МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА СЕРВІСУ АВТОМАТИЗОВАНОГО ЗБОРУ ТА ОБРОБКИ ДАНИХ»**

Виконав: студент 4 курсу, групи 6.1219-2пі
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
О.О. Безбородько
(ініціали та прізвище)
Керівник доцент кафедри програмної інженерії, доцент, к.ф.-м.н. Мильцев О.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент завідувач кафедри фундаментальної та прикладної математики, професор, д.т.н. Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ 07 ” 02 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Безбородьку Олексію Олександровичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка сервісу автоматизованого збору та обробки даних

керівник роботи Мильцев Олександр Михайлович, к.ф.-м.н., доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 26 » 01 2023 року № 102-с

2. Строк подання студентом роботи 07.06.2023 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Розробка сервісу автоматизованого збору та обробки даних.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою докладу

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 07.02.2023 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	08.02.2023	
2.	Збір вихідних даних.	17.02.2023	
3.	Обробка методичних та теоретичних	10.03.2023	
	джерел.		
4.	Розробка першого та другого розділу.	14.04.2023	
5.	Розробка третього розділу.	15.05.2023	
6.	Оформлення та нормоконтроль	01.06.2023	
	кваліфікаційної роботи.		
7.	Захист кваліфікаційної роботи.	21.06.2023	

Студент _____ О.О. Безбородько
(підпис) (ініціали та прізвище)

Керівник роботи _____ О.М. Мильцев
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка сервісу автоматизованого збору та обробки даних»: 55 с., 11 рис., 12 джерел.

БД, ASP .NET, API, C#, ENTITY FRAMEWORK, FRAMEWORK, MVC, .NET, REACT NATIVE, UML.

Об’єкт дослідження – система, інструменти для взаємодії ASP .NET Core та React Native, засоби взаємодії системи з користувачем.

Мета роботи – розробити систему синхронізації цін.

Методи дослідження – моделювання, проектування, програмний, аналітичний.

У наш час сфери торгівлі пропонують клієнту різноманітні товари та послуги, на які продавець має встановити власну ціну. Для спрощення цієї процедури з боку продавця та автоматичної калькуляції цін згідно з матрицею продажів, знижок і так далі виставляв би ціни автоматично усюди такі які вони потрібні бути.

Система розробляється з метою автоматизувати зміну ціни та наявності товарів на ресурсах продажу, таких як магазини та маркетплейси. Вона враховує закупівельну ціну, відсоток ресурсу продажу та відсоток прибутку з товарної позиції, щоб визначити необхідну ціну товару та передати її у систему. Крім того, в залежності від залишків товарів на складах, система передає ці дані у систему джерел продажу, такі як маркетплейс або магазин на OpenCart.

Основний алгоритм роботи системи наступник: система збирає дані з YML або JSON файлу у систему, збирає дані з ресурсів продажу (Rozetka, Prom та інші) за допомогою API у систему, збирає дані про залишки товарів на складах з JSON або YML-файлу та автоматично оновлює наявність та ціни в кожному з джерел продажу, в залежності від раніше описаних параметрів.

Таким чином, за результатами роботи створено зручну та ефективну систему синхронізації цін на ASP .NET Core WebAPI.

SUMMARY

Bachelor’s qualifying paper «Development of an Automated Data Collection and Processing Service»: 55 p., 11 figures, 12 references.

ASP .NET, API, C#, ENTITY FRAMEWORK, FRAMEWORK, MVC, .NET, REACT NATIVE, UML, DB.

The object of the study is a system, tools for the interaction of ASP .NET Core and React Native, the creation of a system interaction with a core.

The aim of the study is to implement the price synchronization system.

The method of research is modeling, design, programming, analytical.

Nowadays, trade areas offer the client a variety of goods and services, for which the seller must set his own price. To simplify this procedure on the part of the seller and automatically calculate prices according to the matrix of sales, discounts, and so on, I would set the prices automatically everywhere as they should be.

The system is being developed to automate price and availability changes on sales resources such as stores and marketplaces. It takes into account the purchase price, the percentage of the sales resource and the profit percentage of the item to determine the required price of the item and transfer it to the system. In addition, depending on the balance of goods in warehouses, the system transmits this data to the system of sales sources, such as a marketplace or an OpenCart store.

The main algorithm of the system is as follows: the system collects data from a YML or JSON file into the system, collects data from sales resources (Rozetka, Prom and others) using an API into the system, collects data about the remaining goods in warehouses from a JSON or YML file and automatically updates the availability and prices in each of the sales sources, depending on the previously described parameters.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

DOM	Document Object Model
API	Application Programming Interface
YML	Тип файлу YML (YAML Application Markup Language)
JSON	JavaScript Object Notation
HTTP	HyperText Transfer Protocol
JS	Javascript
HTML	HyperText Markup Language
CSS	Cascade StyleSheets
ORM	Object Relational Mapping

ВСТУП

Система розробляється з метою автоматизації зміни ціни та наявності товарів на ресурсах продажу, таких як магазини та маркетплейси. Замовнику було необхідно мати можливість без проблем синхронізувати усі дані товарів поміж усіх платформ, на яких він їх продає. Система має визначати необхідну ціну товару залежно від закупівельної ціни, відсотка ресурсу продажу та відсотка прибутку з товарної позиції, а потім передавати її в систему. Також вона повинна передавати дані про залишки товарів на складах в систему джерел продажу (маркетплейс або магазин на OpenCart).

Загальний алгоритм роботи системи наступний:

– парсинг даних (список товарів) з YML-файлу в систему;
– парсинг даних з ресурсів Rozetka і Prom – за допомогою API в систему;
– парсинг даних з інших джерел (якщо такі є) в систему за допомогою YML-файлу;
– парсинг даних про залишки товарів на складах з JSON або YML-файлу в систему.

Автоматичне оновлення наявності та ціни в кожному з джерел продажу в залежності від раніше описаних параметрів.

Актуальність дослідження: актуальність теми зумовлена необхідністю та прагненню замовника автоматизувати синхронізацію цін поміж товарів на різних платформах.

З огляду на це, можна виділити наступні цілі і задачі нашого дослідження:

Мета: розробити систему синхронізації цін поміж товарів.

Задачі:

– сформувати вимоги до системи;
– спроєктувати та побудувати архітектуру системи;
– реалізувати систему синхронизації цін;
– протестувати роботу системи.

1 ТЕХНІЧНЕ ЗАВДАННЯ

1.1 Терміни та визначення

Бекенд – елемент системи, є частиною веб-додатка, яка відповідає за обробку логіки, збереження даних та забезпечення комунікації з фронтендом.

Фронтенд – та частина веб-додатка, яка відповідає за відображення інтерфейсу користувача та взаємодію з користувачем

YML-файл – формат файлу, який використовується для опису товарів і їх характеристик у структурованому вигляді.

API – інтерфейс програмування додатків – набір готових класів, процедур, функцій або структур, які використовуються для взаємодії між програмами.

Ресурс продажу – магазин або маркетплейс, де продаються товари.

Кросплатформовість – властивість програмного забезпечення або додатків, яка дозволяє їм працювати на різних платформах або операційних системах з мінімальними змінами в коді або логіці. Це означає, що можна розробити одну версію додатку і використовувати її на різних платформах, таких як iOS, Android, веб-браузери та інші, замість написання окремого коду для кожної платформи.

Нативний – це термін, що використовується в програмному забезпеченні для опису компонентів або додатків, розроблених для конкретної платформи або операційної системи. Коли говорять про нативний код, це означає, що код написаний з використанням мови програмування та інструментів, які є стандартними для даної платформи.

Фреймворк – це набір компонентів, бібліотек, інструментів та правил, що надають основу для розробки програмного забезпечення або додатків. Він визначає структуру, організацію та спосіб взаємодії між компонентами програми, сприяючи швидкій розробці та підтримці програмного забезпечення.

Перевикористання – це принцип або практика в програмному забезпеченні, коли певні компоненти, модулі, функції або інші елементи коду можуть бути використані повторно в різних частинах програми або навіть в різних програмах.

Маркетплейс – це онлайн-платформа, де продавці та покупці можуть здійснювати торгівлю товарами або послугами.

Закупочна ціна – вартість товару для продавця або постачальника, включаючи витрати на придбання, доставку та інші витрати.

Процент ресурсу продажі – відсоток вартості товару, який додається до закупочної ціни для визначення ціни продажу на ресурсі продажу.

Процент прибутку – відсоток прибутку, який додається до вартості товару для визначення ціни продажу.

JSON – легкий формат обміну даними, який легко читається людьми і легко аналізується машинами.

React Native – фреймворк для розробки мобільних додатків.

1.2 Функціональні вимоги

Парсинг даних з YML-файлу: система повинна здати обробити та зчитати інформацію про товари з YML-файлу.

Парсинг даних з історників продажу по API: система повинна використовувати API історників продажу (наприклад, Rozetka і Prom) для отримання актуальних даних про ціни та наявність товарів.

Парсинг даних з інших джерел: якщо наявні, система повинна здати обробити та зчитати інформацію про товари з інших джерел, які використовують YML-файл для опису товарів.

Парсинг даних про складські залишки: система повинна зчитати дані про наявність товарів на складах з JSON або YML-файлів.

Автоматичне оновлення цін і наявності: система повинна автоматично оновлювати ціни та наявність товарів на ресурсах продажу згідно з встановленими параметрами, такими як закупочна ціна, процент ресурсу продажу та процент прибутку.

1.3 Нефункціональні вимоги

Мова програмування: розробка системи повинна здійснюватися з використанням мови програмування C#.

Розробка на платформі .NET: система повинна бути розроблена з використанням платформи .NET.

Використання Visual Studio: розробка повинна проводитися в середовищі Visual Studio.

Використання ASP .NET Core: для розробки веб-застосунку слід використовувати фреймворк ASP .NET Core.

Збереження даних: система повинна використовувати базу даних для збереження інформації про товари, ціни та наявність.

1.4 Опис предметної області

Дана система розробляється з метою автоматизації процесу зміни цін та наявності товарів на різних ресурсах продажу, таких як магазини та маркетплейси. Вона використовує дані з різних джерел, включаючи YML-файли, API ресурсів продажу та JSON-або YML-файли з даними про складські залишки. Система враховує закупочну ціну, процент ресурсу продажу та процент прибутку для визначення оптимальної ціни продажу.

1.5 Опис системи

Система розробляється для автоматизації процесу зміни цін та наявності товарів на різних ресурсах продажу. Вона має наступні функціональні можливості: парсинг даних з YML-файлів, використання API ресурсів продажу для отримання актуальних даних, парсинг даних з інших джерел, парсинг даних

про складські залишки та автоматичне оновлення цін та наявності товарів на ресурсах продажу згідно з заданими параметрами.

Серверна частина системи розроблена з використанням .NET, ASP .NET Core та MSSQL DB і відповідає за обробку та керування даними. Вона складається з низки сервісів, які забезпечують основні функції системи, такі як парсинг даних з різних джерел, визначення оптимальної ціни товарів, керування наявністю товарів та передачу даних до систем джерел продажу.

Також, серверна частина включає REST API, яке забезпечує взаємодію між сервером та клієнтською частиною додатка. API надає набір ендпоінтів, які дозволяють клієнтському додатку виконувати різноманітні операції, такі як отримання, оновлення та видалення даних.

Клієнтська частина додатка розроблена з використанням React Native та React Native Web і представляє собою інтерфейс, через який користувачі взаємодіють з системою. Вона містить набір екранів та компонентів, які надають доступ до функціональності системи, такі як перегляд та редагування інформації про товари, налаштування автоматичних правил для зміни ціни та наявності товарів, а також відстеження статусу певних операцій.

