Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних наук та інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

КОМПЛЕКСНА СИСТЕМА ОБЛІКУ ПРОДУКЦІЇ ПІДПРИЄМСТВА НА БАЗІ ТЕХНОЛОГІЇ .NET

A COMPREHENSIVE SYSTEM OF ACCOUNTING FOR THE COMPANY'S PRODUCTS BASED ON .NET TECHNOLOGIES

спеціальність 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

освітня програма «Інженерія програмного забезпечення»
(назва освітньої програми)

Виконав: здобувач вищої освіти
Групи ІПЗс-31
_____ Михалевич Н.Д.
(підпис)

Керівник: к.т.н., доцент
_____ Ящук. А.А
(підпис)

Кваліфікаційну роботу
допущено до захисту
«1» 06 2023 р.
Гарант освітньої програми:
к.т.н., доцент
Ліщина Наталія Миколаївна
_____
(підпис)

Луцьк – 2023 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *інженерії програмного забезпечення*
Ступінь вищої освіти: *бакалавр*
Галузь знань: *12 Інформаційні технології*
Спеціальність: *121 Інженерія програмного забезпечення*
Освітня програма: *«Інженерія програмного забезпечення»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____
«02» 02 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Михалевичу Назару Дмитровичу

1. Тема кваліфікаційної роботи Комплексна система обліку продукції підприємства на базі технології .Net

Керівник роботи: к.т.н., доцент Ящук Андрій Анатолійович
затверджені наказом закладу вищої освіти від «23» грудня 2022 р. № 961-01-02

2. Строк подання здобувачем вищої освіти кваліфікаційної роботи «8» червня 2023 р.

3. Вихідні дані до роботи: *технічне та програмне забезпечення ЕОМ, вимоги до розробки програмного забезпечення, ергономічні вимоги до функціонування програмного засобу. Мова програмування серверної частини – .Net. Технології програмування клієнтської частини – HTML, CSS, React JS.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити):
Аналіз сучасного стану проблеми, існуючих методів і засобів її розв'язання, аналіз і вибір засобів проєктування, опис функціонального наповнення об'єкта проєктування, розробка й обґрунтування системного наповнення, оцінка ергономічних та надійністних параметрів проєктованої системи, розробка моделі бази даних об'єкта проєктування

5. Перелік графічного матеріалу:
16 рис; 2 схеми.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області	*Ящук А.А.*		
Специфікації вимог до розробленої системи	*Ящук А.А.*		
Розробка об'єкта проєктування	*Ящук А.А.*		
Нормоконтроль	*Ящук А.А.*		
Гарант ОП	*Ліщина Н.М.*		
Показник запозичень тексту	16%		
Академічна доброчесність	*Ящук А.А.*		

7. Дата видачі завдання «28» 02 2023р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів проєкту	Строк виконання етапів проєкту	Примітка
1	*Огляд літературних джерел по темі кваліфікаційної роботи бакалавра*	*05.02.2023 р.*	вик.
2	*Аналіз проблеми розробки та впровадження об'єкту проєктування*	*27.02.2023 р.*	вик.
3	*Обґрунтування вибору шляхів, технологій (алгоритмів) і засобів вирішення поставленого завдання*	*10.03.2023 р.*	вик.
4	*Розробка функціонально-структурної схеми роботи об'єкта проєктування та проєктування бази даних*	*17.04.2023 р.*	вик.
5	*Практична реалізація об'єкта проєктування та розробка бази даних*	*18.05.2023 р.*	вик.
6	*Тестування та налагодження об'єкта проєктування*	*01.06.2023 р.*	вик.
7	*Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедру*	*08.06.2023 р.*	вик.

Здобувач вищої освіти _____ (підпис) Михалевич Н.Д. (прізвище та ініціали)

Керівник кваліфікаційної роботи _____ (підпис) Ящук А.А. (прізвище та ініціали)

Рецензія
на кваліфікаційну роботу

Здобувач вищої освіти: *Михалевич Назар Дмитрович*

Тема: *«Комплексна система обліку продукції підприємства на базі технології .Net*

Коротка характеристика кваліфікаційної роботи та прийнятих рішень: *Кваліфікаційна робота бакалавра складається з трьох розділів, в яких проаналізовані проблематика та поставлені завдання за темою роботи, здійснено теоретичне дослідження та практичну реалізацію системи обліку продукції підприємства, розкрито суть експериментального дослідження результативності системи обліку продукції підприємства на базі технології .Net*

Самостійні розробки і пропозицій автора: *Автор самостійно створив та налаштував систему обліку продукції підприємства на базі технології .Net за допомогою використання новітніх технологій програмного забезпечення, сучасних підходів до написання коду програми та створення дизайну користувацького інтерфейсу.*

Практичне значення роботи *полягає в теоретичному обґрунтуванні методів проектування системи обліку продукції на базі технології .Net та створення системи обліку продукції для застосування її на дрібних підприємствах.*

Недоліки: *Істотних недоліків в роботі не виявлено.*

Загальний висновок: *У кваліфікаційній роботі бакалавра наведені результати вирішення актуального наукового завдання з обґрунтування обліку продукції підприємства на базі технології .Net. Робота є результатом самостійного наукового дослідження. Істотних недоліків не виявлено. Кваліфікаційна робота здобувача освіти Михалевича Назара Дмитровича рекомендується до захисту екзаменаційній комісії та заслуговує відмінної оцінки.*

Рецензент: *к.п.н. Саварин Павло Вікторович, доцент каф. ЦОЕТ*
(прізвище, ім'я, по-батькові, посада)

Рецензент кваліфікаційної роботи _____
(підпис)

«*8*» *червня* 202*3* р.

АНОТАЦІЯ

Михалевич Н. Д. Комплексна система обліку продукції підприємства на базі технології .Net. Рукопис.

Кваліфікаційна робота бакалавра за спеціальністю 121 Інженерія програмного забезпечення. Луцький національний технічний університет. Луцьк, 2023. 46 сторінок.

Кваліфікаційна робота присвячена розробці та дослідженню системи обліку продукції на базі технології .Net.

Робота складається з 3 розділів, в який проаналізовані проблематика та поставлені завдання за темою роботи, здійснено теоретичне дослідження та практичну реалізацію системи обліку продукції на базі технології .Net, розкрито суть експериментального дослідження результативності системи обліку. Запропоновані та розроблені конкретні інструменти для оптимізації обліку та легкого пошуку продукції.

Ключові слова: система обліку, візуальний інтерфейс, .Net, WEB Api, React, бібліотека, модель, сервіс.

ABSTRACT

Mykhalevych N. D. Complex product accounting system of the enterprise based on .Net technology. Manuscript.

Bachelor's qualifying work on specialty 121 Software engineering. Lutsk National Technical University. Lutsk, 2023. 46 pages.

The qualification work is devoted to the development and research of the product accounting system based on .Net technology.

The work consists of 3 sections, in which the issues are analyzed and tasks are set according to the topic of the work, theoretical research and practical implementation of the product accounting system based on .Net technology is carried out, the essence of the experimental study of the effectiveness of the accounting system is revealed. Specific tools have been proposed and developed to optimize accounting and easily search for products.

Keywords: accounting system, visual interface, .Net, WEB API, SVG, React, library, model, service.

ЗМІСТ

ВСТУП

Сучасні підприємства зіштовхуються з необхідністю ефективного обліку продукції для забезпечення своєчасного та точного контролю за виробничим процесом. Водночас, існуючі системи обліку продукції можуть бути недостатньо комплексними, не задовольняти всі потреби підприємства або не використовувати сучасні технології для максимальної ефективності. Одним із перспективних рішень для покращення систем обліку є використання технології .Net.

На сьогоднішній день існують деякі практичні розв'язки задач обліку продукції на підприємствах, але вони часто є неповними або неадаптованими до потреб конкретного підприємства. Існують прогалини знань у цій предметній галузі, особливо в контексті використання технології .Net для побудови комплексних систем обліку. Це створює потребу в проведенні досліджень та розробці нових підходів до створення комплексних систем обліку продукції на базі технології .Net.

Актуальність кваліфікаційної роботи полягає у необхідності розв'язання проблеми комплексного обліку продукції на підприємстві з використанням технології .Net. Здійснення досліджень та розробка нових підходів до створення комплексних систем обліку на базі .Net є перспективним напрямом, який може принести значний внесок у поліпшення ефективності виробничого процесу та управління на підприємствах.

Щоб реалізувати таку систему потрібно чітко ставити вимоги до розроблюваного продукту: необхідно спроектувати та розробити базу даних, забезпечити їх безпеку і безпеку сайту, реалізувати валідацію даних задля збереження їх цілісності. Для зручного користування такою системою також необхідний користувацький інтерфейс, вимоги до якого в сучасному світі теж дуже високі: дизайн має бути не перенавантажений та інтуїтивно зрозумілий, має бути присутня зміна теми інтерфейсу, реалізовано зручні форми для внесення

нових елементів в систему та перегляду вже існуючих даних, а також реалізована можливість розміщення екземплярів техніки у форматі SVG на мапі підприємства.

Отже, ця кваліфікаційна робота має на меті дослідити сучасний стан проблеми комплексного обліку продукції на підприємствах, проаналізувати світові тенденції вирішення цієї проблеми з використанням технології .Net, також розробити пропозиції та рекомендації щодо побудови комплексної системи обліку продукції на базі технології .Net для підприємства і розробити таку систему для певної категорії підприємств.

Метою дослідження є розробка веб-додатку на базі технології .Net, що міститиме мінімальний набір потрібних можливостей для ведення інвентаризації.

Об'єктом дослідження є розробка веб-додатку, який надасть можливість ведення інвентаризації продуктів для малого бізнесу. Веб-застосунок дуже зручний тим, що доступ до нього можна отримати як з мобільного телефону, так і з будь-якого браузера на комп'ютері. А в разі потреби чи забаганки його можна перевести в нативний додаток для IOS чи Android.

Предметом дослідження є програмна реалізація Web API на базі платформи ASP.Net(dotnet) і користувацького веб-інтерфейсу, використовуючи базовий стек веб-технологій, фреймворк ReactJS і додаткові бібліотеки для нього.

Завдання на кваліфікаційну роботу бакалавра:

- розробити базу даних;
- забезпечити безпеку та контрольований доступ до розроблюваної системи;
- забезпечити валідацію даних;
- створити користувацький інтерфейс для роботи з системою;
- забезпечити можливість розміщення даних на карті підприємства.

Новизна роботи. Існують безкоштовні та open-source рішення, що здатні покрити конкретні задачі, але вони або не працюють в якості веб-застосунку, або важкі у налаштуванні. Тому розробка саме такого додатку має шанс зайняти свою нішу і стати комусь в нагоді. Принаймні на початку ведення бізнесу для закриття базових потреб інвентаризації.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАНЬ НА КВАЛІФІКАЦІЙНУ РОБОТУ

1.1 Аналіз сучасного стану проблеми

На даному етапі розвитку інформаційних технологій і принципів побудови бізнесу обов'язково використовується те чи інше програмне забезпечення, що дозволяє зручно та ефективно вносити та накопичувати інформацію по конкретному напрямку роботи. Це, до прикладу, може бути:

- система обліку вхідного та вихідного товару для бізнесу, що займається закупівлями та продажем будь-якого товару;
- система обліку техніки, яка є у власності компанії, де є можливість закріплювати конкретний екземпляр за конкретним працівником і таким чином знати хто за що несе відповідальність;
- система обліку працівників та контролю доступу;
- інші вузькоспеціалізовані системи збору та контролю інформації.

Але у наявних на ринку продуктів частіше за все є головний мінус – це рішення для великих компаній, які коштують дорого. Малий бізнес не може собі дозволити витрачати в середньому 1500-2000 доларів в рік на програму, функціонал якої буде використовуватись у кращому випадку на 20%.

Серед програм для інвентаризації техніки та ведення моніторингу мереж є наступні гіганти:

- Lansweeper (від 2000 доларів на рік);
- PDQ (в середньому 1500 доларів на рік);
- Device42 (від 1500 доларів на рік лише за ліміт у 100 активів).

Мають надзвичайно широкий функціонал, високий рівень автоматизації та можливість інтеграції з іншими системами, що використовуються протягом багатьох років в ентерпрайз сегменті (наприклад, Active Directory). Якщо компанія заробляє достатньо, то мати серед своїх інструментів щось із наведених

вище програмних продуктів – обов’язковий елемент. Проте малий бізнес готовий миритись з певними недоліками, нюансами та відсутністю автоматизації, щоб не витрачати колосальні кошти на даний напрямок програм. Тому розробка менш функціонального додатку, але з наявністю мінімальних вимог до обліку техніки – виглядає цілком логічним рішенням в даній ситуації.

