділити його на незалежні компоненти та забезпечити простоту управління станом додатка.

Formik разом із Yup стали потужним інструментарієм для валідації та обробки форм. Вони спростили нам процес створення та керування формами, забезпечили зручну валідацію даних та повідомлення про помилки.

Classnames був надійним помічником при роботі з CSS-класами. Він дозволяв нам динамічно додавати або видаляти класи на основі умов, що робило стилізацію компонентів більш гнучкою та контрольованою.

React Icons надавав широкий вибір іконок, які можна було легко використовувати у проекті. Він пропонував набір готових іконок різних стилів та форматів, що спрощувало розробку і покращувало візуальний вигляд додатка.

React Router Dom забезпечував нам маршрутизацію в додатку, дозволяючи навігацію між різними сторінками та компонентами. Це забезпечувало зручне управління структурою додатка та перехід між різними роутами.

Щодо серверної частини, нами використовувався ASP.NET Core Web API з використанням потужних бібліотек, таких як FluentValidation, AutoMapper, Ardalis. Це надало надійну та ефективну основу для створення бази даних та забезпечення її безпеки.

ASP.NET Core Web API разом із FluentValidation, AutoMapper та Ardalis стали нам основою для розробки серверної частини. FluentValidation дозволив нам зручно валідувати вхідні дані на сервері, забезпечуючи їх правильність та безпеку. AutoMapper спрощував мапінг об'єктів між різними моделями даних, що полегшувало роботу з базою даних. Ardalis, з своїм набором корисних інструментів та методологій, сприяв структуруванню та покращенню якості коду.

У підсумку, комбінація всіх цих інструментів та технологій дозволила створити потужний та зручний для використання проект. Вона забезпечила швидку розробку, високу продуктивність та гнучкість, що робить цей проект важливим інструментом для роботи з базами даних та розробки приватного-підприємства.

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. AutoMapper documentation. URL: <https://docs.automapper.org/> (дата звернення 25.06.2023).
2. ClassNames documentation. URL: <https://www.npmjs.com/package/classnames> (дата звернення 25.06.2023).
3. FluentValidation documentation. URL: <https://fluentvalidation.net/> (дата звернення 25.06.2023).
4. Formik documentation. URL: <https://formik.org/> (дата звернення 25.06.2023).
5. Official React documentation URL: <https://legacy.reactjs.org/docs> (дата звернення 25.06.2023).
6. Official ASP. NET Core documentation. URL: <https://learn.microsoft.com/ru-ru/aspnet/core/?view=aspnetcore-7.0>. (дата звернення 25.06.2023).
7. React Router Dom library documentation and examples. URL: <https://reactrouter.com/> (дата звернення 25.06.2023).
8. Stack Overflow. URL: <https://stackoverflow.com/> (дата звернення 25.06.2023).
9. Latest Articles. URL: <https://ardalis.com/blog> (дата звернення 25.06.2023).
10. Learn to Code. URL: <https://www.w3schools.com> (дата звернення 25.06.2023).

## ДОДАТКИ

## Додаток А

## Приклад пошуку продуктів у сховищі за полями

```

[HttpGet("search/{projectId}")]
public async Task<IActionResult> Search(int projectId, [FromQuery]
StorageSearchDto search)
{
    try
    {
        int page = search.Page;
        int pageSize = 8;
        var query = _context.Storage
            .Include(p => p.Project)
            .Include(p => p.Product)
            .Where(s => s.ProjectId == projectId)
            .AsQueryable();

        if (!string.IsNullOrEmpty(search.ProductName))
        {
            query = query.Where(x =>
x.Product.Name.ToLower().Contains(search.ProductName.ToLower()));
        }
        if (!string.IsNullOrEmpty(search.ManufacturerName))
        {
            query = query.Where(x =>
x.Product.Manufacturer.Name.ToLower().Contains(search.ManufacturerName.To
Lower()));
        }
        if (!string.IsNullOrEmpty(search.Description))
        {
            query = query.Where(x =>
x.Description.ToLower().Contains(search.Description.ToLower()));
        }
        if (search.Price != 0)
        {
            query = query.Where(x => x.Price == search.Price);
        }
        if (search.Count != 0)
        {
            query = query.Where(x => x.Count == search.Count);
        }
        var model = await query
            .Skip((page - 1) * pageSize)