1.6 Огляд інструментів розробки

Для розробки системи використовуються наступні інструменти та технології.

Мова програмування: C# – мова програмування, що надає широкі можливості для розробки веб- та десктоп-додатків на платформі .NET.

React Native – це відкритий фреймворк для розробки мобільних додатків, який дозволяє створювати перехідні мобільні програми для iOS та Android, використовуючи JavaScript та платформу React. Він є варіантом фреймворка React, спеціально адаптованим для мобільних платформ. React Native базується на архітектурі «однієї кодової бази, багатьох платформ». Це означає, що весь

основний код додатка пишеться на JavaScript, а компоненти і API React Native перетворюють його на нативний код для кожної платформи. Для iOS використовується Objective-C або Swift, а для Android – Java або Kotlin. Це дозволяє розробникам використовувати спільну кодову базу для обох платформ і зменшує зусилля, потрібні для розробки й підтримки мобільних додатків.

React Native надає розробникам доступ до широкого набору можливостей для побудови мобільних додатків.

TypeScript – це надбудова над JavaScript, розроблена Microsoft, яка додає статичну типізацію та додаткові об’єктно-орієнтовані можливості до мови. TypeScript спрощує написання більш надійного та безпечного коду, полегшує його налагодження та підтримку.

Платформа: .NET – платформа розробки програмного забезпечення, яка надає ряд інструментів та бібліотек для створення програм для Windows, Linux та інших операційних систем.

Розробка веб-застосунків: ASP .NET Core – фреймворк для розробки веб-додатків на мові C#, який надає високу продуктивність, переносимість та масштабованість.

Розробка середовища: Visual Studio – інтегроване середовище розробки (IDE) для платформи .NET, яке надає зручні інструменти для створення, налагодження та відлагодження програмного забезпечення.

Збереження даних: для збереження інформації про товари, ціни та наявність використовується база даних, яка підтримується платформою .NET.

Для забезпечення взаємодії між клієнтською (візуальною) частиною додатка та сервером має використовуватися REST API. REST API дозволяє клієнтському додатку відправляти запити на сервер та отримувати від нього відповіді, що включають необхідні дані або статус операції. Це дає можливість реалізувати високий рівень абстракції, забезпечуючи гнучкість та масштабованість системи.

1.6.1 Структура front-end проєкту

Структура front-end проєкту складається з декількох основних папок:

- **android/** і **ios/**: директорії, які містять код для платформ Android та iOS відповідно;
- **index.js** або **index.js/App.js**: головний файл, в якому відбувається ініціалізація додатка та рендеринг головного компонента;
- **src/**: директорія, де зберігаються власні компоненти, стилі, роутери та інші файли додатка;
- **node_modules/**: директорія, в якій зберігаються залежності проєкту, установлені за допомогою npm або yarn;
- **package.json**: файл, що містить інформацію про ваш проєкт, включаючи залежності та налаштування скриптів.

1.6.2 Структура back-end проєкту

API. Проєкт API відповідає за надання точок доступу та обробку вхідних запитів від клієнтів. Він містить контролери, які визначають маршрути та дії для обробки різних HTTP-запитів. Проєкт API взаємодіє з проєктами BLL та DAL для обробки запитів, отримання або оновлення даних та повернення відповідних відповідей клієнтам.

BLL. Проєкт BLL містить бізнес-логіку додатку. Він укриває правила та операції, що регулюють поведінку додатку. Цей рівень відповідає за реалізацію бізнес-логіки, виконання перевірок, виконання складних операцій та координацію взаємодії між проєктами API та DAL. Він включає служби, менеджери або інші компоненти, які обробляють бізнес-операції.

CORE. Проєкт CORE є загальним проєктом, який містить загальний код та утиліти, які використовуються в усьому рішенні. Він служить центральним місцем для визначення основних функціональностей, загальних компонентів та засобів, які використовуються в проєктах API, BLL та DAL.

DAL. Проєкт DAL відповідає за доступ до даних та взаємодію з джерелами даних, такими як база даних або зовнішні сервіси. Він містить класи та компоненти, які забезпечують операції збереження, отримання, оновлення та видалення даних. DAL виконує завдання доступу до даних, а BLL використовує ці операції для реалізації бізнес-логіки.

SharedModels. Проєкт SharedModels містить моделі даних або класи, які використовуються як загальний код між різними проєктами в рішенні. Цей проєкт дозволяє спільно використовувати моделі даних та інші класи, що є необхідними для взаємодії між проєктами API, BLL та DAL.

2 ПРОЄКТУВАННЯ

2.1 Використання UML під час розробки системи

Для розробки системи використовуються різні методології. Одним з найпоширеніших мов моделювання є уніфікована мова моделювання UML (Unified Modeling Language), призначена для моделювання, уявлення, проєктування та документування програмних систем, організаційно-економічних систем, технічних систем та інших систем різної природи. UML дозволяє розробникам програмного забезпечення досягти згоди щодо графічних позначень для представлення загальних понять, таких як класи, компоненти, узагальнення, об’єднання і поведінка, і зосередитися на проєктуванні та архітектурі.

UML використовується для моделювання бізнес-процесів, системного проєктування та відображення організаційних структур. Вона не прив’язана до певної платформи або мови програмування, що дозволяє з’єднати мережі різних систем. UML містить стандартний набір діаграм і нотацій різних видів, які допомагають організувати спілкування між замовниками і розробниками.

Нижче наведено список діаграм, які містить UML.

Діаграми структури.

Діаграма класів (Class Diagrams) – моделює статичну структуру класів системи і зв’язків між ними.

Діаграми компонентів (Component Diagrams) – моделюють ієрархію компонентів або підсистем системи.

Діаграми розміщення (Deployment Diagrams) – моделюють фізичну архітектуру системи.

Моделі поведінки.

Діаграми варіантів використання (Use Case Diagrams) – моделюють функціональні вимоги до системи у вигляді сценаріїв взаємодії користувачів з системою.

2.2 Архітектура

На даному проєкті була реалізована багатошарова архітектура проєктування системи.

Багатошарова архітектура є популярним підходом при проєктуванні ASP.NET Core проєктів, оскільки вона сприяє модульності, розширюваності та підтримці принципів SOLID, що дозволяє краще управляти складністю проєкту та покращує його обслуговування у майбутньому.. У такій архітектурі можна виділити кілька ключових шарів, таких як шар бізнес-логіки, шар доступу до даних і шар презентації.

Перший шар, шар бізнес-логіки – Business Logic Layer (BLL), відповідає за обробку бізнес-правил та логіки додатку. Він містить класи, які визначають моделі даних, сервіси та менеджери, що здійснюють операції з цими моделями. Шар бізнес-логіки виконує основні операції над даними, реалізує бізнес-правила та логіку додатку.

Другий шар, шар доступу до даних – Data Access Layer (DAL), відповідає за роботу з базою даних або іншими джерелами даних. В ньому реалізовано патерн репозиторій, де класи репозиторіїв забезпечують методи для звернення до даних, додавання, видалення та оновлення записів дозволяючи розділити логіку доступу до даних від бізнес-логіки. Це полегшує тестування та зміну джерела даних без зміни бізнес-логіки. Шар доступу до даних може включати ORM (Object-Relational Mapping) для спрощення роботи з базою даних. ORM (Object-Relational Mapping) – це технологія, яка дозволяє зв’язувати об’єктно-орієнтовану модель програми з реляційною базою даних. Вона надає зручний спосіб взаємодії з базою даних, де об’єкти програми відображаються в таблиці бази даних, а доступ до даних здійснюється за допомогою об’єктів та їх властивостей. ORM дозволяє розробникам працювати з даними в більш об’єктно-орієнтованому стилі, не звертаючи прямої уваги до складностей реляційних баз даних. Замість написання складних SQL-запитів, розробник може використовувати методи та властивості об’єктів для здійснення операцій з базою даних, таких як вставка, оновлення, видалення та отримання даних. На проєкті

було використано технологію Entity Framework від Microsoft, яка реалізує в собі концепцію ORM.

Третій шар, шар презентації або контролери, відповідає за обробку запитів від клієнта і відображення відповідей. Контролери приймають HTTP-запити, взаємодіють з шаром бізнес-логіки та шаром доступу до даних і повертають відповіді у вигляді HTTP-відповідей або представлень.

Крім основних шарів, багатошарова архітектура може включати інші шари, такі як шар інфраструктури або шар безпеки, в залежності від вимог проєкту. Шар інфраструктури забезпечує зовнішні залежності, такі як підключення до бази даних або зовнішні API. Шар безпеки відповідає за аутентифікацію, авторизацію та захист додатку.

Така багатошарова архітектура дозволяє зберігати логіку додатку відокремлено від інфраструктури та інших компонентів. Вона спрощує тестування, підтримку та розширення проєкту, оскільки кожен шар виконує конкретні функції та має чіткі відповідальності. Крім того, вона дозволяє використовувати принцип інверсії залежностей (Dependency Inversion Principle), де високорівневі компоненти не залежать від низькорівневих компонентів, або залежать від абстракцій, а не від конкретних реалізацій.

Загалом, багатошарова архітектура з патерном репозиторію є потужним підходом для проєктування ASP.NET Core проєктів, оскільки вона допомагає створити добре організовану, розширювану та тестовану систему з чітко визначеними відповідальностями для кожного шару. При правильному проєктуванні та реалізації багатошарової архітектури ASP.NET Core проєкт стає більш масштабованим, підтримуваним та зручним у використанні.

2.3 Діаграма прецедентів

Для опису функціональної структури системи використовується діаграма варіантів використання. Цей тип діаграми описує загальну функціональність системи. Вона відображає взаємодію між варіантами використання, які

представляють функції системи, та діючими особами, які представляють людей або системи, що отримують або передають інформацію в дану систему. Кожна функціональність зображується у вигляді «прецедентів використання» (use case) або просто прецедентів. Прецедент – це типова взаємодія користувача з системою, яка може представляти різні рівні деталізації, описувати видиму користувачем функцію, забезпечувати досягнення конкретної мети, важливої для користувача. Прецедент зображується як овал, пов'язаний з типовими користувачами, які називаються акторами (actors). Список всіх прецедентів фактично визначає функціональні вимоги до системи.

Мета даної діаграми полягає в наступному: проєктовані бізнес-процеси представляються у формі так званих варіантів використання, з якими взаємодіють зовнішні сутності або актори. При цьому актором називається будь-який об'єкт, суб'єкт або система, що взаємодіє з модельованою компанією ззовні. Це може бути людина, технічний пристрій, програма або система, яка служить джерелом впливу на модельовані бізнес-процеси системи так, як визначить розробник. Варіант використання служить для опису сервісів і функцій, які компанія надає замовнику. Іншими словами, кожен варіант використання визначає набір дій, які здійснює компанія при взаємодії з замовником – актором. При цьому не наводиться конкретна реалізація взаємодії акторів з компанією та виконання варіантів використання.

Переваги моделі варіантів використання полягають у наступному:

– визначає користувачів і межі системи;
– визначає системний інтерфейс;
– зручна для спілкування користувачів з розробниками;
– використовується для написання тестів;
– служить основою для написання документації користувача.

Між користувачами і варіантами використання можуть існувати різні види зв'язків. Основні види зв'язків наступні та наведені нижче.

Розширення (extend): показує, що варіант використання розширює базову послідовність дій і включає власну послідовність. Відмінністю від включення є те, що розширена послідовність може виконуватись у залежності від певних

умов.

Включення (include): показує, що варіант використання включається в базову послідовність і завжди виконується.

На діаграмі прецедентів зображуються ці взаємодії між акторами та прецедентами.

На рисунку 2.1 представлена діаграма прецедентів проєкту.