На основі цього невеликого дослідження ринку було прийнято розробити такий додаток. Для зручності розробки та масштабування проєкту в мабутньому, було вирішено розробити окремо Web API на базі платформи ASP.Net(dotnet) для доступу до бази даних та обробки запитів, а також веб-інтерфейс для користувача на базі стандартного веб-стеку (HTML, CSS, JavaScript) з використанням фреймворку ReactJS і додаткових бібліотек до нього. Такий підхід до розробки дозволяє легко вносити зміни в серверну частину, модифікувати API та в разі забаганки, наприклад, замовника переписати користувацький інтерфейс без необхідності торкатись безпосередньо коду, що відповідає за обробку даних і зв’язок з базою.

1.2 Постановка завдання на кваліфікаційну роботу бакалавра

Метою дослідження є розробка веб-додатку на базі технології .Net, що міститиме мінімальний набір потрібних можливостей для ведення інвентаризації.

Оскільки в розробку цього проєкту входить окремо Back-end та Front-end частини, доцільно буде розділити завдання окремо для кожного напрямку.

Для розробки Back-end частини веб-застосунку ми маємо реалізувати наступне:

- розробити базу даних, використовуючи NoSQL підхід та платформу MongoDB;
- забезпечити регульований доступ сайтів до розробленої API та безпеку даних;

- забезпечити доступність з будь-яких пристроїв та мереж, де буде відкриватись сайт, розмістивши його у вільному доступі в мережі Інтернет;
- забезпечити валідацію даних перед збереженням в базі даних.

Для успішної реалізації Front-end частини необхідно наступне:

- створити зручний, зрозумілий дизайн сайту з можливість змінювати тему інтерфейсу;
- створити форми для додавання нових елементів системи та редагування існуючих;
- забезпечити можливість розміщення екземплярів техніки в форматі SVG на карті підприємства для розуміння їх точного місцезнаходження.

Висновки до розділу 1

У розділі було виділено критерії для порівняння існуючих аналогічних або схожих програмних продуктів. Визначено мету дослідження. Спираючись на досліджені нами дані було поставлено завдання до кваліфікаційної роботи бакалавра.

РОЗДІЛ 2
СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проєктування програмного забезпечення

Ціль моєї розробки – забезпечити зручний базовий функціонал для успішного ведення процесу інвентаризації техніки для малого бізнесу у вигляді веб-застосунку.

Функціональні вимоги

Ключовою механікою роботи з цією системою є внесення даних та відображення їх користувачу в різному вигляді, тому ми можемо передбачити дії користувача: додавання новий даних, редагування вже наявних даних та видалення даних з системи.

Оскільки система на даному етапі розробки бере за основу інвентаризацію техніки, то слід також обмежити і види техніки. Наразі це будуть: комп’ютери, IP-телефони та принтери.

Обов’язкові поля бази даних для збереження інформації відображено на рисунку 2.1.

Нефункціональні вимоги

Майбутній веб-застосунок позиціонується як інструмент саме для маленьких компаній, проте підхід і вибрані засоби для розробки передбачають можливість масштабування проєкту в майбутньому. Також для повноцінної роботи застосунку та забезпечення хорошого користувацького досвіду потрібен не тільки візуально приємний інтерфейс, а й також висока продуктивність та швидкодія всієї системи. В нашому випадку швидкодія напряму залежить від швидкості роботи API, тому потрібно вибирати сервер для хостингу з хорошою пропускною здатністю мережі [1].

Вимоги до програмної системи є ключовим етапом у її розробці, оскільки вони гарантують якість, надійність та безперебійну роботу системи, а також

задовольняють потреби користувачів. Детальна розробка та виконання вимог є необхідною складовою успішної реалізації проєкту програмної системи.

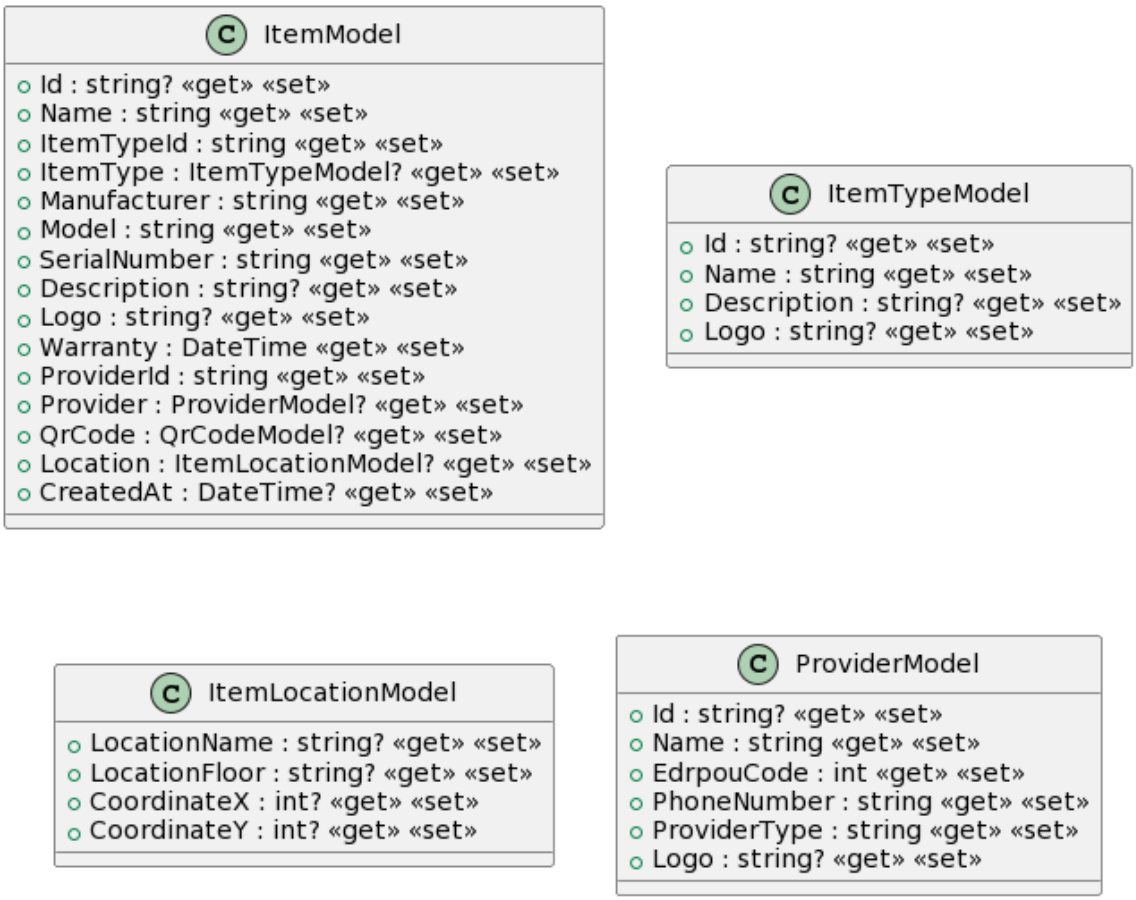

Рисунок 2.1 – Діаграми основних класів програми

2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання

Для розробки комплексного веб-застосунку було обрано два головних напрямки: розробка Web API на базі платформи ASP.Net і розробка користувацького веб-інтерфейсу із застосуванням фреймворку React.js.

Аналіз обраних засобів та інструментів розробки

Для розробки Web API використовувалось середовище розробки ПЗ Visual Studio 2022 Community, оскільки має багато вбудованих інструментів для роботи з кодом, його тестуванням та відслідковуванням помилок. Має підсвітку

синтаксису, функцію автодоповнення коду та пряму інтеграцію з платформою GIT для керування версіями ПЗ. Головною перевагою використання Visual Studio 2022 Community є те, що вона надає можливість студентам та некомерційним організаціям використовувати цю версію IDE безкоштовно для навчання, дослідження та некомерційної розробки програмного забезпечення, що цілком підходить під вимоги кваліфікаційної роботи бакалавра. Зокрема через те, що вибір мови програмування та платформи для розробки випав саме на ASP.Net (C#), робота з якою в середовищі Visual Studio реалізована ледь не найкраще та найзручніше серед інших IDE, таких як JetBrains Rider, ми зупинились саме на ній [2].

Для розробки інтерфейсу із застосуванням React.js було використано редактор коду Visual Studio Code. Головною його перевагою є велика кількість розширень, з допомогою яких можна підлаштувати редактор для себе, встановити підтримку синтаксису різних мов програмування та зручні інструменти, наприклад, для перегляду даних в БД. Наразі це найбільш функціональний редактор коду для написання веб-застосунків з використанням фреймворку React [3]. В ньому доступні розширення, що дозволяють нам отримати підсвічування синтаксису JSX коду, як показано на рисунку 2.2, автодоповнення коду React-компонентів а також візуальні індикатори помилок та попереджень.

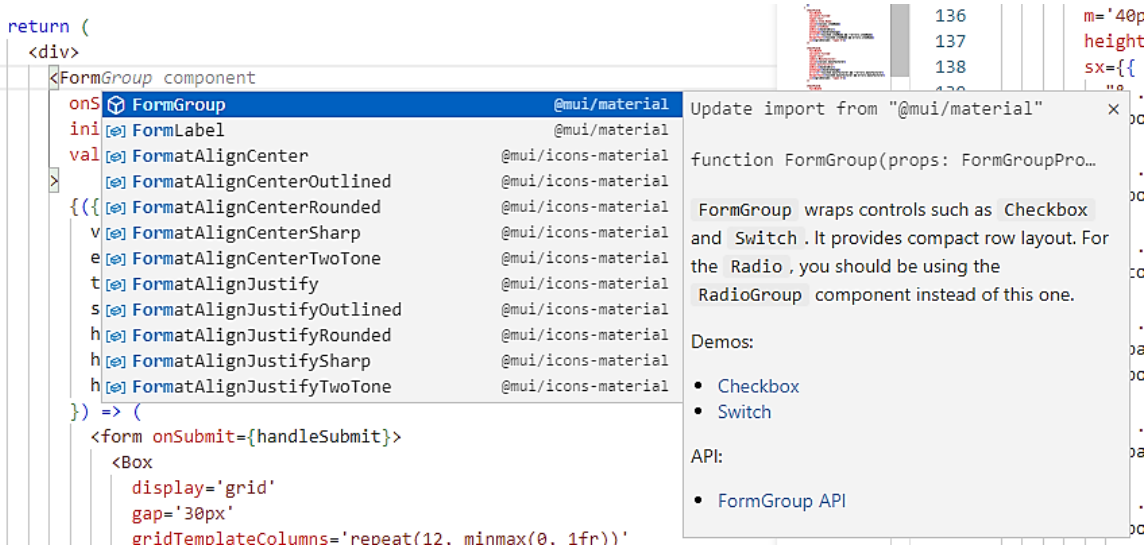

Рисунок 2.2 – Автодоповнення коду React-компонентів

Visual Studio Code підтримує сучасний синтаксис JavaScript, включаючи ES6+ функціональність, таку як стрілкові функції, розпилювання операторів і класи [4]. Це полегшує написання сучасного JavaScript-коду в React-проектах.

При виборі бази даних вибір впав на MongoDB. Це одна з найпопулярніших нереляційних (NoSQL) баз даних, яка зберігає дані у вигляді документів у форматі JSON (JavaScript Object Notation) з динамічною схемою [5]. MongoDB не вимагає фіксованої схеми бази даних, що дозволяє легко змінювати структуру даних без переробки всіх наявних записів. Це особливо корисно в проєктах, де схема даних може змінюватися з часом або варіюватися між різними записами. Ця база даних надає високу продуктивність завдяки своїй архітектурі, яка дозволяє швидко зберігати, оновлювати та отримувати дані. Вона підтримує індекси для прискорення запитів, а також може кешувати часто використовувані дані в оперативній пам'яті. MongoDB має велику кількість драйверів для різних мов програмування, що дозволяє легко інтегрувати її в сучасні додатки. Вона також підтримує різні формати даних, включаючи BSON (розширена версія JSON), що дозволяє зберігати складні дані, такі як дати, бінарні файли та інші [6].

Опис інструментів для вирішення конкретних завдань розробки

React дозволяє легко керувати станом компонентів та взаємодіяти з ними за допомогою useState Hook [7], що робить його ідеальним для створення форм для введення даних, використовуючи контрольовані компоненти. Як працює useState Hook показано на рисунку 2.3.