```

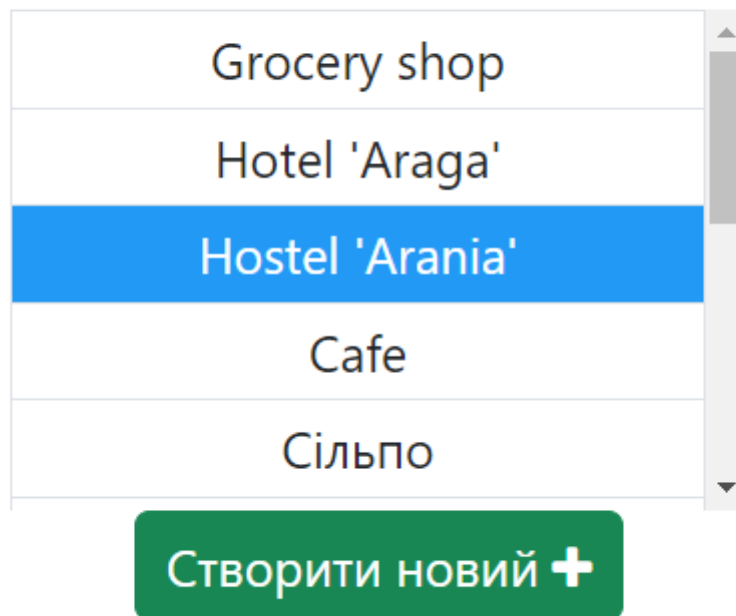
```

        .Take(pageSize)
        .Select(x => _mapper.Map<StorageDto>(x))
        .ToListAsync();
        model.ForEach(m => m.Product =
_mapper.Map<ProductDto>(_context.Products.FirstOrDefault(p => p.Id ==
m.ProductId)));
        model.ForEach(m => m.Product.ManufacturerName =
_context.Manufacturers.FirstOrDefault(x => x.Id ==
m.Product.ManufacturerId).Name);
        model.ForEach(m => m.Project =
_mapper.Map<ProjectDto>(_context.Projects.FirstOrDefault(p => p.Id ==
m.ProjectId)));
        int total = query.Count();
        int pages = (int)Math.Ceiling(total / (double)pageSize);
        return Ok(new StorageSearchResultDto
        {
            Storage = model,
            Total = total,
            CurrentPage = page,
            Pages = pages
        });
    }
    catch(Exception ex)
    {
        return BadRequest(ex.Message);
    }
}

```

## Створення сторінки виводу списку проектів

# Оберіть Проект



```
import "./scroll-logic.js";
import "./styles.css";
import { useEffect, useState } from "react";
import http from ".././././http_common";
import { IProjectItem, IProjectServiceResponse, StoreProjectActionType }
from "../types";
import { Link, useNavigate } from "react-router-dom";
import { store } from ".././././containers/store";

const ChooseProject = () => {
  const navigate = useNavigate();
  const [projects, setProjects] = useState<IProjectItem[]>([]);
  const [chosenProject, setChosenProject] = useState<IProjectItem>({
```



```

    id: -1,
    name: ""
  });

useEffect(() => {
  http.get<IProjectServiceResponse>("api/project/projects")
    .then(resp => {
      const { payload } = resp.data;
      console.log(payload);
      setProjects(payload);
    }).catch(err => {
      console.log("ERR", err);
    })
}, []);

const viewList = projects.map((item) => (
  <tr key={item.id} className="project-item">
    <td
      className="project-data"
      onClick={() => onClickProject(item)}
      onDoubleClick={() => onDoubleClickProject(item)}
    >
      {item.name}
    </td>
  </tr>
))

const onClickProject = (project: IProjectItem) => {
  setChosenProject({ id: project.id, name: project.name });
}

```

```

const onDoubleClickProject = (project: IProjectItem) => {
  console.log("Project clicked =>", project.id);
  window.localStorage.chosenProjectId = chosenProject.id;
  window.localStorage.chosenProjectName = chosenProject.name;
  store.dispatch({ type:
StoreProjectActionType.STORE_CREATE_PROJECT, payload: chosenProject })
  console.log("Store project into the redux ChooseProject.tsx");
  navigate("/main-page");
}

return (
  <>
    <div className="container d-flex flex-column justify-content-center
align-items-center">
      <h1 className="text-center">Оберить Проект</h1>

      <div className="container mt-3 w-25 fs-5">
        <table
          id="dtVerticalScrollExample"
          className="table table-bordered table-sm"
          cellSpacing={0}
          width="100%">
          <thead>
            <tr>
              <th className="d-none"></th>
            </tr>
          </thead>
          <tbody className="text-center">
            <tr className="d-none">

```

```

        <td></td>
    </tr>
    {viewList}
</tbody>
</table>
<div className="d-flex justify-content-center">
    <Link className="btn btn-success fs-5" to="project-
create">Створити новий <i className="fa fa-plus" aria-
hidden="true"></i></Link>
    </div>
</div>
</div>
</>
)
}

export default ChooseProject;

```

### Налаштування таблиці списку проектів для прокрутки

```

var $ = require("jquery");
var DataTable = require("datatables.net-dt");

$(document).ready(function () {
  $("#dtVerticalScrollExample").DataTable({
    scrollY: "200px",
    scrollCollapse: true,
  });
  $(".dataTables_length").addClass("bs-select");
  $("#dtVerticalScrollExample_length").css("display", "none");
  $("#dtVerticalScrollExample_filter").css("display", "none");
  $("#dtVerticalScrollExample_info").css("display", "none");
  $("#dtVerticalScrollExample_paginate").css("display", "none");
});

table.dataTable thead .sorting:after,
table.dataTable thead .sorting:before,
table.dataTable thead .sorting_asc:after,
table.dataTable thead .sorting_asc:before,
table.dataTable thead .sorting_asc_disabled:after,
table.dataTable thead .sorting_asc_disabled:before,
table.dataTable thead .sorting_desc:after,
table.dataTable thead .sorting_desc:before,
table.dataTable thead .sorting_desc_disabled:after,
table.dataTable thead .sorting_desc_disabled:before {
  bottom: 0.5em;
}

```

```
.project-item {  
  cursor: alias;  
}
```

```
.project-data:hover {  
  color: #ffffff;  
  background-color: #229af5;  
}
```