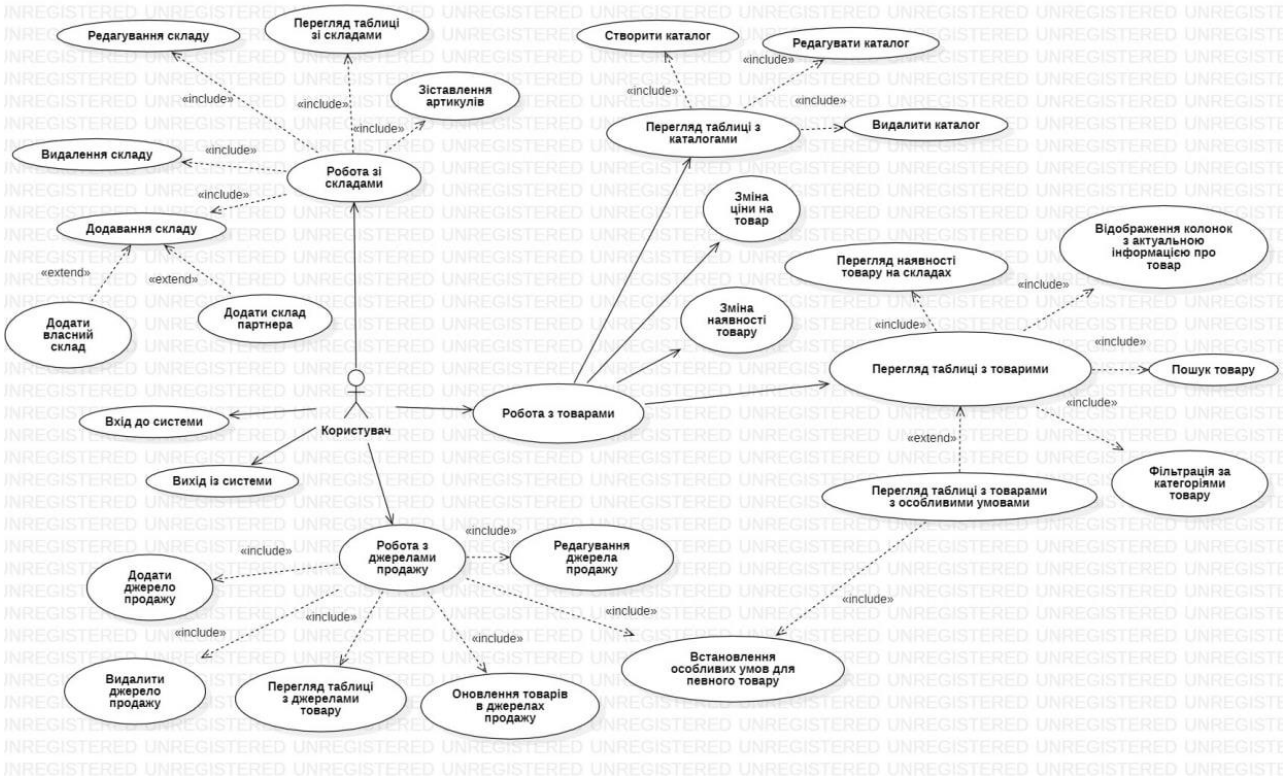

Рисунок 2.1 – Діаграма прецедентів

2.4 Діаграма класів

Діаграма класів є графічним зображенням структури класів та взаємозв’язків між ними в об’єктно-орієнтованому програмуванні. Вона допомагає візуалізувати основні компоненти системи, такі як класи, атрибути та методи, а також залежності та взаємодії між класами.

Діаграми класів забезпечують зрозумілу та структуровану інформацію про архітектуру програми, допомагають при аналізі, проєктуванні та розробці програмного забезпечення. Вони можуть використовуватись як комунікаційний

засіб між розробниками, дизайнерами та іншими учасниками проєкту.

За допомогою діаграм класів можна визначити ієрархію класів, спадкування, асоціації, агрегації та композиції між об’єктами. Вони також дозволяють відображати різні рівні видимості, модифікатори доступу та залежності між класами.

Загалом, діаграми класів є потужним інструментом для аналізу та проєктування об’єктно-орієнтованих систем, який сприяє зрозумілості, модульності та розширюваності програмного забезпечення.

На зображенні 2.2 представлено діаграму класів проєкту.

В системі є декілька основних класових моделей – класів з закінченням Base: ProductsBase, SalesSourceBase, SalesSourceProductBase, StorageBase, StorageProductBase. Кожний базовий клас успадковується відповідними моделями у контрактах передачі даних (DTO) та у шарі взаємодії з даними (DAL).

ProductsBase: цей клас є базовим класом для обробки продуктів базового каталогу. Він містить загальні властивості та методи, що стосуються управління та обробки продуктів в системі. Має в собі наступні поля: назва продукту, ціна, категорія, кількість, посилання.

SalesSourceBase: цей клас представляє базовий клас для джерел продажів. Він визначає загальні властивості та методи, необхідні для керування джерелами продажів в системі. Має в собі наступні поля: назва джерела продажу, адреса, список продуктів, список категорій, курс, тип експорту даних та додаткові поля з даними авторизації, де знаходяться різні дані в залежності від ресурсу продажів.

SalesSourceProductBase: цей клас є базовим класом для обробки продуктів, що пов’язані з джерелами продажу. Він розширює клас ProductsBase та містить додаткові властивості та методи, специфічні для продуктів, які продаються через конкретне джерело продажу має в собі наступні поля: доступна кількість продукту на складі цього джерела продажу, посилання на товар з базового каталогу, дата оновлення запасів та додаткова інформація в залежності від ресурсу продажів.

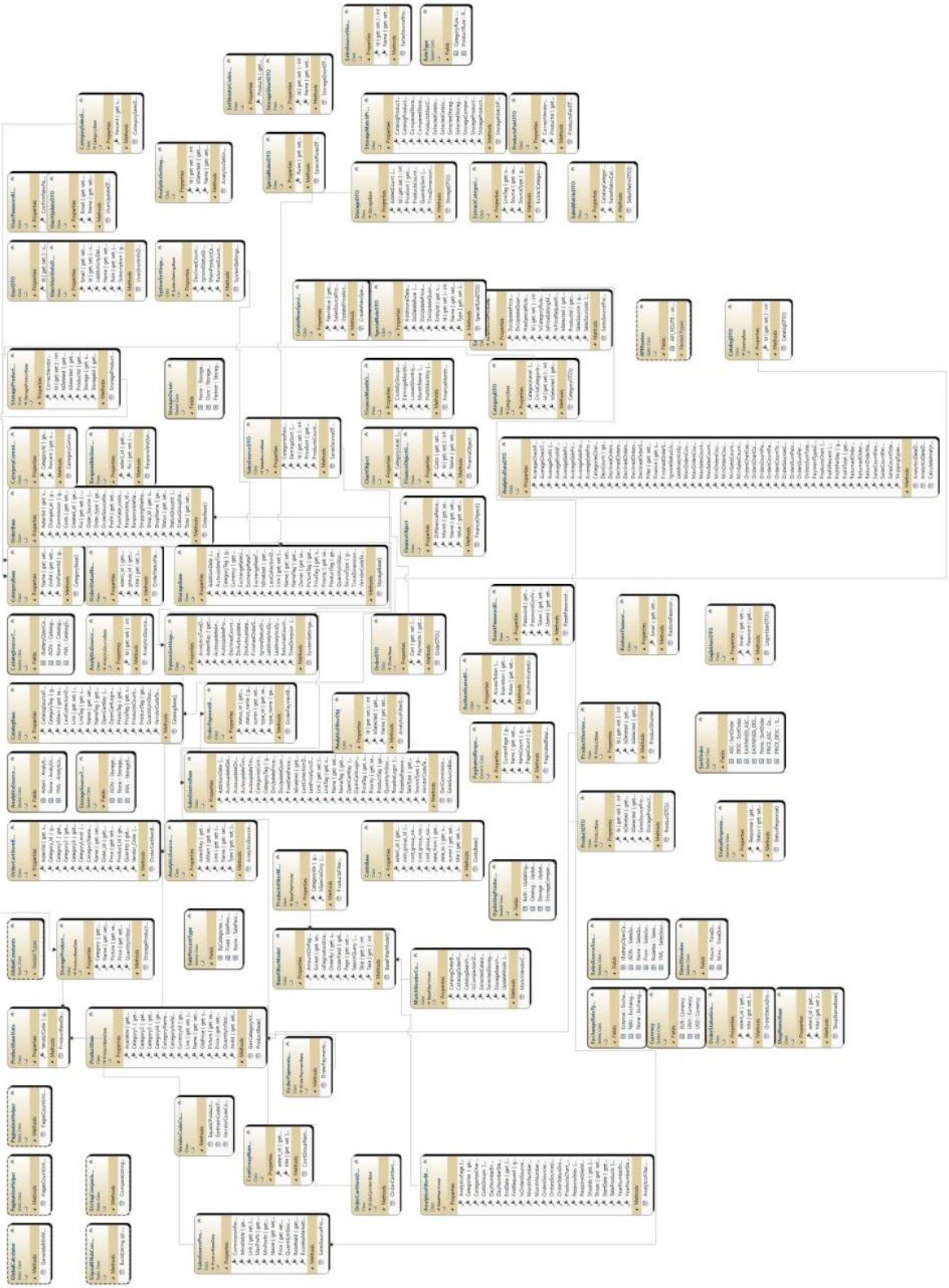

Рисунок 2.2 – Діаграма класів

StorageBase: цей клас є базовим класом для управління складами. Він містить загальні властивості та методи, необхідні для керування різними аспектами складського управління. Має в собі наступні поля: назва складу, адреса, відповідальна особа, список товарів, курс, валюта, поля для тегів, які дозволять реалізувати парсинг даних.

StorageProductBase: цей клас є базовим класом для обробки продуктів на складах. Він розширює клас ProductsBase та містить додаткові властивості та методи, специфічні для продуктів, що знаходяться на конкретному складі.

Також існують додаткові типи даних, які використовуються в різних частинах системи.

AnalyticsFilterModel: модель фільтра, яка використовується для зазначення параметрів фільтрації при отриманні аналітичних даних.

AnalyticsFilterObj: об’єкт, який містить значення фільтру для аналітичного запиту.

AnalyticsSettingsObj: об’єкт, що представляє налаштування для аналітики, такі як період збору даних, режим аналізу і т.д.

AnalyticsSourceDTO: DTO-об'єкт (Data Transfer Object), що містить інформацію про джерела аналітики, такі як назва, тип, URL тощо.

AnalyticsSourceType: перелік можливих типів джерел аналітики, таких як Google Analytics, Facebook Pixel тощо.

CatalogSourceType: перелік можливих типів джерел каталогу.

CategorySalesSourceDTO: DTO-об’єкт, який об’єднує інформацію про категорію товарів з джерелами продажу.

CreateNewSpecialRulesDTO: DTO-об’єкт, який використовується для створення нових спеціальних правил, наприклад, правил знижки або акцій.

Currency: об’єкт, який представляє валюту, включаючи код, назву, символ тощо.

ExchangeRateType: перелік можливих типів курсу обміну валют, наприклад, покупка, продаж, середній.

LoginUserDTO: DTO-об’єкт, який містить дані для авторизації

користувача, такі як ім’я користувача та пароль.

MatchVendorCodesFilterModel: модель фільтра, яка використовується для зіставлення кодів вендорів товарів.

ProductDTO: DTO-об’єкт, що містить інформацію про товар, таку як назва, опис, ціна тощо.

ProductsFilterModel: модель фільтра, що використовується для фільтрації списку продуктів за певними критеріями.

ProductsPairDTO: DTO-об’єкт, який містить пару продуктів для порівняння або аналізу.