Контрольовані компоненти зберігають дані в потрібному стані компонента React, і відображають їх відповідно до цього стану. Якщо користувач вводить дані в поле введення, стан контрольованого компонента оновлюється, що дозволяє підтримувати синхронізацію між відображенням користувацького інтерфейсу та даними, які вводить користувач. Однією з переваг використання контрольованих компонентів є можливість валідації введених даних перед відправкою їх на сервер [8]. React дозволяє створити функції перевірки даних, які можна викликати при

надсиланні форми. Це дозволяє забезпечити правильність введених даних та запобігти надсиланню неправильних або недостатніх даних.

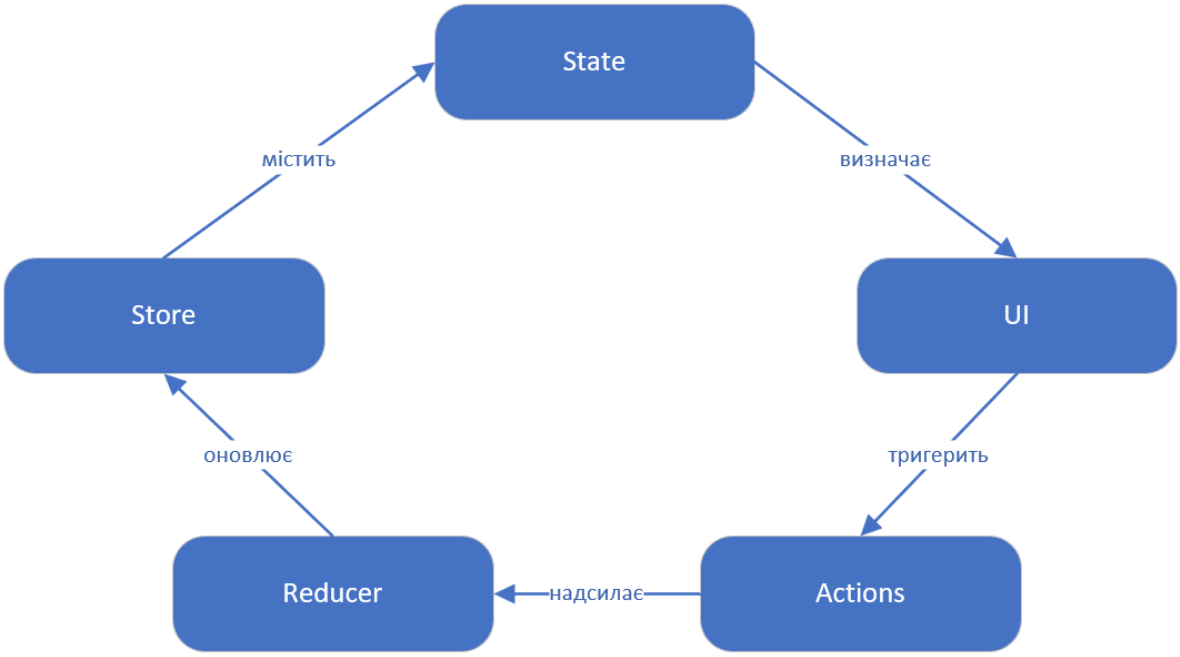

Рисунок 2.3 – Принцип роботи useState Hook в React

Завдяки тому, що фреймворк є open-source рішенням, для нього існує безліч абсолютно різних сторонніх бібліотек, які створюються «користувачами для користувачів». Багато з них виконують одну і ту ж задачу, проте з різним підходом, тому можна легко вибрати найбільш зручні та потрібні. Для роботи з формами є чимало бібліотек, наприклад Formik, Redux-Form, React-Hook-Form.

Formik – це бібліотека для React, яка дозволяє легко управляти формами на стороні клієнта. Вона надає ряд корисних функцій для створення інтерактивних форм, таких як автоматична валідація [9], контроль стану введення користувача та ін. Завдяки Formik, розробникам не потрібно створювати власний код для кожної форми, що дозволяє економити час та зусилля. Однією з вагомих переваж цієї бібліотеки є простий інтерфейс та широкі можливості налаштування, які цілком здатні задовольнити потреби як при створенні простих форм, так і при створенні комплексних форм з різними типами полів. Другою вагомою перевагою можна

назвати те, що Formik підтримує відстеження стану введення користувача, що дозволяє зберігати дані форми в реальному часі та відображати валідаційні повідомлення.

Однак, Formik має свої недоліки. Зокрема, він може бути складним у використанні для новачків, особливо якщо потрібно створювати складні форми з великою кількістю полів. Також, деякі розробники можуть знайти Formik надто обмеженим в плані налаштування та можливостей, особливо у порівнянні з іншими бібліотеками, такими як Redux-Form.

Redux-Form, як і Formik, зосереджується на управлінні формами в React, проте він має деякі особливості. Redux-Form заснований на Redux [10], бібліотеці для керування станом додатків в React. Це означає, що Redux-Form забезпечує можливість зберігати стан форми у Redux-сторі, що може бути корисно для складних форм з багатьма полями. Крім того, Redux-Form надає багато готових компонентів для відображення різних типів полів та розширену підтримку валідації даних. Використання Redux-Form може бути більш складним для новачків, оскільки він потребує розуміння Redux-стору та додаткового коду для налаштування форми. Крім того, Redux-Form не підтримує асинхронну валідацію даних, що часто може стати причиною оминути саме цю бібліотеку.

React-Hook-Form, на відміну від Formik та Redux-Form, не використовує компоненти класів та заснований на хуках React. Це означає, що React-Hook-Form дозволяє управляти формами без створення додаткових класів та компонентів. Він також надає просту підтримку валідації даних та має досить простий синтаксис.

Однак, React-Hook-Form має деякі обмеження, такі як обмежену підтримку для складних форм з багатьма полями та обмежені можливості налаштування компонентів. Крім того, він може бути складним для використання для розробників, які не мають досвіду роботи з хуками React.

Кожна з цих бібліотек має свої переваги та недоліки, і вибір конкретної бібліотеки буде залежати від потреб проєкту та досвіду програміста в роботі з React та JavaScript взагалі.

Вимоги до кваліфікаційної роботи бакалавра повністю покриває бібліотека Formik, тому для вирішення задачі з заповненням даних, їх валідацією та відображенням ми вибрали саме її.

При розробці API основною вимогою є швидкодія роботи та збереження цілісності даних, тому основна бібліотека, яку ми будемо використовувати називається MongoDB.Driver. Вона надає можливість підключитися до MongoDB сервера з використанням різних опцій налаштування, таких як адреса сервера, порт, облікові дані, аутентифікація та SSL. MongoDB.Driver має вбудовану підтримку маппінгу об'єктів C# на документи MongoDB. Ми можемо використовувати атрибути або конфігураційні файли для визначення відповідності між класами C# і колекціями MongoDB. Що дуже важливо, бібліотека має вбудовану підтримку синтаксису LINQ для C#, що дозволяє нам створювати складні але водночас гнучкі запити до бази даних, використовуючи такі оператори запитів, як Where, Select, OrderBy, Join, Find та інші. Приклад такого запиту наведено в лістингу 2.1.

Лістинг 2.1 – Приклад запиту до бази даних з використанням LINQ

```
var ordersCollection = database.GetCollection<Order>("orders");
var customersCollection = database. Get Collection<Customer>("customers");

var result orders Collection. AsQueryable()
  .Join(customersCollection. AsQueryable(), o => o. CustomerId, c => c. Id, (o, c) => new
    {
      OrderId = o.Id,
      CustomerName = c. Name,
      Order Total= o. TotalAmount
    })
  .ToList();
```

кінець лістингу 2.1

Оскільки наш проєкт передбачає можливість масштабування в майбутньому, інструменти, що ми використовуємо, мають мати запас по функціоналу, аби можна було в подальшому додати нові можливості, і MongoDB.Driver їх має. Зокрема це:

- підтримка транзакцій для забезпечення консистентності даних при виконанні кількох операцій;
- підтримка виконання складних агрегаційних запитів до MongoDB: ми можемо використовувати пайплайни агрегації для групування, фільтрації, сортування та обчислення статистики за допомогою різноманітних операторів;
- підтримка реплікацій та розподілених операцій: бібліотека надає функціональність для роботи з реплікацією та розподіленими операціями в MongoDB кластері, можна налаштовувати і керувати репліками та балансуванням даних.

Для регульованого доступу сайтів до розробленої API ми використаємо інструмент CORS. Cross-Origin Resource Sharing – це механізм, який забезпечує безпеку веб-додатків, що працюють у середовищі браузера і використовують JavaScript для звернення до ресурсів з інших доменів (origin). За замовчуванням браузери застосовують політику одного походження (same-origin policy), що означає, що JavaScript-код може звертатися до ресурсів лише з того ж самого домену, протоколу та порту, з якого була завантажена сторінка. Це зроблено для запобігання атакам зловмисників, які намагаються використовувати скрипти на інших доменах для отримання конфіденційної інформації або зламу системи.

Коли браузер робить запит до ресурсу на іншому домені, він спочатку надсилає запит OPTIONS (Preflight Request), щоб перевірити, чи сервер дозволяє доступ з вказаного домену. Сервер повинен відповісти з заголовками, що містять дозволи (Access-Control-Allow-Origin, Access-Control-Allow-Methods, тощо). Якщо сервер підтверджує дозвіл, то браузер здійснює основний запит (GET, POST, PUT, DELETE) до ресурсу. CORS використовується для контролю доступу до ресурсів

на інших доменах і дозволяє серверам точно налаштовувати, які домени мають доступ до їх ресурсів. Це зроблено для забезпечення безпеки та захисту конфіденційної інформації. ASP.Net WEB Api має нативну підтримку механізму CORS і зручне налаштування (лістинг 2.2).

Лістинг 2.2 – Приклад використання CORS в ASP.Net

```
var MyAllowSpecificOrigins = "_myAllowSpecificOrigins";

var builder = WebApplication.CreateBuilder (args);

builder.Services.AddCors (options =>
{
    options.AddPolicy (MyAllowSpecificOrigins, policy =>
        {
            policy. WithOrigins ("http://example.com", "http://www.contoso.com");

        });
});

builder.Services.AddControllers();
var app = builder.Build();
app.UseHttpsRedirection();
app.UseStaticFiles();
app.UseRouting();
app.UseCors (MyAllowSpecificOrigins);
app.UseAuthorization();
app.MapControllers();
app.Run();
```

кінець лістингу 2.2

Висновки до розділу 2

У цьому розділі було проаналізовано засоби та інструменти для розробки ПЗ, а також розглянуто декілька інструментів, що будуть використовуватись задля вирішення конкретних задач під час розробки. Було проведено порівняння з аналогами, сформовано список переваг вибраних нами інструментів і причин, чому було вибрано конкретно одні, а не інші.

РОЗДІЛ 3
РОЗРОБКА КОМПЛЕКСНОЇ СИСТЕМИ ОБЛІКУ ПРОДУКЦІЇ ПІДПРИЄМСТВА НА БАЗІ ТЕХНОЛОГІЇ .NET

3.1 Практична реалізація об'єкта проєктування

Для реалізації WEB API [11] на C# нам потрібно на початку продумати як буде виглядати структура нашого проєкту і як ми хочемо його реалізовувати. Оскільки основною задачею API є робота з БД, то структура коду буде будуватись довкола реалізації зручного доступу до написаних обгорток звернення до бази. Було вирішено розділити код на наступні компоненти:

- models – відповідають за опис моделей для бази даних, окреслюють як саме мають виглядати документи в MongoDB. На основі моделей будуються запити;
- services – сервіси по суті є набором написаних обгорток для запитів до бази;
- controllers – основна частина API, тому що саме тут ми прописуємо як саме виглядатимуть наші посилання для звернення до API, що звернення мають в собі містити і що мають давати у відповідь на успішний запит.

Наведемо приклад того, як виглядає код кожної з цих частин, щоб розуміти взаємозалежності. Приклад моделі наведений в лістингу 3.1.

Лістинг 3.1 – Модель «ItemModel»

```
using MongoDB.Bson;
using MongoDB.Bson.Serialization.Attributes;

namespace StorageProject.Api.Models
{
  public class ItemModel
  {
    [BsonId]
    [BsonRepresentation(BsonType.ObjectId)]
    public string? Id { get; set; }
    [BsonRequired]
    public string Name { get; set; }
    [BsonRequired]
    public string ItemTypeId { get; set; } = null!;
```

Продовження лістингу 3.1

```
    [BsonIgnore]
    public ItemTypeModel? ItemType { get; set; }

    [BsonRequired]
    public string Manufacturer { get; set; }
    [BsonRequired]
    public string Model { get; set; }
    [BsonRequired]
    public string SerialNumber { get; set; }
    public string? Description { get; set; }
    public string? Logo { get; set; }
    public DateTime Warranty { get; set; }

    [BsonRequired]
    public string ProviderId { get; set; } = null!;

    [BsonIgnore]
    public ProviderModel? Provider { get; set; }

    public QrCodeModel? QrCode { get; set; }

    public ItemLocationModel? Location { get; set; }

    public DateTime? CreatedAt { get; set; } = DateTime.UtcNow;
  }
}
```

кінець лістингу 3.1

У приведеному вище лістингу коду ми можемо побачити структуру документу бази MongoDB, яка була описана у функціональних вимогах 2 розділу. Тепер щоб проводити якісь операції з цією моделюю документу нам потрібно описати обгортки запитів до бази даних [12]. Наведемо приклад того, як виглядають запити до БД, що базуються на моделі ItemModel (лістинг 3.2).

Лістинг 3.2 – Сервіс «ItemService»

```
using Microsoft.Extensions.Options;
using MongoDB.Driver;
using StorageProject.Api.Configurations;
using StorageProject.Api.Models;
```

Продовження лістингу 3.2

```
namespace StorageProject.Api.Services
{
   public class ItemsService
   {
      private readonly IMongoCollection<ItemModel> _itemCollection;

      public ItemsService(IOptions<DatabaseSettings> databaseSettings)
      {
         var mongoClient = new MongoClient(databaseSettings.Value.ConnectionString);
         var mongoDb = mongoClient.GetDatabase(databaseSettings.Value.DatabaseName);

         _itemCollection =
mongoDb.GetCollection<ItemModel>(databaseSettings.Value.ItemsCollection);
      }
      public async Task<List<ItemModel>> GetAsync() => await _itemCollection.Find(_ =>
true).ToListAsync();

      public async Task<ItemModel> GetAsync(string id) =>
         await _itemCollection.Find(x => x.Id == id).FirstOrDefaultAsync();

      public async Task CreateAsync(ItemModel item) => await _itemCollection.InsertOneAsync(item);

      public async Task UpdateAsync(ItemModel item) => await _itemCollection.ReplaceOneAsync(x
=> x.Id == item.Id, item);

      public async Task RemoveAsync(string id) => await _itemCollection.DeleteOneAsync(x => x.Id ==
id);

   }
}
```

кінець лістингу 3.2

В цьому коді описані запити до бази даних для отримання інформації, додавання нових даних, редагування та видалення. Вже з наявними методами ми можемо реалізувати API і протестувати його роботу. Для цього ми створимо контролер, який буде відповідати за обробку безпосередньо запитів від користувача до БД. Лістинг коду такого контролера наведено в Додатку А. В контролері ми описуємо шаблон url для запиту (Route) і доповнюємо його перед оголошенням кожного методу, коли вказуємо який це має бути запит (GET, POST, PUT, DELETE).

API має бути обмежене у даних, які вона видає. Блок-схема, що зображена на рисунку 3.1, ілюструє принцип роботи методу GetAll, який поверне нам дані про всі наявні записи в колекції «items».

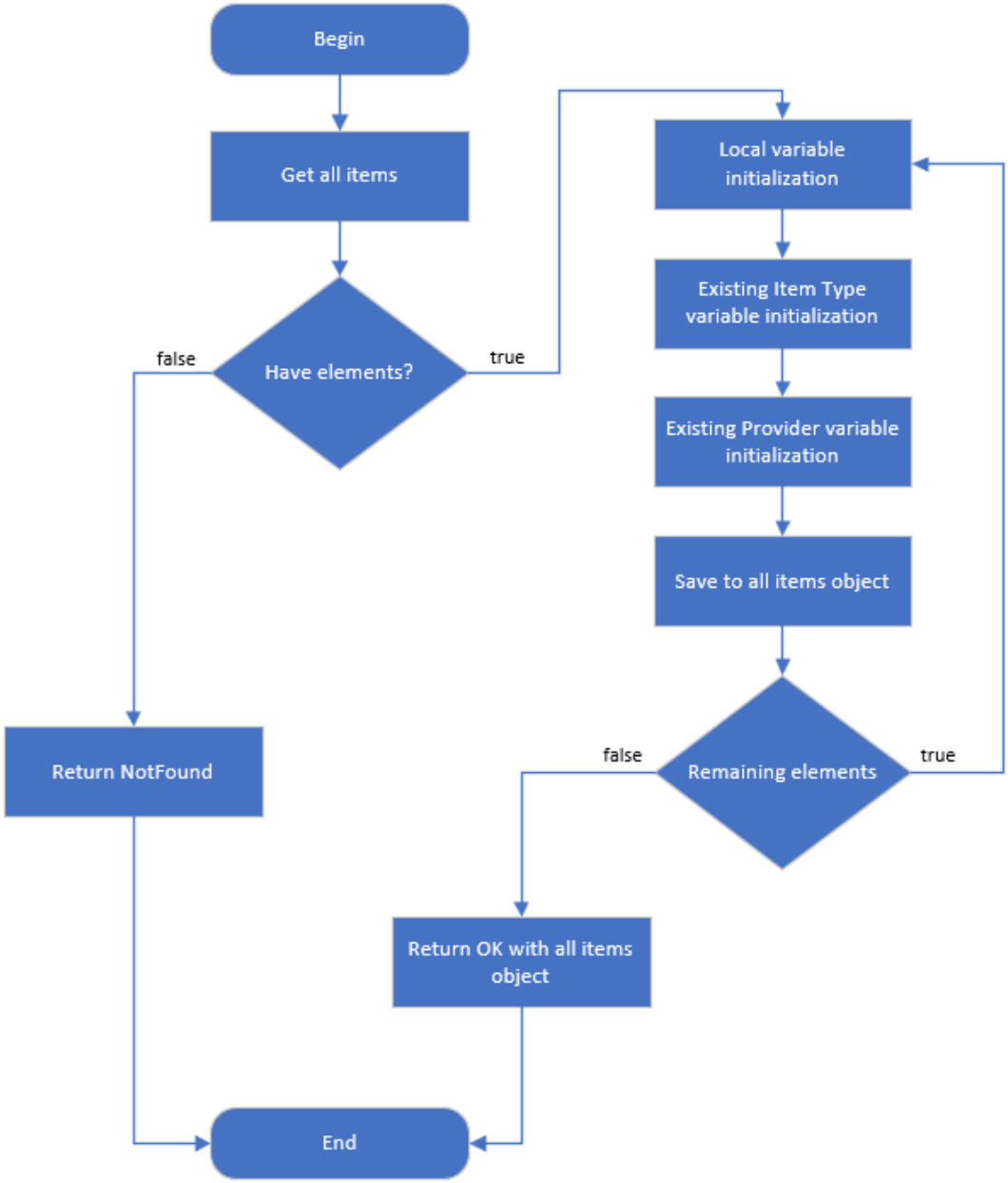

Рисунок 3.1 – Принцип роботи методу GetAll

Також, як приклад, наведемо принцип роботу методу Update у вигляді блок-схеми. Вона зображена на рисунку 3.2.

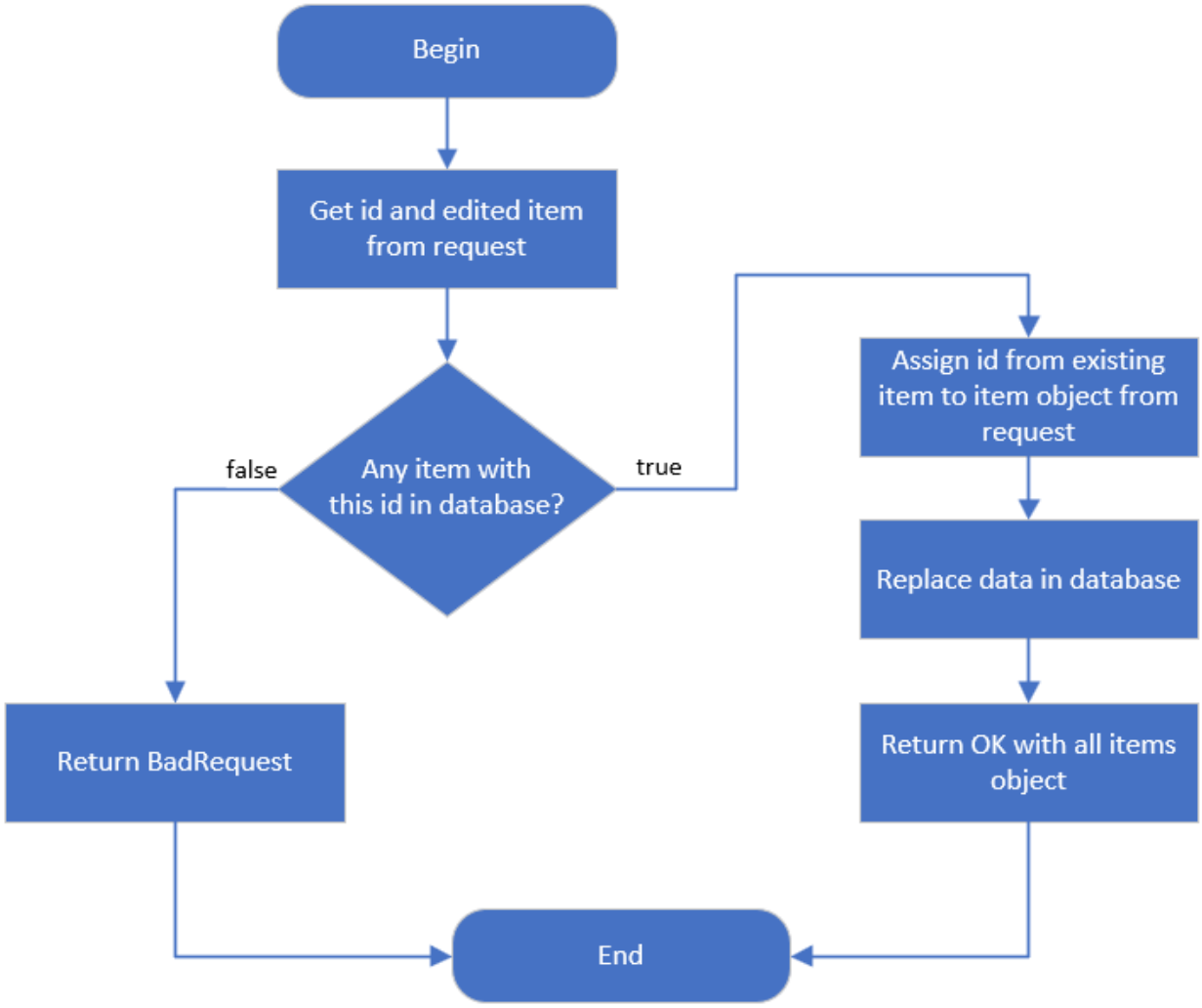

Рисунок 3.2 – Принцип роботи методу Update

Для зручного користування розробленим API було також спроєктовано та розроблено веб-інтерфейс. Із застосуванням фреймворку React JS. Відповідно до вимог, реалізуємо форму для додавання і редагування даних, можливість розміщення техніки на мапі та таблицю з доступом даними. Дизайн веб-застосунку і таблицю з даними показано на рисунку 3.3.

Зліва присутнє бокове меню для швидкого доступу до різних елементів сайту та інформація про активного користувача. Центральна частина зроблена великою, щоб вміщувати достатньо інформації. Інтерфейс не перенавантажений. В правій частині таблиці присутні кнопки для редагування та видалення. При натисканні на кнопку редагування нам відкриється вікно діалогу, де ми можемо

вносити зміни або просто переглянути інформацію в компактнішому вигляді. Як це виглядає показано на рисунку 3.4.