RuleType: перелік можливих типів правил, які можуть бути застосовані до продуктів або категорій.

SalePercentType: перелік типів відсотків знижки, таких як фіксований, відсоток від ціни тощо.

SalesMatrixCategoryDTO: DTO-об’єкт, який містить інформацію про категорію в матриці продажів.

SalesSourceDTO: DTO-об’єкт, що містить інформацію про джерело продажу, таку як назва, URL, тип тощо.

SalesSourceProductDTO: DTO-об’єкт, який об’єднує інформацію про продукт з джерелом продажу.

SalesSourceShortDTO: короткий DTO-об'єкт, який містить основну інформацію про джерело продажу.

SalesSourceSourceType: перелік можливих типів джерел продажу, таких як онлайн-магазин, POS-система тощо.

SetVendorCodesDTO: DTO-об’єкт, який використовується для встановлення кодів вендорів для продуктів.

SortOrder: порядок сортування, такий як зростання або спадання.

SpecialRuleDTO: DTO-об’єкт, який містить інформацію про спеціальне правило, яке застосовується до продукту або категорії.

SpecialRulesDTO: DTO-об’єкт, що містить список спеціальних правил.

StorageDTO: DTO-об’єкт, що містить інформацію про склад, таку як назва, адреса, власник тощо.

StorageOwner: власник складу, який може бути користувачем системи.

StorageProductDTO: DTO-об’єкт, який об’єднує інформацію про продукт на складі.

StorageShortDTO: короткий DTO-об’єкт, який містить основну інформацію про склад.

StorageSourceType: перелік можливих типів джерел складу, таких як власний склад, постачальник тощо.

SystemSettingsDTO: DTO-об’єкт, що містить системні налаштування, такі як час оновленя базового каталогу, аналітики тощо.

2.5 Діаграма компонентів

Діаграма компонентів є важливим інструментом в архітектурі програмного забезпечення для візуалізації компонентної структури системи. Вона надає зрозуміле та зручне представлення взаємодії між компонентами системи, їх розміщення та взаємозв’язок.

Діаграма компонентів відображає фізичну або логічну структуру системи, включаючи компоненти, їх взаємозв’язки та взаємодію зовнішніх систем або користувачів. Вона часто використовується на етапі проєктування системи, де деталізується архітектурний дизайн та взаємодія компонентів.

У діаграмі компонентів компоненти представлені у вигляді прямокутників або куль, які відображають фізичні або логічні модулі системи. Кожен компонент має ім’я та може мати додаткові атрибути, такі як версія, автор або дата створення. Компоненти можуть бути вкладені один в одного для показує структури та організації системи.

Взаємозв’язки між компонентами відображаються за допомогою стрілок або ліній, які показують напрямок та тип взаємодії. Це можуть бути залежності між компонентами, такі як використання одного компонента іншим, або взаємодія за допомогою інтерфейсів, подій або повідомлень.

Діаграма компонентів також може включати інформацію про розміщення

компонентів на фізичних або логічних вузлах. Наприклад, компоненти можуть бути розподілені по різних серверах або включені в один процес. Це допомагає уявити фізичну структуру системи та розподілення навантаження.

Діаграма компонентів є корисним інструментом для комунікації між членами команди розробників, архітекторами та зацікавленими сторонами. Вона допомагає уточнити взаємозв'язки між компонентами, визначити границі відповідальності та підтримувати зрозумілість архітектури системи.

Крім того, діаграма компонентів може бути використана як основа для подальшого проєктування системи, визначення розподілення ресурсів та оптимізації взаємодії між компонентами.

Узагальнюючи, діаграма компонентів є потужним інструментом для візуалізації структури та взаємодії компонентів системи. Вона допомагає розробникам, архітекторам та зацікавленим сторонам краще розуміти систему, визначити залежності та забезпечити належну організацію програмного забезпечення.

На рисунку 2.3 представлено діаграму компонентів проєкту.

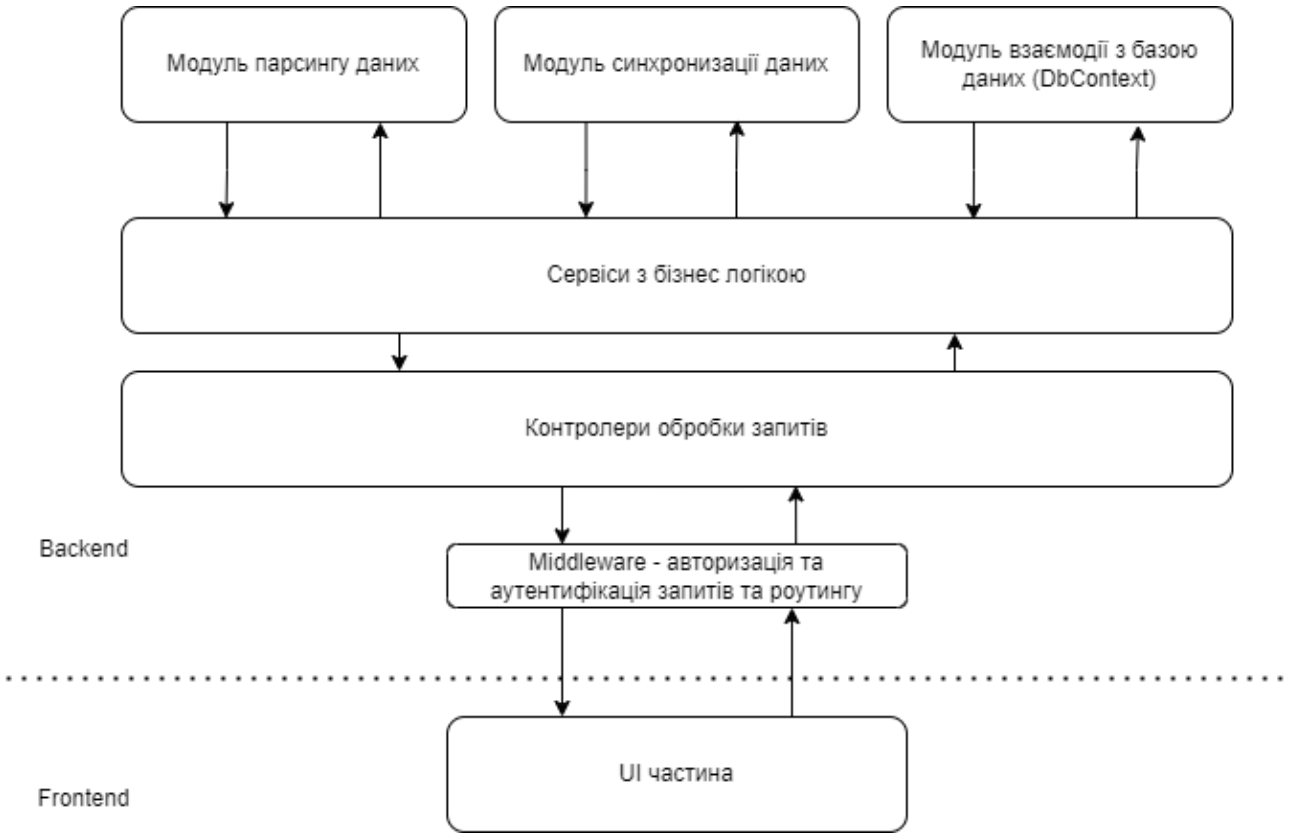

Рисунок 2.3 – Діаграма компонентів

Проєкт складається з двох основних частин – бекенду та фронтенду. Фронтенд відображає усю необхідну інформацію, яка обробляється та зберігається на бекенді.

Бекенд складається з контролерів обробки запитів. Кожен контролер стосується своєї логічної частини системи. Усього є 6 контролерів: User, Storage, SalesSource, Products, Authorization, Analytics.

User Controller. Функціонал:

- **GET /api/v1/user/info**: повертає інформацію про поточного користувача (ролі: Employee, Admin);
- **PUT /api/v1/user/update**: оновлює дані про поточного користувача (ролі: Employee, Admin).

Storage Controller. Функціонал:

- **POST /api/v1/storages/{isMainPage}**: отримує список складів (пагінація) (ролі: Employee, Admin);
- **PUT /api/v1/storages/{id}/update**: оновлює інформацію про конкретний склад (ролі: Employee, Admin);
- **DELETE /api/v1/storages/{id}/delete**: видаляє конкретний склад (ролі: Employee, Admin);
- **POST /api/v1/storages/{id}/getData**: завантажує дані зі складу (ролі: Employee, Admin);
- **POST /api/v1/storages/{id}/getMatchProducts**: отримує перелік продуктів для відповідності (пагінація) (ролі: Employee, Admin);
- **POST /api/v1/storages/{id}/setVendorCodes**: встановлює коди постачальника (ролі: Employee, Admin).

SalesSource Controller. Функціонал:

- **POST /api/v1/salessources**: отримує список джерел продажів (ролі: Employee, Admin);
- **PUT /api/v1/salessources/{id}/update**: оновлює інформацію про конкретне джерело продажу (ролі: Employee, Admin);
- **DELETE /api/v1/salessources/{id}/delete**: видаляє конкретне джерело

продажу (ролі: Employee, Admin);

- **POST /api/v1/salessources/{id}/getData**: завантажує дані з джерела продажу (ролі: Employee, Admin);
- **POST /api/v1/salessources/{id}/syncprices**: синхронізує ціни на джерелі продажу (ролі: Employee, Admin);
- **POST /api/v1/salessources/specialrules/{id}**: отримує спеціальні правила (ролі: Employee, Admin);
- **POST /api/v1/salessources/specialrules/{id}/set**: встановлює спеціальні правила (ролі: Employee, Admin);
- **POST /api/v1/salessources/specialrules/createNew**: створює нові спеціальні правила (ролі: Employee, Admin);
- **POST /api/v1/salessources/salesmatrix/{isEarningPercentage}**: отримує матрицю продажів (ролі: Employee, Admin);
- **POST /api/v1/salessources/salesmatrix/set/{isEarningPercentage}**: встановлює матрицю продажів (ролі: Employee, Admin).

Products Controller. Функціонал:

- **POST /api/v1/products**: отримує список всіх продуктів (пагінація) (ролі: Employee, Admin);
- **POST /api/v1/products/categories**: отримує список всіх категорій (ролі: Employee, Admin);
- **GET /api/v1/products/{id}**: отримує інформацію про конкретний продукт (ролі: Employee, Admin);
- **PUT /api/v1/products/{id}/update**: оновлює інформацію про конкретний продукт (ролі: Employee, Admin);
- **DELETE /api/v1/products/{id}/delete**: видаляє конкретний продукт (ролі: Employee, Admin);
- **GET /api/v1/products/settings**: отримує налаштування продуктів (ролі: Employee, Admin);
- **PUT /api/v1/products/settings/update**: оновлює налаштування продуктів (ролі: Employee, Admin);

– **GET /api/v1/products/catalogs**: отримує каталоги продуктів (ролі: Employee, Admin);
– **POST /api/v1/products/catalog/update**: оновлює каталоги продуктів (ролі: Employee, Admin);
– **DELETE /api/v1/products/catalog/{id}/delete**: видаляє каталог продуктів (ролі: Employee, Admin);
– **DELETE /api/v1/products/clearProducts**: очищає дані про продукти (ролі: Employee, Admin);
– **POST /api/v1/products/getDataFromMainCatalog**: отримує дані з основного каталогу (ролі: Employee, Admin);
– **GET /api/v1/products/getLaunchedDataExtractings**: отримує запущені процеси екстракції даних (ролі: Employee, Admin).

Analytics Controller. Функціонал:

– **POST /api/v1/analytics/analytics**: отримує дані аналітики (ролі: Employee, Admin);
– **POST /api/v1/analytics/getDataFromMainCatalog**: отримує дані з основного каталогу (ролі: Employee, Admin);
– **GET /api/v1/analytics/catalogs**: отримує каталоги аналітики (ролі: Employee, Admin);
– **POST /api/v1/analytics/catalog/update**: оновлює каталоги аналітики (ролі: Employee, Admin);
– **DELETE /api/v1/analytics/catalog/{id}/delete**: видаляє каталог аналітики (ролі: Employee, Admin);
– **DELETE /api/v1/analytics/clearAnalytics**: очищає дані аналітики (ролі: Employee, Admin).

Authorization Controller. Функціонал:

– **POST /api/v1/auth/login**: ендпоінт для входу в систему (повертає ролі, призначені користувачеві).

В залежності від контролера та роуту – викликається різний сервіс бізнес логіки які вже, в свою чергу, в залежності від бізнес задачі можуть

використовувати додаткові модулі системи – модуль парсингу, модуль синхронізації чи модуль взаємодії з базою даних (DbContext).

2.6 Діаграма розміщення

Діаграма розміщення є важливим інструментом в архітектурі програмного забезпечення, який дозволяє візуалізувати фізичну або логічну розстановку компонентів системи та їх взаємодію з фізичними або віртуальними ресурсами. Ця діаграма допомагає розробникам та архітекторам краще розуміти, як компоненти системи розміщені на різних фізичних або логічних вузлах і як вони спілкуються між собою та з навколишнім середовищем.

У діаграмі розміщення компоненти системи відображаються у вигляді прямокутників або куль, а фізичні або віртуальні вузли представлені у вигляді прямокутників або овалів. Кожен компонент та вузол має ім’я та може мати додаткові атрибути, такі як версія, IP-адреса, серійний номер тощо.

Взаємодія між компонентами та вузлами відображається за допомогою стрілок або ліній, які показують напрямок комунікації. Це можуть бути з’єднання між компонентами на одному вузлі, мережеві з’єднання між вузлами, залежності між компонентами та інші типи взаємодії.

Діаграма розміщення також може включати додаткову інформацію про фізичні або логічні характеристики вузлів, такі як обладнання, операційна система, середовище виконання та доступність ресурсів. Це допомагає визначити, як компоненти системи будуть розподілені та працювати в реальному середовищі.

Діаграма розміщення є важливим інструментом для планування, аналізу та оптимізації фізичної архітектури системи. Вона допомагає зрозуміти, як компоненти взаємодіють з фізичними або віртуальними ресурсами, як забезпечувати високу доступність та масштабованість, а також як управляти використанням ресурсів.

Діаграма розміщення може бути корисною як на етапі проєктування системи, так і під час її подальшого розвитку і супроводу. Вона допомагає виявити потенційні проблеми в архітектурі, визначити необхідність масштабування або оптимізації ресурсів та забезпечити належну роботу системи в реальному середовищі.

На рисунку 2.4 представлено діаграму розміщення проєкту.

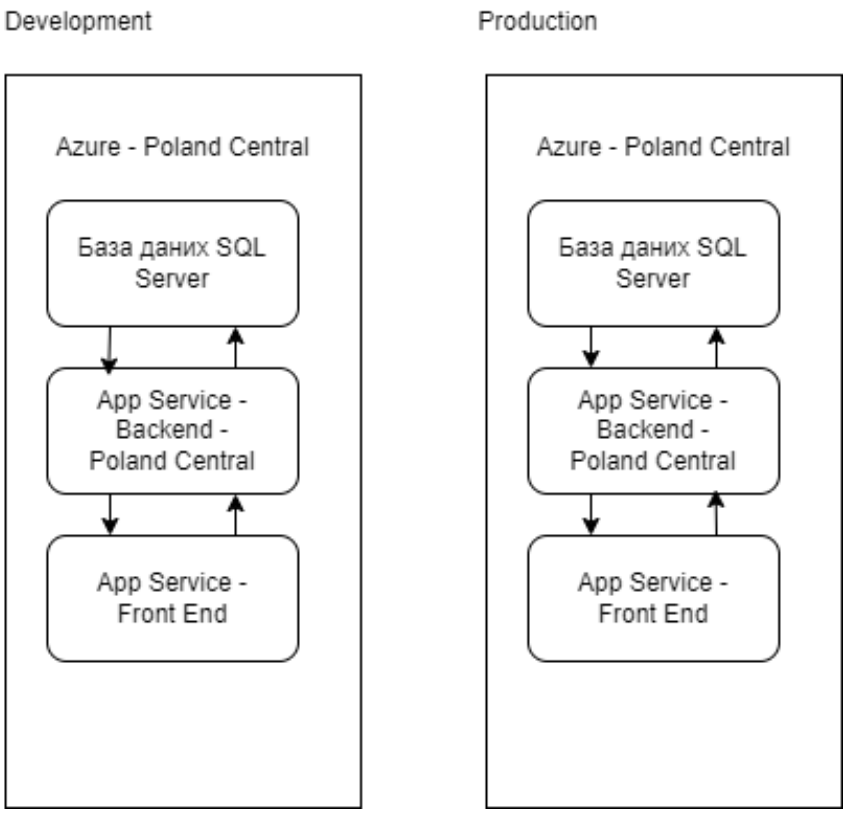

Рисунок 2.4 – Діаграма розміщення

В нашому випадку, наша система розміщена на серверній архітектурі Azure, що знаходиться в Польщі. Azure є хмарною платформою, яка надає надійне та масштабоване середовище для розгортання та виконання нашого додатку. Вибір Azure для розміщення нашої системи має декілька переваг. Перш за все, Azure забезпечує високу доступність і надійність завдяки своїй глобальній інфраструктурі та резервному копіюванню даних. Це означає, що наш додаток буде доступним і функціональним для користувачів у будь-який час. Крім того, Azure надає різноманітні сервіси, які полегшують розгортання, масштабування та керування нашою системою. Наприклад, ми можемо використовувати служби

Azure для керування базами даних, зберігання файлів, моніторингу та логування, автоматичного масштабування ресурсів та багато іншого. Розміщення нашої системи в Польщі важливо з точки зору швидкості взаємодії з системою. Основна локація взаємодії – Україна, найближчий сервер – Польща. Усі ці фактори роблять Azure відмінним вибором для розміщення нашої системи, забезпечуючи нам надійність, масштабованість та забезпечення дотримання вимог щодо захисту даних.

2.7 Серверні статуси

Серверні статуси, також відомі як статуси відповіді сервера, є кодами, які повертаються сервером у відповідь на HTTP-запити. Ці статуси передають результат обробки запиту сервером і надають клієнту інформацію про стан запиту, чи було все успішно виконано, чи сталася помилка, які дії треба вжити далі тощо. Серверні статуси допомагають узгоджувати комунікацію між клієнтом і сервером, а також визначати наступні кроки для обробки запиту.

Основні групи серверних статусів визначаються першим числом у коді статусу. Десятки вказують на клас статусу, а останні дві цифри вказують на конкретний статус. Розглянемо, що включають основні класи статусів.

1xx – Інформаційні статуси: ці статуси вказують, що сервер отримав запит, і процес обробки триває. Наприклад, 100 Continue показує, що сервер готовий прийняти наступну частину запиту.

2xx – Статуси успішного виконання: ці статуси показують, що запит клієнта був успішно оброблений сервером. Наприклад, 200 OK вказує на успішну обробку запиту і передачу результату клієнту.

3xx – Статуси перенаправлення: ці статуси вказують на необхідність перенаправлення клієнта на іншу адресу або ресурс. Наприклад, 301 Moved Permanently означає, що ресурс був переміщений назавжди і клієнт повинен звертатися до нової адреси.

4xx – Статуси помилок клієнта: ці статуси показують, що виникла помилка на боці клієнта. Наприклад, 404 Not Found означає, що запитуваний ресурс не знайдено на сервері.

5xx – Статуси помилок сервера: ці статуси показують, що виникла помилка на боці сервера. Наприклад, 500 Internal Server Error означає, що сталася внутрішня помилка сервера при обробці запиту.

Кожен серверний статус має своє призначення і використовується для передачі певної інформації про стан запиту. Розуміння цих статусів допомагає розробникам та адміністраторам серверів ефективно взаємодіяти з клієнтами і вирішувати проблеми, які виникають під час обробки запитів.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1 Опис інструментів розробки

Для реалізації були використані стек технологій .NET, React Native, Visual Studio, C#, ASP.NET Core, MS SQL DB, Swagger, Postman, Azure, .NET CLI, Nuget, Chrome DevTools та набір технологій для парсингу даних.

C# – це об’єктно-орієнтована мова програмування, що використовується в екосистемі .NET. Вона дозволяє розробникам створювати різноманітні програми для платформ Windows, веб-додатків, служб та інших рішень. Розроблений у 1998-2001 роках групою інженерів під керівництвом Андерса Хейлсберга у компанії Microsoft як мова розробки програм для платформи Microsoft .NET Framework і згодом був стандартизований як ECMA-334 та ISO/IEC 23270.C# відноситься до сім’ї мов з C-подібним синтаксисом, з них його синтаксис найближчий до C++ та Java. Мова має статичну типізацію, підтримує поліморфізм, навантаження операторів (у тому числі операторів явного та неявного приведення типу), делегати, атрибути, події, властивості, узагальнені типи та методи, ітератори, анонімні функції з підтримкою замикань, LINQ, винятки, коментарі в формат XML.

C#, мова програмування, багато вчилася від своїх попередників, таких як C++, Pascal, Modula, Smalltalk і Java, і спирається на їхню практику. Проте, вона також уникає деяких моделей, які виявилися проблематичними при розробці програмних систем. Наприклад, C# не підтримує множинне успадкування класів, але дозволяє множинне успадкування інтерфейсів. C# був розроблений як мова програмування прикладного рівня для Common Language Runtime (CLR) і, як такий, залежить від можливостей CLR [7]. Основна система типів C#, яка відображає Base Class Library (BCL), також залежить від CLR. Мовні особливості C# визначаються можливістю їх перекладу в відповідні конструкції CLR. З розвитком CLR від версії 1.1 до 2.0, мова C# значно збагатилася. Очікується

подальше збагачення мови відповідно до розвитку CLR. Зауважимо, що з виходом C# 3.0 було порушено цю закономірність, оскільки ця версія мови представляє собою розширення, яке не спирається на розширення платформи. CLR надає C# (а також іншим мовам .NET) багато можливостей, які відсутні в «класичних» мовах програмування. Наприклад, збирання сміття не реалізовано безпосередньо в C#, але виконується CLR для програм, написаних на C#, аналогічно до того, як це відбувається для програм на VB.NET, J# та інших мовах.

Visual Studio – це інтегроване середовище розробки (IDE) для платформи .NET. Воно надає розробникам зручні інструменти для написання, налагодження та візуального проєктування програм на різних мовах, включаючи C#.

React Native – це фреймворк для розробки мобільних додатків, який базується на бібліотеці React. Використовуючи React Native, розробники можуть писати кросплатформний код на JavaScript, що дозволяє створювати мобільні додатки для iOS та Android. React Native дозволяє розробникам створювати компоненти, які можна повторно використовувати із високою продуктивністю та нативними можливостями. За допомогою React Native можна розробляти нативні мобільні додатки з високою продуктивністю та нативними функціями, такими як камера, геолокація та інші. Фреймворк також надає розробникам можливість збільшити продуктивність своїх додатків, оскільки він використовує реконціліацію віртуального DOM, яка дозволяє оновлювати тільки необхідні елементи інтерфейсу. Крім того, React Native підтримує гарячу перезавантаження, що дозволяє швидко побачити зміни в реальному часі без перезавантаження додатку. Завдяки своїй популярності та активній спільноті розробників, React Native має велику кількість додаткових плагінів та компонентів, що полегшують розробку і розширення функціональності додатків. Також, React Native є чудовим вибором для команд, що працюють над багатьма платформами, оскільки значна частина коду може бути спільно використана на різних платформах, що збільшує ефективність розробки та зменшує витрати часу і зусиль.