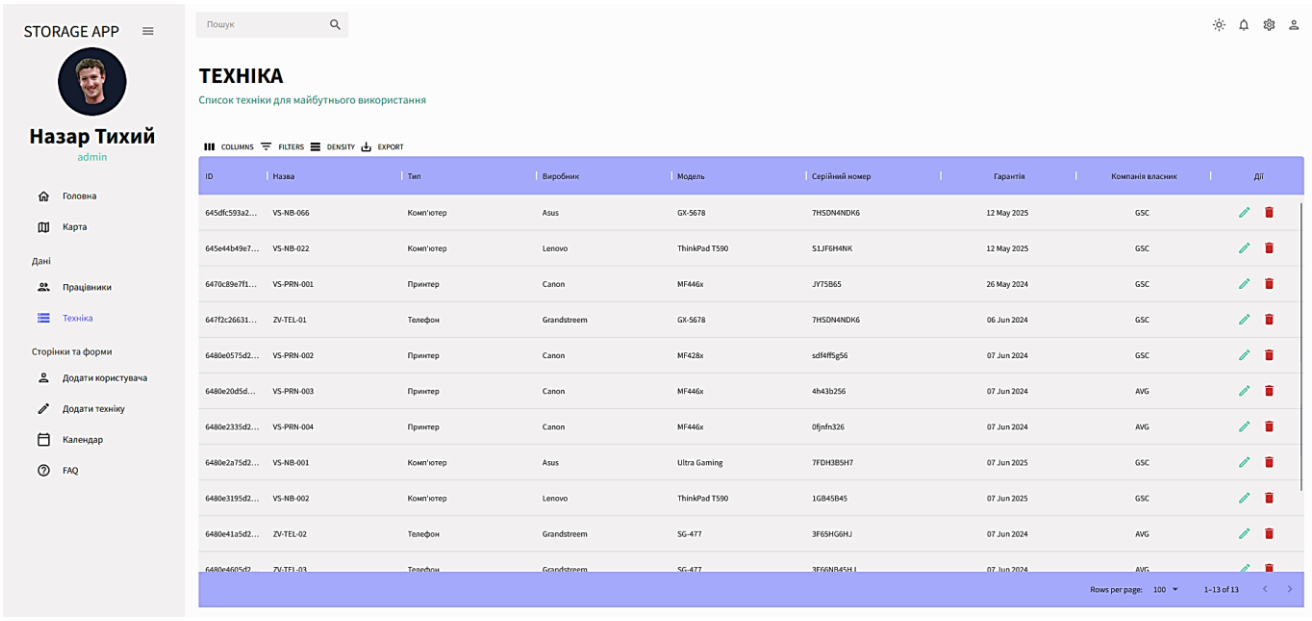

Рисунок 3.3 – Дизайн сайту та таблиці

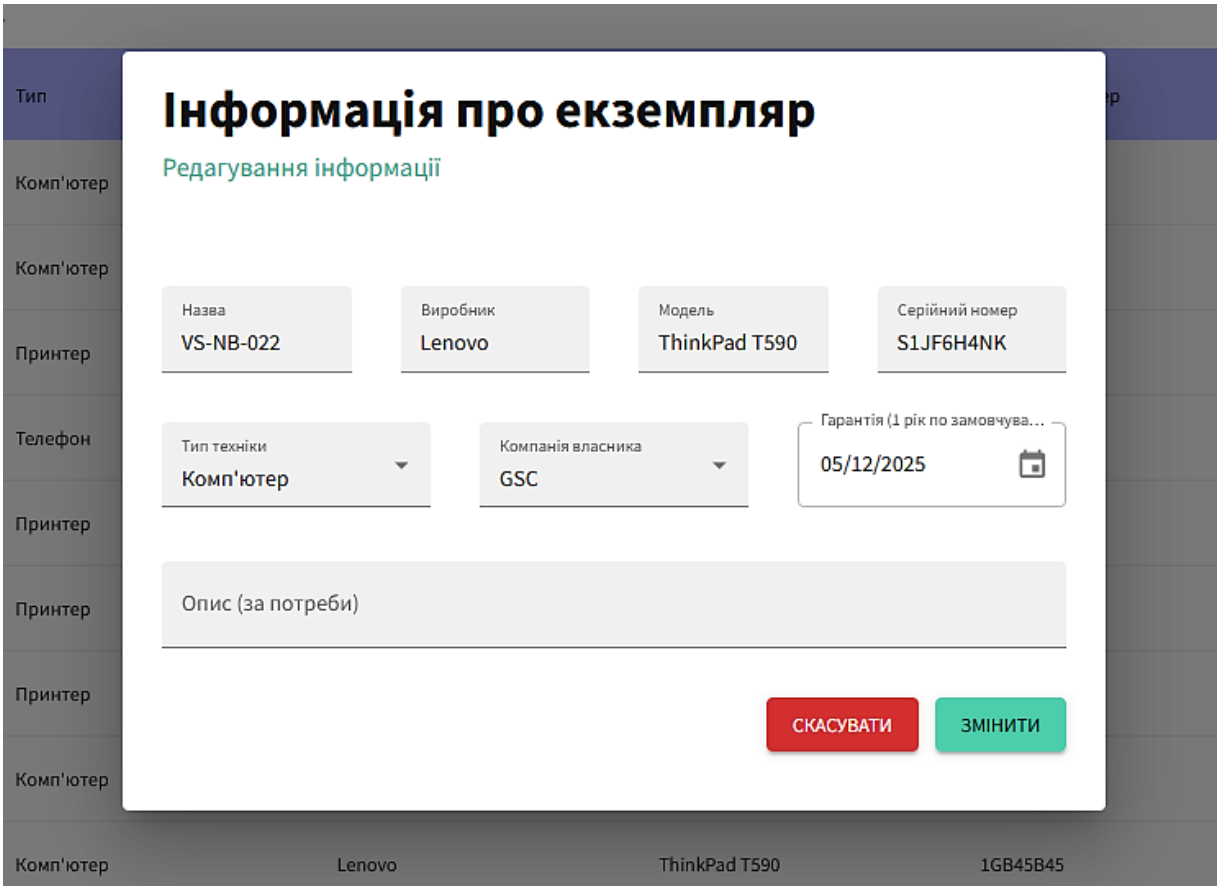

Рисунок 3.4 – Вікно редагування інформації

Форма для додавання нових екземплярів з валідацією даних на стороні клієнта має ті ж поля, що й форма для редагування (рис. 3.5).

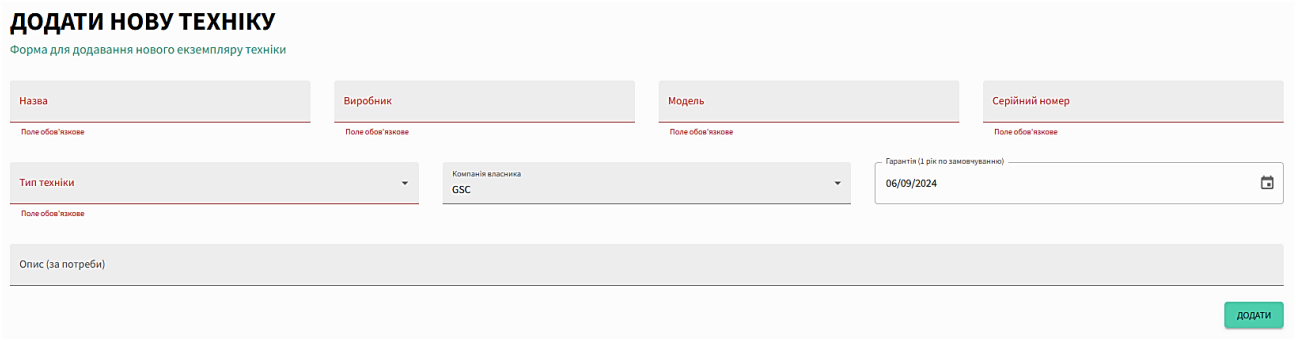

Рисунок 3.5 – Форма для додавання нових екземплярів техніки

Мапа із можливістю розміщення на ній екземплярів техніки показана на рисунку 3.6.

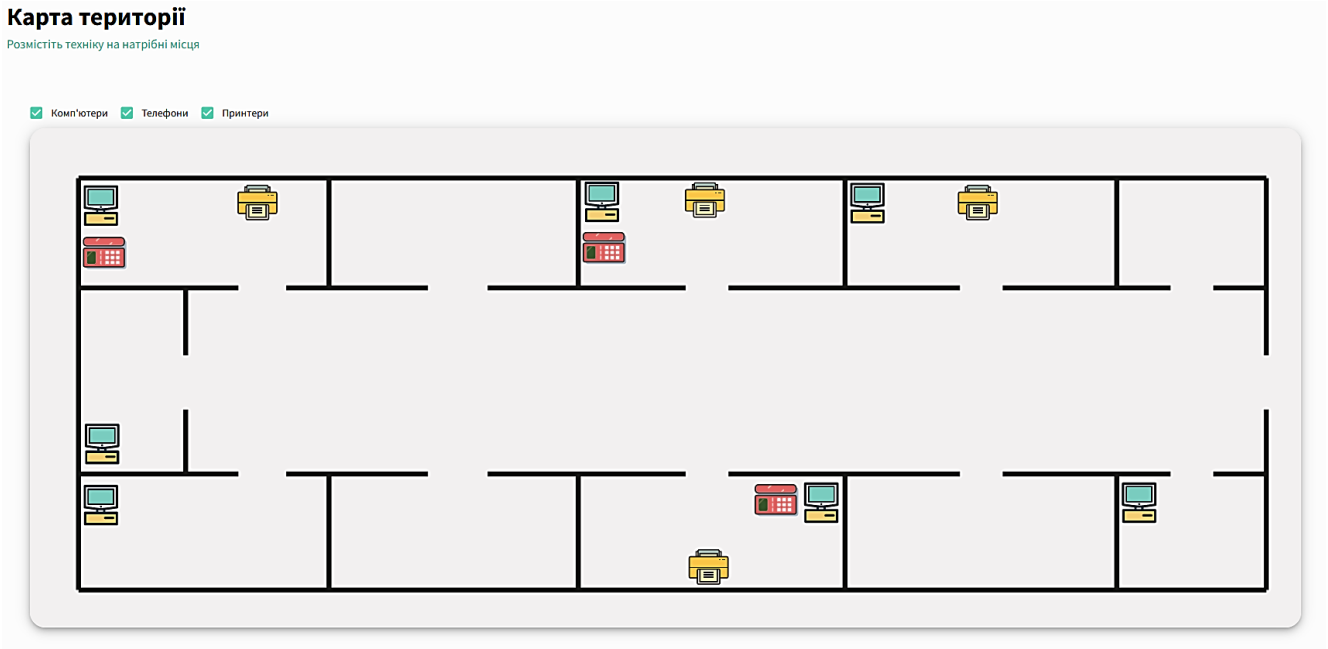

Рисунок 3.6 – Мапа території

На ньому ми також можемо побачити список з фільтрами у вигляді меню прапорців, які дозволяють нам приховати певний тип техніки та відображати лише потрібний нам наразі (рис. 3.7).

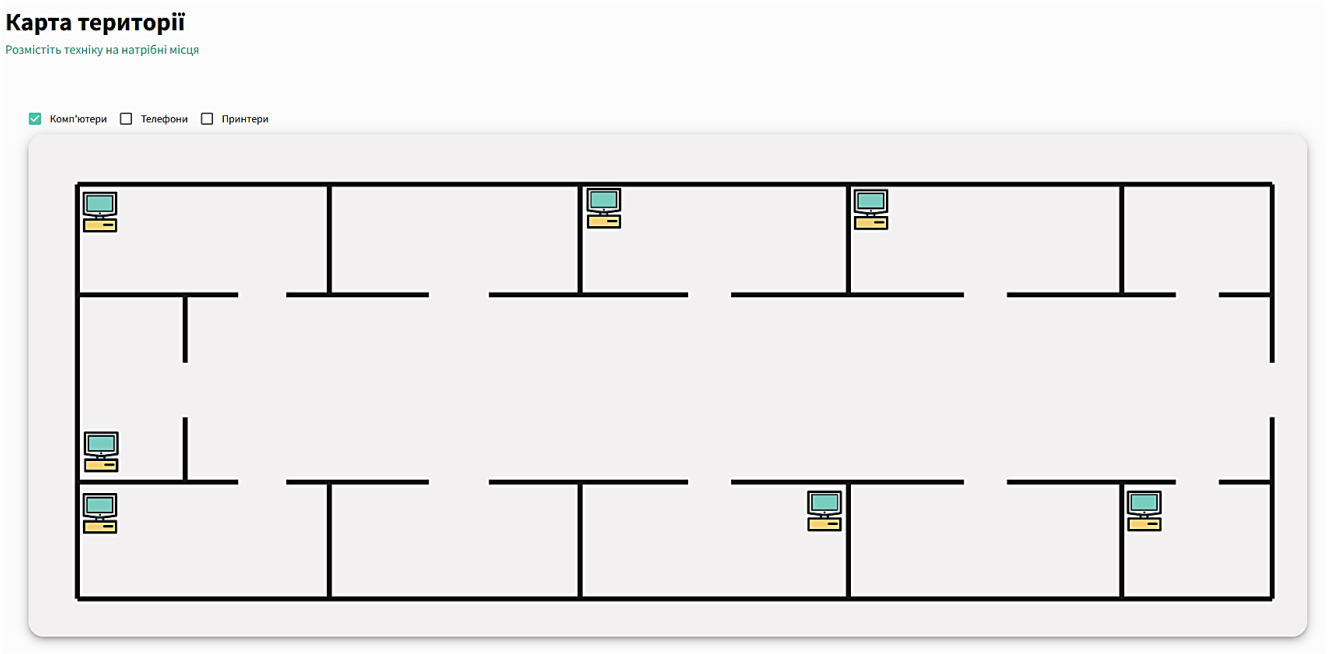

Рисунок 3.7 – Робота фільтру мапи території

Як реалізовано цей фільтр в коді програми показано в лістингу 3.3.

Лістинг 3.3 – Фільтр в компоненті карти

```
import Checkbox from "@mui/material/Checkbox";
import {
  FormControl,
  FormGroup,
  FormControlLabel,
  Box,
} from "@mui/material";
import {useState, useEffect} from "react";

const ItemTypesCheckbox = (props) => {
  const types = props.types.map((type) => ({
    ...type,
    checked: true,
  }));

  const [itemTypes, setItemTypes] = useState(types);

  useEffect(() => {
    props.onCheckBoxStatusChange(itemTypes);
  }, [itemTypes]);

  const updateCheckStatus = (event) => {
    const checkboxId = event.target.id;
```

Продовження лістингу 3.3

```
  setItemTypes((prevItemTypes) => {
    return prevItemTypes.map((checkbox) => {
      if (checkbox.id === checkboxId) {
        return {
          ...checkbox,
          checked: !checkbox.checked,
        };
      } else {
        return checkbox;
      }
    });
  });
 };

 return (
  <Box>
    <FormControl component='fieldset'>
      <FormGroup aria-label='position' row>
        {itemTypes &&
          itemTypes.map((type) => (
            <FormControlLabel
              key={type.id}
              label={`${type.name}и`}
              labelPlacement='end'
              control={
                <Checkbox
                  id={type.id}
                  value={type.id}
                  checked={type.checked}
                  onChange={updateCheckStatus}
                  color='secondary'
                />
              }
            />
          ))}
      </FormGroup>
    </FormControl>
  </Box>
 );
};

export default ItemTypesCheckbox;
```

кінець лістингу 3.3

3.2 Захист інформаційно-комп’ютерної системи

Для захисту розробленої API нам слід обмежити доступ до неї використавши механізм CORS [13], про який розповідалось в розділі «Опис інструментів для вирішення конкретних завдань розробки». Для реалізації політики CORS в коді розробленої API нам потрібно у файл Program.cs прописати код, наведений у лістингу 3.4. У дозволені IP ми додамо лише адресу з якої доступний наша Front-end частина.

Лістинг 3.4 – Реалізація політики CORS в коді WEB Api

```
builder.Services.AddCors(p => p.AddPolicy("CorsPolicy", build =>
{
   build.WithOrigins("https://storage-app-sxlb.onrender.com").AllowAnyMethod().AllowAnyHeader();
}));
var app = builder.Build();
app.UseCors("CorsPolicy");
```

кінець лістингу 3.4

Для захисту інформації ми зобов’язані використовувати валідацію даних зокрема й на стороні користувача. Ми використовуємо бібліотеку Formik для створення та керування формами для заповнення даних. Для неї також є «хороший компаньйон» у вигляді бібліотеки для валідації даних під назвою yup [14].

Yup дуже проста у налаштуванні але водночас має багатий функціонал. На даному етапі розробки в нас немає потреби робити якісь важкі валідації та перевірки, тому в лістингу 3.5 приведено код, який відповідає за валідацію даних у формах для створення та редагування.

Лістинг 3.5 – Валідація даних на стороні клієнта

```
const checkoutSchema = yup.object().shape({
  name: yup.string().min(2, "Надто коротке")
        .max(50, "Надто довге").required("Поле обов'язкове"),
  itemTypeId: yup.string().required("Поле обов'язкове"),
  itemProviderId: yup.string().required("Поле обов'язкове"),
```

Продовження лістингу 3.5

```
manufacturer: yup.string().required("Поле обов'язкове"),
  model: yup.string().required("Поле обов'язкове"),
  serialNumber: yup.string().required("Поле обов'язкове"),
});
```

кінець лістингу 3.5

3.3 Відомості щодо розгортання системи

Хостинг – невід’ємна частина будь-яких веб-застосунків. Не має значення чи це односторінковий сайт-візитівка, чи комплексний додаток з серверною частиною – для доступу до нього через мережу інтернет, його потрібно в мережі інтернет розмістити. Наразі сайти, написані з допомогою React, можна максимально легко розмістити в інтернеті. З WEB Api, написаному з допомогою мови C# трішки складніше, але розберемо все по-порядку.

Безпосередньо хостинг провайдерів, тобто платформ, на яких можна розмістити власний продукт, є дуже багато. Вони між собою відрізняються часто лише візуальною складовою, платними і безкоштовними варіантами, можливими потужностями та позиціонуванням на ринку. Потрібно підбирати під власні потреби. Наприклад, нелогічно купувати хостинг за 200 доларів, щоб розмістити на ньому зовсім невеличкий сайт без бази даних.

Веб-застосунки та API, написані на мові C#, складніше розмістити в мережі з однієї найважливішої причини – дана мова набагато менш популярна для написання веб-застосунків, ніж JavaScript, TypeScript чи PHP. Натомість додатки, написані на базі фреймворку React JS, розмістити в мережі дуже легко. Знову ж таки через вагому популярність та підтримку спільноти розробників.

Нам вдалось знайти таку платформу, де можна розмістити відразу як і Front-end частину, написану на React, так і Back-end, що написана на C#. Платформа має підтримку GitHub і це означає, що розміщені додатки автоматично оновлюються як тільки на GitHub з’являється оновлений код. Вона називається Render і далі буде проілюстровано й описано процес розгортання.

На рисунку 3.8 ми можемо побачити всі наявні можливості даної платформи. Нас насамперед буде цікавити Static Site і Web Service.

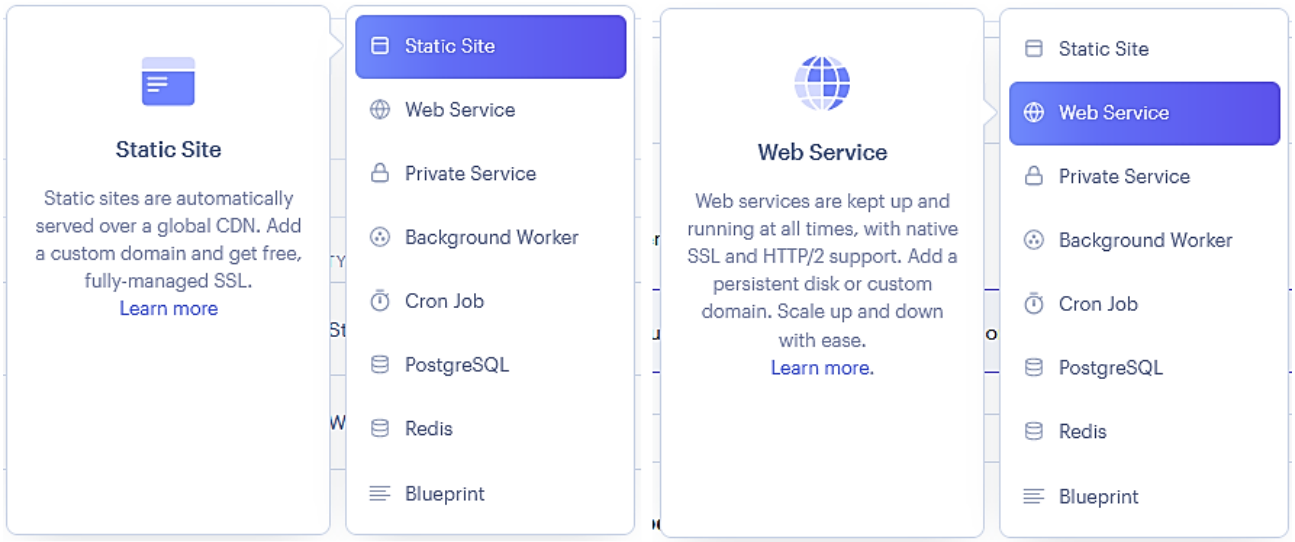

Рисунок 3.8 – Можливості платформи Render

Оскільки наша система є розділеною на дві частини і сайт з користувацьким інтерфейсом по суті є статичним – відображає інформацію користувачу – то послуга Static Site для нього підходить ідеально. WEB Api по своїй суті є сервісом, що обробляє інформацію – для його розміщення використаємо послугу Web Service.

Приєднавши до платформи свій GitHub аккаунт і натиснувши кнопку додати статичний сайт, перед нами з’явиться список всіх наших проєктів, нам залишається вибрати необхідний (рис. 3.9).

Далі нам потрібно дати сайті назву, вибрати з якої гілки GitHub отримувати оновлення автоматично, вказати команди для компілювання проєкту (рис. 3.10). Нам це не потрібно, оскільки на GitHub відразу розміщено лише проєкт, але якщо ситуація інакша і в одному репозиторії присутні окрім кореневої папки проєкту іще якісь, то потрібно платформі вказати шлях до кореневої папки у відповідному полі (Root Directory).

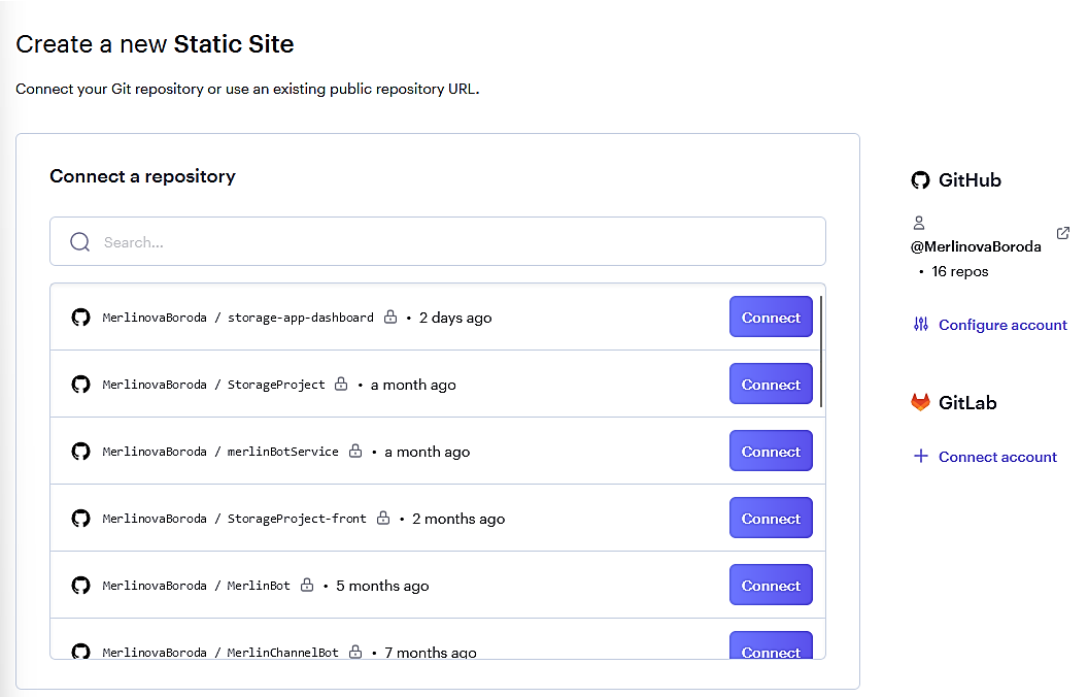

Рисунок 3.9 – Список доступних проєктів

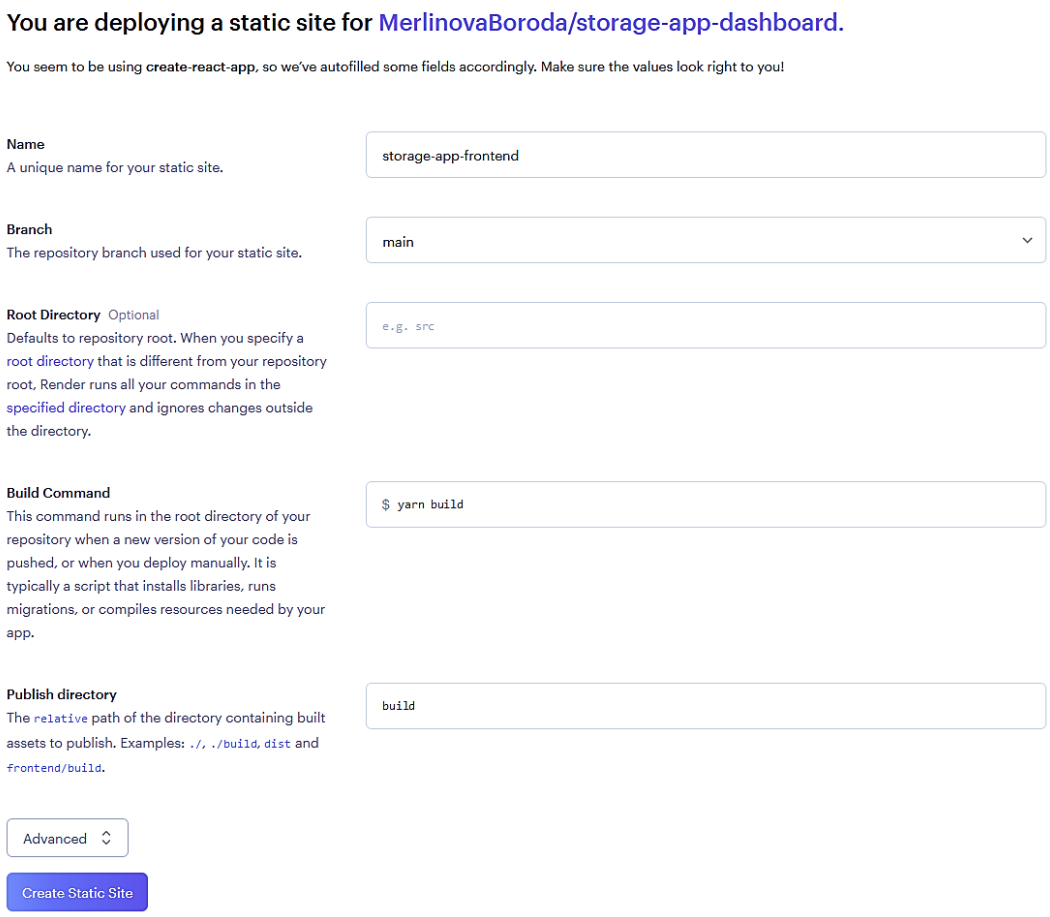

Рисунок 3.10 – Налаштування компілювання проєкту

Останнім кроком буде додати змінну середовища, яка використовуватиметься кодом програми (рис. 3.11) і правило перенаправлення (рис.3.12). Правило перенаправлення потрібне, оскільки в коді Front-end частини ми використовуємо React-Routes для надання окремим сторінкам url адреси [15].