ASP.NET Core – це веб-фреймворк, розроблений компанією Microsoft, призначений для створення веб-додатків і веб-сервісів [1]. Цей фреймворк є наступником попередньої версії ASP.NET і має багато переваг і поліпшень. Однією з ключових переваг ASP.NET Core є його крос-платформеність [2]. Фреймворк підтримує роботу на різних операційних системах, таких як Windows, macOS та Linux, що дає можливість розробляти веб-додатки на будь-якій обраній платформі. ASP.NET Core забезпечує широкі можливості для розробки веб-додатків різного типу. Зокрема, він підтримує розробку веб-сторінок з використанням HTML, CSS і JavaScript, а також розробку веб-сервісів (API) для обробки і передачі даних [3]. Для роботи з API часто використовується ASP.NET Core WebAPI, який надає зручні засоби для створення RESTful API. ASP .NET WebAPI забезпечує широкі можливості для реалізації контролерів, які обробляють HTTP запити та повертають дані у вигляді JSON або XML [8].

MS SQL DB (Microsoft SQL Database) – це система управління базами даних, розроблена компанією Microsoft [4]. Вона є однією з найпопулярніших реляційних баз даних і широко використовується в багатьох веб-додатках і підприємствених системах [5]. MS SQL DB надає надійну та масштабовану роботу з даними, підтримує мову запитів SQL для маніпулювання даними і має багато інструментів для адміністрування та оптимізації бази даних [6].

Entity Framework Code First – це підхід до розробки програмного забезпечення, що використовує Entity Framework (EF) для взаємодії з базою даних. Цей підхід дозволяє розробникам визначати модель даних у вигляді класів (класи-сутності), а потім на основі цих класів автоматично створювати схему бази даних. У режимі Code First, розробники визначають класи-сутності, які відображають таблиці бази даних, а також зв’язки між ними. Entity Framework використовує цю модель класів для створення або оновлення схеми бази даних, що відповідає цим класам. Крім того, Entity Framework Code First надає можливості для налаштування взаємодії з базою даних, включаючи визначення обмежень, індексів, збережених процедур і т.д. За допомогою атрибутів або Fluent API, розробники можуть вказати додаткові деталі про модель даних і

взаємодію з базою даних. Один з ключових принципів Code First – це автоматична міграція бази даних. При зміні моделі даних (додавання, видалення або зміна властивостей класів-сутностей), Entity Framework може автоматично створити скрипти для оновлення схеми бази даних, що забезпечує зручну роботу з розвитком бази даних в процесі розробки. Entity Framework Code First надає зручний спосіб розробки програмного забезпечення, що використовує базу даних. Він дозволяє розробникам зосередитися на моделюванні даних у вигляді класів і мінімізує необхідність безпосередньо працювати з SQL-запитами і мануальною роботою з базою даних.

Swagger – це набір інструментів для створення, документування та використання RESTful API. Він надає можливість автоматично створювати документацію для API на основі його коду і опису ресурсів. Swagger також дозволяє виконувати тестування API, візуалізувати його структуру та взаємодіяти з ним через веб-інтерфейс. Завдяки Swagger розробники можуть легко спілкуватися та співпрацювати при роботі з API, а також забезпечувати його документацію для сторонніх користувачів.

Postman – це популярний інструмент для тестування та взаємодії з API. Він дозволяє розробникам створювати HTTP запити до API, виконувати їх, переглядати та аналізувати відповіді сервера. Postman надає зручне середовище для налагодження, тестування і належного функціонального тестування API. Він також підтримує автоматизацію тестування та можливість створювати колекції запитів для організації роботи з багатьма API.

Azure – це хмарна платформа, розроблена компанією Microsoft, яка надає набір послуг для розгортання, керування та розширення веб-додатків та інфраструктури. Azure пропонує широкий спектр сервісів, таких як веб-хостинг, бази даних, аналітика, інтеграція, машинне навчання та багато інших. Ця платформа дозволяє розробникам швидко та ефективно розгортати свої додатки в хмарі, масштабувати їх ресурси в залежності від потреб і отримувати надійність і безпеку, які надаються Microsoft. Azure є популярним вибором для розробки та розгортання ASP.NET Core додатків.

Chrome DevTools – це набір інструментів для розробників, що надається в браузері Google Chrome. Вони дозволяють розробникам аналізувати, налагоджувати та оптимізувати веб-сторінки та веб-додатки. Chrome DevTools надає розширені можливості для перевірки та відлагодження коду, відстеження мережевих запитів, профілювання продуктивності, зміни елементів сторінки в режимі реального часу та багато іншого. Вони є потужним інструментом для веб-розробників, що допомагає покращити продуктивність та якість веб-додатків.

.NET CLI (Command Line Interface) – це інструментарій командного рядка для розробки, збирання та керування проєктами на платформі .NET. Завдяки .NET CLI розробники можуть створювати нові проєкти, компілювати їх, запускати, тестувати та розгортати додатки з використанням команд терміналу. .NET CLI надає простий та уніфікований спосіб управління проєктами на платформі .NET та є невід'ємною частиною розробки .NET-додатків.

NuGet – це менеджер пакетів для платформи .NET, що дозволяє розробникам легко встановлювати, оновлювати та керувати залежностями пакетів в їхніх проєктах. Використання NuGet дозволяє легко і швидко додавати сторонні бібліотеки, компоненти та інші ресурси до проєкту .NET. Він має велику кількість доступних пакетів, які можна встановити з централізованого репозиторію NuGet.

HtmlAgilityPack, HttpClient, XPath і JsonPath – це набір інструментів, які часто використовуються в розробці програмного забезпечення для отримання, обробки та аналізу веб-даних.

HtmlAgilityPack – це бібліотека для роботи з HTML-документами у .NET. Вона дозволяє отримувати доступ до різних елементів HTML-сторінки, зчитувати і записувати дані, модифікувати структуру сторінки і виконувати різні маніпуляції з HTML-документами. HtmlAgilityPack дозволяє зручно взаємодіяти з HTML-кодом, використовуючи різні методи та властивості.

HttpClient – це клас у .NET, який надає можливості для відправки HTTP-запитів до веб-серверів і отримання відповідей. HttpClient дозволяє розробникам взаємодіяти з різними веб-ресурсами, виконувати GET, POST, PUT, DELETE-

запити і отримувати дані з веб-серверів. Він забезпечує широкий функціонал для роботи з HTTP-протоколом.

XPath – це мова запитів, яка використовується для навігації і вибору елементів у XML-документах [10]. XPath дозволяє розробникам точно визначати шлях до певних елементів XML, використовуючи певні правила та патерни. Це дозволяє зручно отримувати доступ до конкретних даних в XML-структурі.

JsonPath – це синтаксис для вибору елементів з JSON-документів. Він надає можливість точно визначити шлях до певних елементів у JSON-структурі даних, використовуючи патерни та оператори [11]. З JsonPath можна витягти конкретні значення з великих JSON-об’єктів, робити фільтрацію, сортування та інші маніпуляції з даними.

Комбінування HtmlAgilityPack, HttpClient, XPath і JsonPath дозволяє розробникам легко отримувати доступ до веб-даних, отримувати HTML-сторінки або JSON-відповіді з веб-серверів, а потім здійснювати аналіз цих даних, вибирати потрібні елементи за допомогою XPath або JsonPath і обробляти їх відповідно до вимог додатку. Це дозволяє розробникам ефективно працювати з веб-даними і забезпечувати потрібну функціональність своїм додаткам.

В цьому проєкті для додавання будь якого складa, ресурса продажів або базового каталога, де підтягування даних буде через XML або JSON – потрібно заповнити форму, де потрібно вказати в синтаксисі XPath \ JSONPath теги або елементи, до яких потрібно звертатись, щоб отримати необхідні дані. Це дозволяє зручно вказувати шлях до конкретних тегів або елементів у налаштуваннях ресурсу \ складу \ каталогу, щоб отримати потрібну інформацію. При заповненні форми для додавання складу, ресурсу продажів або базового каталогу, розробник може вказати XPath або JSONPath відповідно до структури документа, з якого потрібно отримати дані. За допомогою XPath або JSONPath вказується шлях до тегів, елементів або полів, які містять необхідну інформацію. Це дозволяє розробнику гнучко налаштувати звернення до даних у вхідному XML або JSON, і отримувати саме ту інформацію, яка необхідна для проєкту. Завдяки цьому підходу, проєкт може ефективно працювати з різними джерелами даних, витягати потрібні дані за допомогою специфічних шаблонів XPath або

JSONPath і використовувати ці дані в подальшій обробці бізнес логікою системи. Такий підхід до отримання даних забезпечує гнучкість та розширюваність проєкту, оскільки замовник може змінювати або додавати нові XPath або JSONPath вирази в залежності від потреб. Це спрощує процес додавання нових складів \ ресурсів \ каталогів, оскільки зміни в структурі вхідних даних можуть бути легко відображені у виразах XPath або JSONPath без необхідності змінювати код додатку. Усе це дозволяє розробникам ефективно використовувати потужні можливості XPath та JSONPath для взаємодії з XML або JSON даними, отримувати необхідні дані та динамічно працювати з ними в рамках проєкту.

3.2 Процес створення React Native компонента

Для створення React Native-компонента необхідно налаштувати середовище розробки та встановити залежності, такі як Node.js, React Native CLI, Expo CLI, Mobile SDKs, тощо. Потім можна створювати нові компоненти з використанням JSX, який є розширенням JavaScript, що дозволяє описувати інтерфейс користувача.

Процес створення React Native-компонента включає наступні кроки:

- створення нового проєкту React Native з використанням команди створення проєкту в командному рядку або інструментів розробки;
- визначення структури компонента, включаючи розміщення елементів, стилізацію, обробку подій та інші необхідні функції;
- використання компонентів React Native для побудови інтерфейсу користувача, таких як View, Text, Button, Image, тощо;
- налаштування взаємодії компонента з іншими компонентами або зовнішніми джерелами даних, такими як API-сервіси;
- тестування компонента на різних пристроях, використовуючи емулятори або фізичні пристрої.

3.3 Процес створення Entity Framework Code First-моделі

Для створення Entity Framework Code First-моделі необхідно налаштувати проєкт та встановити необхідні залежності, такі як Entity Framework [9].

Процес створення Entity Framework Code First-моделі включає такі кроки:

– визначення класів моделі, які представляють таблиці бази даних;
– визначення взаємозв’язків між класами, таких як один до одного, один до багатьох, багато до багатьох;
– використання атрибутів або Fluent API для налаштування деталей моделі, таких як назви стовпців, обмеження, тощо;
– конфігурування зв’язку з базою даних, включаючи вибір провайдера бази даних та налаштування рядка підключення;
– застосування міграцій для створення або оновлення схеми бази даних з використанням механізму міграцій Entity Framework;
– тестування моделі на взаємодію з базою даних та відповідність вимогам додатку.

3.4 Процес створення ASP.NET WebAPI-контролера

Для створення ASP.NET WebAPI-контролера необхідно налаштувати проєкт та встановити необхідні залежності, такі як ASP.NET WebAPI Framework.