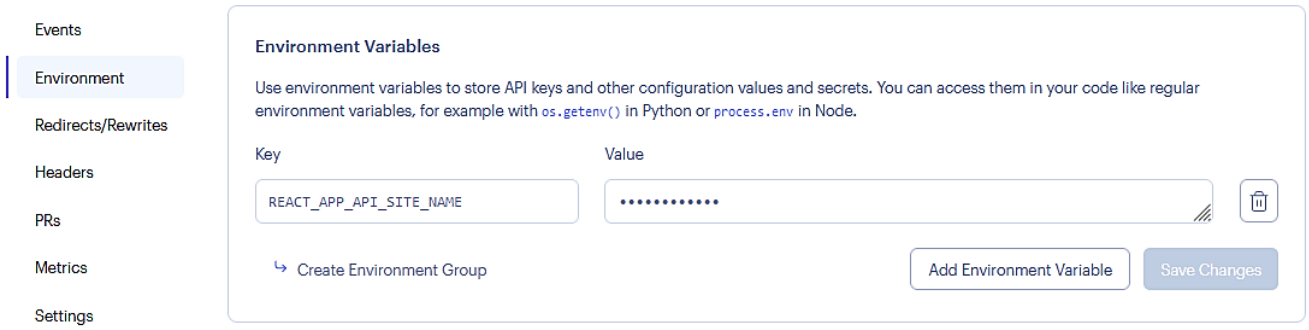

Рисунок 3.11 – Змінна середовища

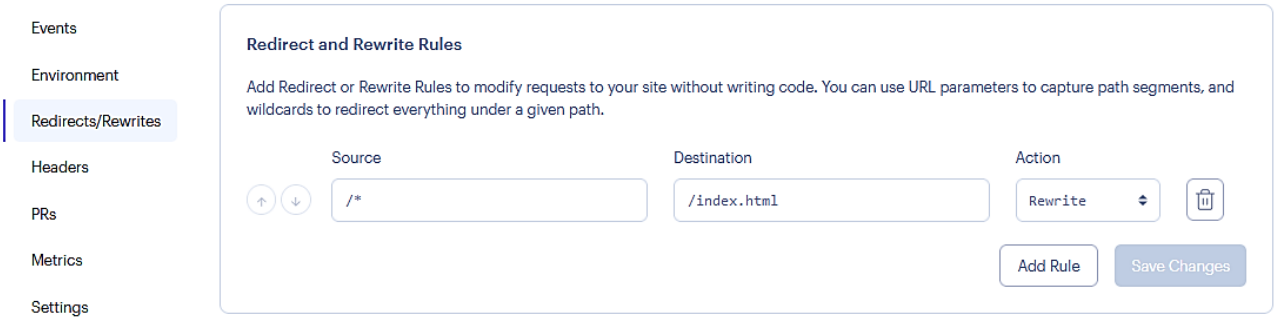

Рисунок 3.12 – Правило Redirect

Після цього нам необхідно лише власноруч запустити компілювання проєкту, щоб всі ці зміни набули значення і сайт стане доступним через зовнішню адресу, яку нам надасть платформа.

Перейдемо до розміщення на платформі нашої WEB Api. Тут буде трішки складніше, оскільки нативної підтримки мови C# тут немає. Натомість є підтримка Docker. Тому для того, щоб розмістити Api на хостингу, нам потрібно повернутись в проєкт і додати файл з назвою Dockerfile, код якого наведений в лістингу 3.6.

Лістинг 3.6 – код Dockerfile

```
FROM mcr.microsoft.com/dotnet/aspnet:6.0 AS base
WORKDIR /app
EXPOSE 80
EXPOSE 443

FROM mcr.microsoft.com/dotnet/sdk:6.0 AS build
WORKDIR /src
COPY ["StorageProjectAPI/StorageProject.Api.csproj", "StorageProjectAPI/"]
RUN dotnet restore "StorageProjectAPI/StorageProject.Api.csproj"
COPY . .
WORKDIR "/src/StorageProjectAPI"
RUN dotnet build "StorageProject.Api.csproj" -c Release -o /app/build

FROM build AS publish
RUN dotnet publish "StorageProject.Api.csproj" -c Release -o /app/publish /p:UseAppHost=false

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "StorageProject.Api.dll"]
```

кінець лістингу 3.6

Після того як запушимо зміни в GitHub, можемо переходити назад на платформу Render та налаштовувати наш сервіс (рис. 3.13). Тут нас так само зустрічає форма, де ми дамо назву сервісу, виберемо регіон, вкажемо гілку GitHub, з якої автоматично брати оновлення та виберемо середовище виконання, в нашому випадку це Docker.

Ми можемо помітити, що тут з’явився вибір потужностей. Звичайно за потреби можна перейти на більш потужні платі рішення платформи, але наразі нам вистачить безкоштовної.

Якщо все зроблено правильно, то за кілька хвилин Docker вже скомпілює наш проєкт і Api стане доступне за публічним посиланням, яке нам надасть платформа.

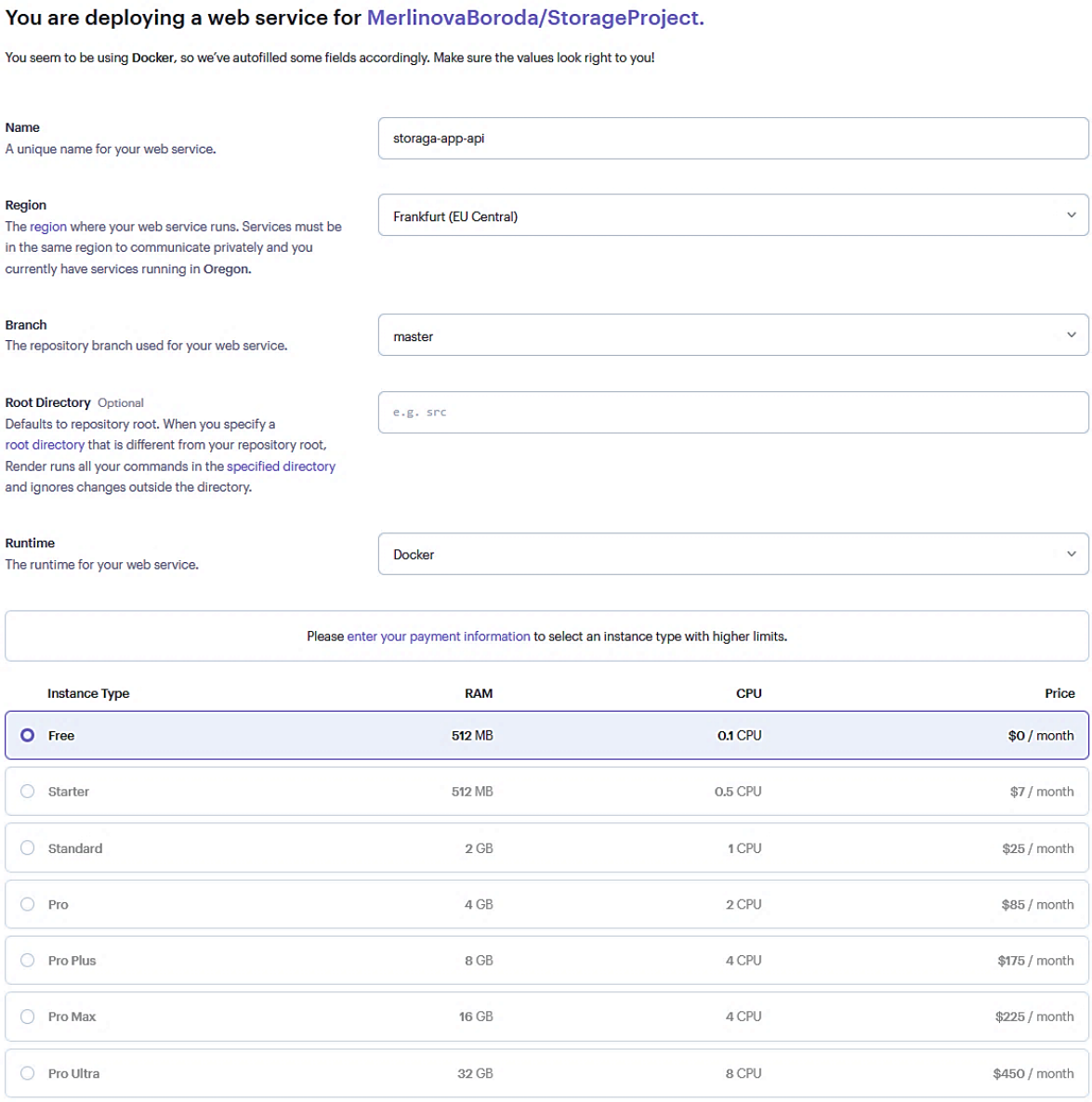

Рисунок 3.13 – Налаштування компілювання сервісу

Останнім кроком розміщення всієї нашої нашої системи буде база даних. Для безпеки та доступності було вирішено не розміщувати базу даних відразу поруч з додатком, а скористатись наявним у MongoDB веб сервісом для розміщення баз даних під назвою MongoDB Atlas (рис. 3.14). Все, що нам треба зробити – це зареєструватись та створити кластер. Після чого в коді програми, де ми декларуємо шлях для підключення до БД, вказати його назву.

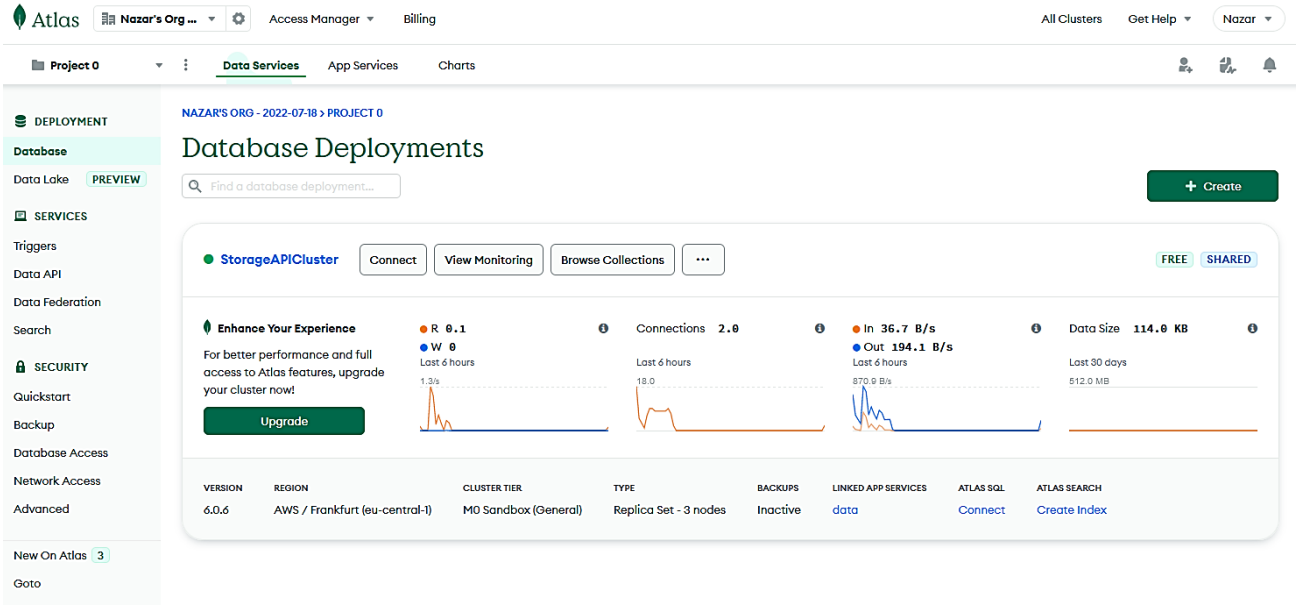

Рисунок 3.14 – Платформа MongoDB Atlas

Колекції ми в ній створювати не будемо, як і не будемо додавати дані безпосередньо, для цього в ми розробили користувацький інтерфейс. Але що ми тут обов'язково зробимо – це створимо окремого користувача, через якого розроблене Api буде взаємодіяти з даними, та обмежимо доступ до бази даних по мережі. Ми не зацікавлені в тому, щоб будь-хто міг мати доступ до Api або ж доступ до БД безпосередньо.