Процес створення ASP.NET WebAPI-контролера включає такі кроки:

– створення нового контролера з використанням шаблонів або генераторів коду, що надаються ASP.NET WebAPI Framework;
– визначення методів дій (actions) контролера, які відповідають на HTTP-запити, такі як GET, POST, PUT, DELETE;
– налаштування маршрутизації, що визначає URL-шаблони, за якими будуть доступні методи дій контролера;

- використання атрибутів, таких як [HttpGet], [HttpPost], [Route], для налаштування різних аспектів контролера, таких як тип HTTP-запиту, URL-шаблон, тощо;
- реалізація логіки дій контролера, включаючи обробку вхідних параметрів, виклик сервісів або репозиторіїв, формування відповіді;
- тестування контролера з використанням інструментів для тестування API, таких як Postman або Swagger.

Ці процеси можуть включати додаткові кроки в залежності від специфіки проєкту та використаних технологій, але наведені кроки є основними для реалізації та тестування Entity Framework Code First-моделі та ASP.NET WebAPI-контролера.

3.5 Тестування системи (backend, frontend)

Тестування системи, включаючи як бекенд, так і фронтенд, є важливою частиною розробки програмного забезпечення. Вона допомагає перевірити, чи відповідає система вимогам, функціонує правильно і не містить помилок або проблем.

Тестування системи включає наступні етапи, які розглянемо нижче.

Планування тестування: на цьому етапі визначаються цілі тестування, обсяг тестування, розроблюється тестова стратегія і план, а також визначаються тестові набори і критерії прийняття.

Функціональне тестування: функціональне тестування перевіряє, чи відповідає система функціональним вимогам. Це включає перевірку роботи окремих функцій, взаємодію між компонентами, обробку вхідних даних та виведення результатів.

Тестування інтеграції: на цьому етапі перевіряється, чи правильно працюють компоненти системи разом. Виконуються тестові сценарії, що

охоплюють взаємодію між бекендом і фронтендом, перевірку обміну даними, інтеграцію з базою даних та іншими системами.

Тестування продуктивності: це тестування, що оцінює продуктивність системи за допомогою різних навантажень і сценаріїв використання. Можуть використовуватися інструменти для вимірювання швидкодії, часу відгуку, витрат ресурсів і реагування системи на велику кількість користувачів.

Тестування безпеки: на цьому етапі перевіряється система на наявність потенційних проблем безпеки, таких як уразливості, витоки даних, недостатні заходи безпеки. Застосовуються різні методи тестування, включаючи вторгнення, перехоплення даних та аналіз вразливостей.

Тестування відповідності: на цьому етапі перевіряється, чи відповідає система вимогам, стандартам і регуляторним вимогам, які можуть стосуватися конкретної галузі або ринку.

Валідація та верифікація: цей етап включає перевірку, чи виконується система відповідно до очікувань користувачів і бізнес-вимог, а також валідацію вхідних та вихідних даних.

Документування результатів: після завершення тестування важливо документувати результати, виявлені проблеми, заплановані виправлення і рекомендації для поліпшення системи.

3.5.1 Unit тести

Модульне тестування, або юніт-тестування – процес у програмуванні, що дозволяє перевірити на коректність окремі модулі вихідного коду програми.

Ідея полягає в тому, щоб писати тести для кожної нетривіальної функції чи методу. Це дозволяє досить швидко перевірити, чи не привела чергова зміна коду до регресії, тобто до появи помилок у вже відтестованих місцях програми, а також полегшує виявлення та усунення таких помилок.

3.6 Керівництво користувача

3.6.1 Рівень підготовки користувача

Для роботи на сайті користувач повинен мати певну кваліфікацію, включаючи навички роботи з ПК та web-браузером. Це охоплює вміння працювати з операційною системою, встановлювати програми, працювати з файловою системою, а також розуміння основних понять та операцій в браузері.

Крім того, знайомство з керівництвом користувача допоможе зрозуміти функціонал сайту та використовувати його відповідно до інструкцій.

3.6.2 Підготовка до роботи

Запуск системи. Для запуску системи необхідний доступ до сайту через інтернет за допомогою стандартного web-браузера, такого як Google Chrome, Mozilla Firefox, Opera або Safari.

Після входу на сайт користувач буде перенаправлений на сторінку входу до системи. На сайті розрізняються дві групи користувачів: анонімні користувачі, які не увійшли або не зареєстровані, і користувачі – приватні особи, які авторизувалися на сайті та мають доступ до функцій користувача у системі.

3.6.3 Авторизація, вхід в систему

Авторизація та вхід в систему є важливим функціоналом для забезпечення безпеки та обмеження доступу до конфіденційної інформації. Для керівництва користувача надаються такі деталі:

а) вхід в систему:

- користувач вводить свої облікові дані (електронну пошту та пароль) на сторінці входу в систему;
- система перевіряє правильність введених даних та перевіряє, чи існує користувач з такими обліковими даними;
- при вдалій авторизації користувачу надається доступ до захищених ресурсів та функціоналу системи;

б) відновлення паролю:

- якщо користувач забув свій пароль, він може скористатися функцією відновлення паролю;
- зазвичай це включає введення електронної пошти, пов’язаної з обліковим записом;
- система надсилає на вказану електронну пошту інструкції щодо відновлення паролю;
- користувач слідує цим інструкціям, щоб встановити новий пароль для свого облікового запису;

в) захист від несанкціонованого доступу:

- для забезпечення безпеки, система використовує різні механізми, такі як шифрування паролів та механізми аутентифікації;
- зазвичай, система також встановлює обмеження на кількість невдалих спроб входу перед тим, як тимчасово заблокувати обліковий запис;
- користувачам рекомендується використовувати міцні паролі та забезпечувати їх конфіденційність.

Розробник повинен ретельно виконувати всі заходи безпеки та передбачати потенційні загрози, пов’язані з авторизацією та входом в систему. Користувачам слід бути обізнаними про необхідність використання надійних паролів та правил безпеки для збереження цілісності свого облікового запису та конфіденційності особистих даних.

На рисунку 3.1 представлена сторінка авторизації в системі.

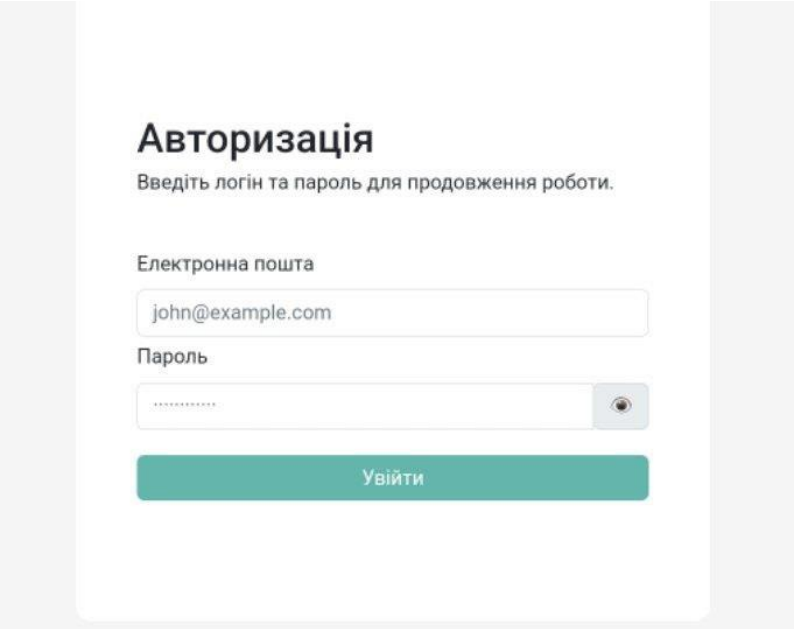

Рисунок 3.1 – Авторизація в системі

3.6.4 Робота з товарами

В цьому блоку відображається список товарів, з якими в подальшому будуть здійснюватись операції. Включає наступні поля:

– номер: унікальний номер, який присвоюється кожному товару при імпорті даних в систему;
– категорія: головна категорія, до якої обов’язково належить кожний товар, вона є батьківською для Підкатегорії-1;
– підкатегорія-1: не обов’язкове поле, присутня не у всіх товарів, вона є батьківською для Підкатегорії-2;
– підкатегорія-2: не обов’язкове поле, присутня не у всіх товарів, вона не є батьківською;
– фото: для кожного товару в YML-файлі є посилання на фото, в цьому полі відображається саме фото, а не посилання;
– артикул: унікальний ідентифікатор для кожного товару, він використовується для ідентифікації кожного товару;
– назва: назва товару, яка також міститься в YML-файлі, і з якою будуть

пов’язані подальші операції.

В даному блоку відображається список товарів з відповідними характеристиками, які будуть використовуватись для автоматизації зміни ціни та наявності товарів на ресурсах продажу.

На рисунку 3.2 представлено сторінку роботи з товарами.

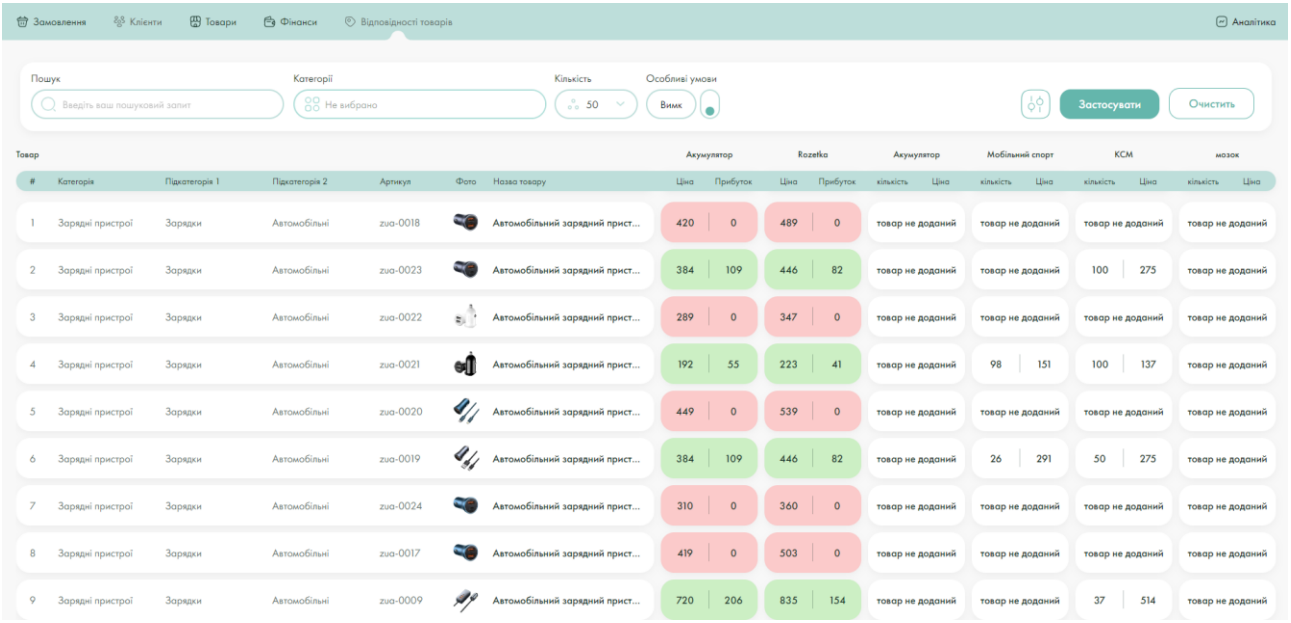

Рисунок 3.2 – Робота з товарами

3.6.5 Робота зі складами

Переходячи в розділ «Налаштування» -> «Відповідності товарів» та обравши у лівому меню розділ «Склади», відкриється сторінка з таблицею підключених складів до системи.