Платформа Render після створення та налаштування веб-сервісу надає статичні ip-адреси, які використовуватиме розміщений сервіс. Тому у вкладці Network Access додамо лише їх (рис. 3.15).

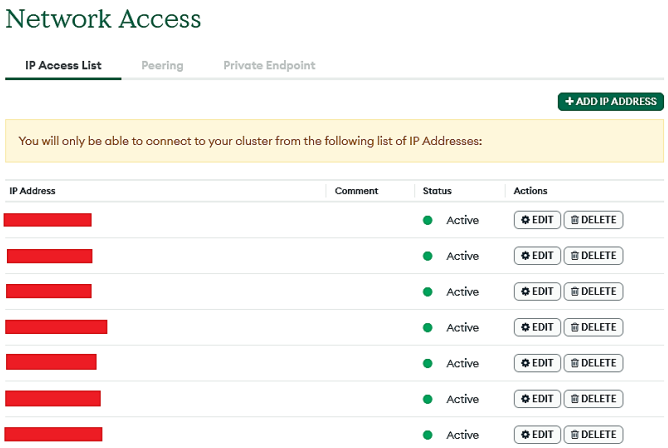

Рисунок 3.15 – Список IP-адрес, що мають доступ до БД

Висновки до розділу 3

У розділі було описано роботу розробленого проєкту, наведено лістинги, проілюстровано логіку роботи певних елементів веб-додатку, що відповідають вимогам, поставленим на початку роботи. Описано які методи захисту та збереження цілісності даних було використано. Детально описано процес розміщення розробленої системи на платформі веб-хостингу.

ВИСНОВКИ

У кваліфікаційній роботі було проведено розробку комплексної системи обліку продукції на базі технології .Net. Для реалізації поставлених задач було використано такі бібліотеки як: MongoDB.Driver, React-draggable, React-svg, Formik, yum та низка інших.

На початку роботи було проведено аналіз предметної області дослідження. Розглянуто аналоги та обґрунтовано основні засоби проєктування системи.

Для створення комплексної системи було використано платформу .Net для написання серверної частини і фреймворк React JS для написання користувацького інтерфейсу. Серверна частина реалізована у вигляді WEB Api, завдяки можливостям мови C# її вдалось зробити швидкою та захищеною. Front-end частина, що реалізовано з допомогою React JS, дала можливість створити швидкий уніфікований інтерфейс, використані бібліотеки розширили функціонал системи.

В ході розробки було виконано поставлені завдання, а саме:

- розроблено NoSQL базу даних на платформі MongoDB, розміщено її на віддаленому сервері для забезпечення постійного доступу та високого рівня безпеки;
- забезпечено контрольований доступ до API завдяки інструменту CORS, за допомогою якого в коді програми ми змогли обмежити доступ до ресурсу, та можливостями хостинг-провайдерів, які надали нам список своїх статичний ip-адрес, щоб дозволити доступ до розробленої системи лише їм;
- розмістивши наші окремі частини системи децентралізовано на різних ресурсах ми не тільки забезпечили безпеку, але й їх доступність з будь-яких мереж та пристроїв;
- успішно створена валідація даних, яка не дозволяє користувачам внести в базу неправильні дані або ж надто велику їх кількість, що могло б потенційно не тільки знищити структуру БД, а й призвести до приведення її

у повну непрацездатність. Нам вдалось це зробити завдяки інструментам бібліотеки MongoDB.Driver та стороні Back-end і бібліотеці yum на боці користувача;

- створено зручний користувацький інтерфейс, який не перенавантажений інформацією, швидко працює та має функцію зміни теми інтерфейсу. Цього результату ми досягли завдяки сучасному фреймворку для створення сайтів з користувацькими інтерфейсами – React JS;
- з допомогою бібліотеки Formik створено масштабовані, зручні в налаштуванні та керування форми заповнення даних. Окрім зручності для програміста вони є зручними і для користувача, тому що можуть відображати підказки, інформацію про помилки, повідомлення валідації і загалом працюють дуже швидко;
- створено інструмент, який дозволяє кожен елемент бази даних розмісити на карті в форматі SVG та переміщувати його, зберігаючи при цьому координати. Це дозволяє нам бачити в якому кабінеті знаходиться та чи інша техніка, що є дуже зручним при пошуку потрібних екземплярів.

На кожному з етапів розробки виникали свої проблеми, зокрема такі, як неправильне збереження координат елементів на мапі, втрата даних в ході збереження в базу, але завдяки тому, що використані нами інструменти є популярними та мають велику спільноту людей, готових допомогти та підказати, ці проблеми було вирішено.

Наступним етапом було тестування програмного продукту. Результати, які були отримані під час тестування є задовільними і підтверджують реалізацію розроблюваної нами системи.

Головним досягненням кваліфікаційної роботи є створення комплексної децентралізованої системи обліку продукції, яка дозволяє швидко та зручно взаємодіяти з інформацією і є при цьому захищеною.

Система розроблялась з таким підходом, що масштабування проєкту буде максимально простим, тому в подальшому можна додати наступний функціонал:

дозволити користувачам самостійно додавати види техніки та продукції, завантажувати власні мапи території в форматі SVG, надати змогу власникам системи створювати ролі користувачів та реєструвати інших користувачів (своїх працівників, наприклад). Якщо додатково реалізувати написаний вище функціонал, то розроблену систему можна буде позиціонувати не тільки як тимчасове або дуже просте рішення для дрібних підприємств, а й як щось більше.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Gottfried J. A complete guide to modern web applications. MEDIUM. URL: https://medium.com/jeremy-gottfrieds-tech-blog/a-complete-guide-to-modern-web-applications-793ae71b57ad (date of access: 27.04.2023).

2. Modern web tooling | visual studio – visual studio. Visual Studio. URL: https://visualstudio.microsoft.com/vs/features/web/ (date of access: 01.06.2023).

3. Nghiem T. How to set up VS code for web development in A few simple steps. freeCodeCamp.org. URL: https://www.freecodecamp.org/news/how-to-set-up-vs-code-for-web-development/ (date of access: 23.05.2023).

4. JavaScript ES6. W3Schools Online Web Tutorials. URL: https://www.w3schools.com/js/js_es6.asp (date of access: 023.05.2023).

5. What Is NoSQL? NoSQL Databases Explained. MongoDB. URL: https://www.mongodb.com/nosql-explained (date of access: 02.04.2023).

6. Explaining BSON with examples. MongoDB. URL: https://www.mongodb.com/basics/bson (date of access: 02.04.2023).

7. State: a component's memory – react. React. URL: https://react.dev/learn/state-a-components-memory (date of access: 11.04.2023).

8. Data validation. IBM documentation | IBM. URL: https://www.ibm.com/docs/en/cdfsp/7.6.0?topic=environment-data-validation (date of access: 12.04.2023).

9. Validation | formik. Formik: Build forms in React, without the tears. URL: https://formik.org/docs/guides/validation (date of access: 12.04.2023).

10. Jain A. React + redux architecture: separation of concerns. Medium. URL: https://medium.com/@abhinav_jain123/react-redux-architecture-part-1-separation-of-concerns-812da3b08b46 (date of access: 03.05.2023).

11. Pattankar M., Hurbuns M. Mastering ASP.NET web API. Packt Publishing Ltd, 2017. 330 p.

12. MongoDB fundamentals C#/.NET. MongoDB: The Developer Data Platform | MongoDB. URL: https://www.mongodb.com/docs/drivers/csharp/current/fundamentals/ (date of access: 02.04.2023).

13. Enable cross-origin requests (CORS) in ASP.NET core. Microsoft Learn: Build skills that open doors in your career. URL: https://learn.microsoft.com/en-us/aspnet/core/security/cors?view=aspnetcore-7.0 (date of access: 02.06.2023).

14. Validation Schema | formik. Formik: Build forms in React, without the tears. URL: https://formik.org/docs/guides/validation#validationschema (date of access: 17.05.2023).

15. Deploy a react app static site | Render. URL: https://render.com/docs/deploy-create-react-app#using-client-side-routing (date of access: 01.06.2023).

ДОДАТКИ

Додаток А

Лістинг А – контролер «ItemsController»

```
using Microsoft.AspNetCore.Mvc;
using StorageProject.Api.Models;
using StorageProject.Api.Services;

namespace StorageProject.Api.Controllers
{
   [Route("api/items/")]
   [ApiController]
   public class ItemsController : ControllerBase
   {
      private readonly ItemsService _itemsService;
      private readonly ItemTypesService _itemTypesService;
      private readonly ProvidersService _providersService;

      public ItemsController(ItemsService itemsService, ItemTypesService itemTypesService,
ProvidersService providersService)
      {
         _itemsService = itemsService;
         _itemTypesService = itemTypesService;
         _providersService = providersService;
      }

      private async Task<ItemModel> getHelper(ItemModel item)
      {
         var existingItemType = await _itemTypesService.GetAsyncById(item.ItemTypeId);
         var existingProvider = await _providersService.GetAsyncById(item.ProviderId);

         if (existingItemType is not null) { item.ItemType = existingItemType; }
         if (existingProvider is not null) { item.Provider = existingProvider; }

         return item;
      }


      [HttpGet("get/{id:length(24)}")]
      public async Task<IActionResult> Get(string id)
      {
         var existingItem = await _itemsService.GetAsync(id);
         if (existingItem is null) { return BadRequest(); }

         var result = await getHelper(existingItem);

         return Ok(result);
      }
```

```
[HttpGet("get/all")]
public async Task<IActionResult> Get()
{
   var allItems = await _itemsService.GetAsync();
   if (allItems.Any())
   {
      foreach (var item in allItems)
      {
         var existingItemType = await _itemTypesService.GetAsyncById(item.ItemTypeId);
         var existingProvider = await _providersService.GetAsyncById(item.ProviderId);

         if (existingItemType is not null) { item.ItemType = existingItemType; }
         if (existingProvider is not null) { item.Provider = existingProvider; }
      }
      return Ok(allItems);
   }
   return NotFound();
}


[HttpPost("new-item")]
public async Task<IActionResult> Create(ItemModel item)
{
   var allItems = await _itemsService.GetAsync();

   if (allItems.Any(x => x.SerialNumber == item.SerialNumber) &&
      allItems.Any(x => x.Name.ToLower() == item.Name.ToLower()))
   {
      return BadRequest("Item with whis name and serial number already exists");
   }

   var qrCodeCoreValue = $"{item.Name}/{item.SerialNumber}";
   item.QrCode!.CoreValue = qrCodeCoreValue;

   await _itemsService.CreateAsync(item);

   return Ok(await getHelper(item));

}


[HttpPut("update/{id:length(24)}")]
public async Task<IActionResult> Update(string id, ItemModel item)
{
   var existingItem = await _itemsService.GetAsync(id);

   if (existingItem is null)
```

```
            return BadRequest();

        item.Id = existingItem.Id;

        await _itemsService.UpdateAsync(item);

        return Ok(item);
    }


    [HttpDelete("delete/{id:length(24)}")]
    public async Task<IActionResult> Delete(string id)
    {
        var existingItem = await _itemsService.GetAsync(id);

        if (existingItem is null) { return BadRequest(); }

        await _itemsService.RemoveAsync(id);

        return Ok(existingItem);
    }

  }
}
```

кінець лістингу А