Вгорі розміщена таблиця «Список складів» з наступними пунктами:

– номер;
– назва;
– власник (два варіанти вибору: «власний» або «партнера»);
– джерело (звідки беруться дані, на цьому етапі розробки повинно бути два варіанти: «YML-файл або JSON»);

– кількість товарів (загальна кількість товарів у джерелі);
– додано (кількість доданих товарів з наявних у джерелі до нашої системи, стосується складів-партнерів, оскільки не всі товари можуть бути відповідними за артикулами);
– пріоритет (порядок, у якому виводиться склад у вищезазначеній таблиці);
– статус (варіанти: «увімк» або «вимк», якщо вимкнено, склад не повинен відображатися в основній «візуальній» таблиці).

Дії. Будуть присутні наступні дії:

– змінити (при натисканні на кнопку відкривається заповнена карточка складу, як при додаванні);
– видалити (появляється форма для підтвердження видалення. Якщо обрано «так», склад видаляється);
– відповідності артикулів (цей пункт буде доступний ТІЛЬКИ для складів-партнерів, тут буде здійснюватися відповідність артикулів партнерів з нашими);
– нижче таблиці буде розміщена кнопка «Додати склад», при натисканні кнопки з’явиться випадаюче вікно зі змістом.

На рисунку 3.3 представлено сторінку перегляду складів.

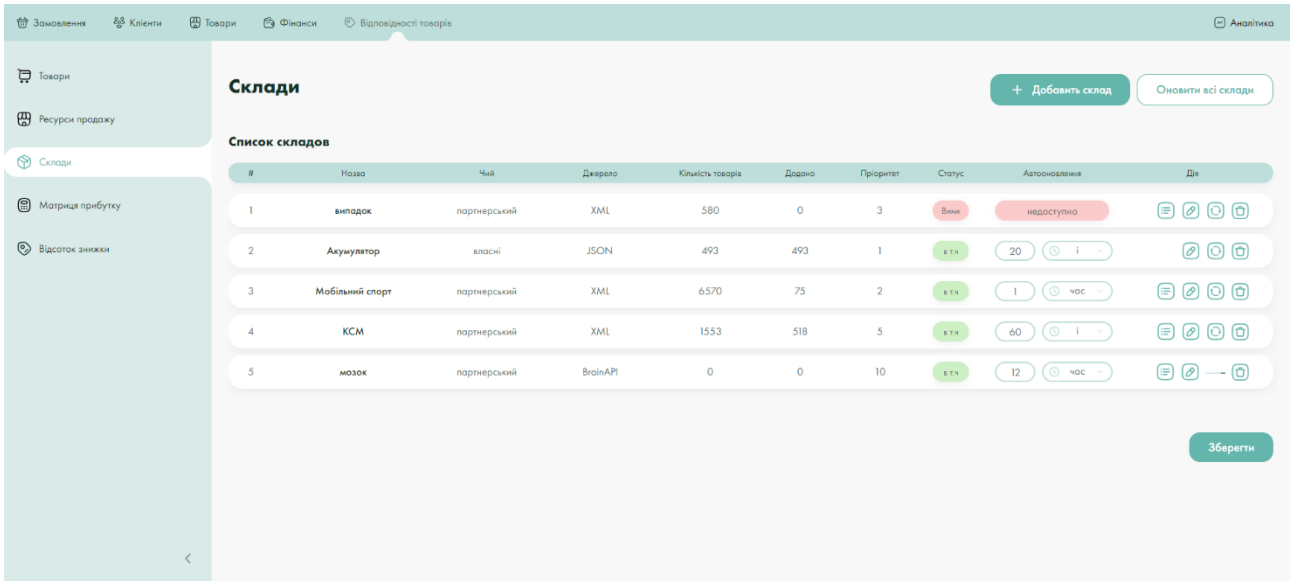

Рисунок 3.3 – Перегляд доступних складів

У даній формі на рисунку 3.4 вводиться назва і обирається джерело даних: «YML-файл» або «JSON». Вводиться посилання і встановлюються відповідності артикулів. Нижче з’являється блок «Додаткові». Поле «Тип складу» має два варіанти: «власний» або «партнера». Якщо обрано «власний», додаткових полів не з’явиться. Якщо обрано партнера, поряд з ним з’являється поле для вибору валюти. Тут буде 3 варіанти: «грн», «долар», «євро». При виборі «грн» введення завершується. При виборі будь-якої валюти форма доповнюється наступним чином: Тут потрібно буде вибрати тип курсу, варіанти: «НБУ», «власний». Якщо обрано курс НБУ, просто отримуємо офіційний курс валюти з банку, власний – він підтягується з посилання за допомогою XPath \ JsonPath.

На рисунку 3.4 представлено сторінку створення нового складу.

Рисунок 3.4 – Створення нового складу

Співставлення артикулів зі складами-партнерами. У партнерів – артикули на товари встановлені не ті, які є в базовому каталогу, а їх власні. Тому нам було потрібно реалізувати можливість встановлення відповідності артикулів. При відкриття цього розділу зверху та знизу є дві таблиці – товари зі складу та товари з базового каталогу. Треба обрати товар зі складу-партнера, та система, враховуючи лексикографічні особливості назв товарів – знайде найбільш схожі товари за ім’ям та відсортує їх в порядку спаду (чим вище – тим більший шанс що товар тей самий). Також система підсвітить однакові слова в назвах товарів базового каталогу згідно з назвою вибраного товару в таблиці товарів складу-партнера.

На рисунку 3.5 представлено сторінку співставлення артикулів.

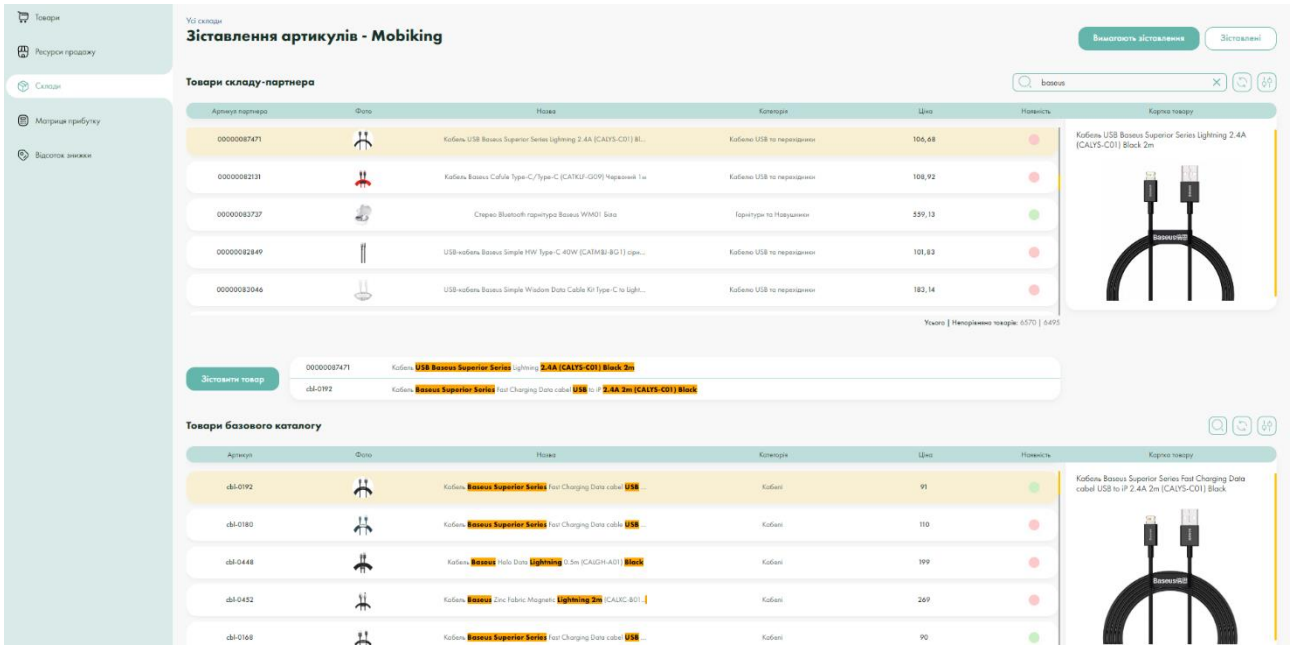

Рисунок 3.5 – Співставлення артикулів поміж складом та базовим каталогом

3.6.6 Робота з джерелами продажу

Переходячи в розділ «Налаштування» -> «Відповідності товарів» та вибравши в лівому меню пункт «Ресурси продажу», відкриється сторінка з

можливістю перегляду ресурсів продажу. Тут є можливість продивлятись створені ресурси продажів, дивитись особливі умови по кожному ресурсу окремо (особливі умови по категоріям та по продуктах окремо).

На рисунку 3.6 представлено сторінку перегляду ресурсів продажу.

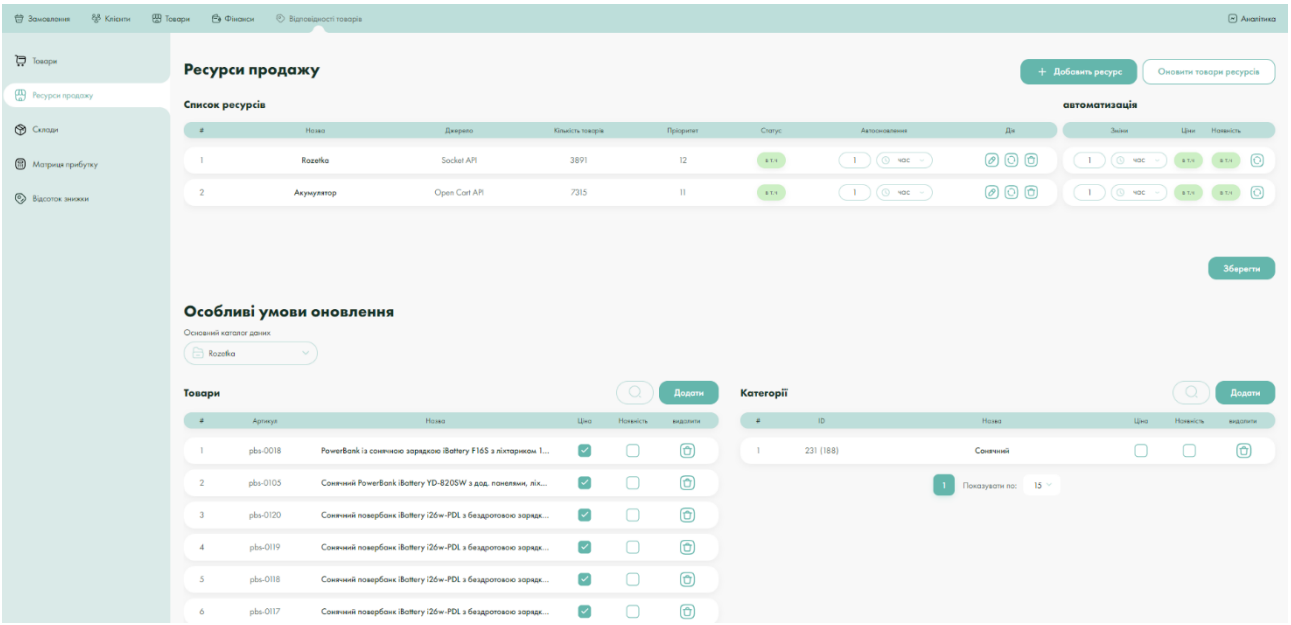

Рисунок 3.6 – Перегляд ресурсів продажу

Нижче таблиці розміщена кнопка «Додати склад». На рисунку 3.7 представлено випадаюче вікно, яке з’являється при натисканні кнопки.

В цій формі є можливість заповнити XPath \ JsonPath поля аналогічно як і на складах, та створити новий ресурс продажів, по якому буде можливість отримувати та оновлювати дані в залежності від типу ресурсу продажів.

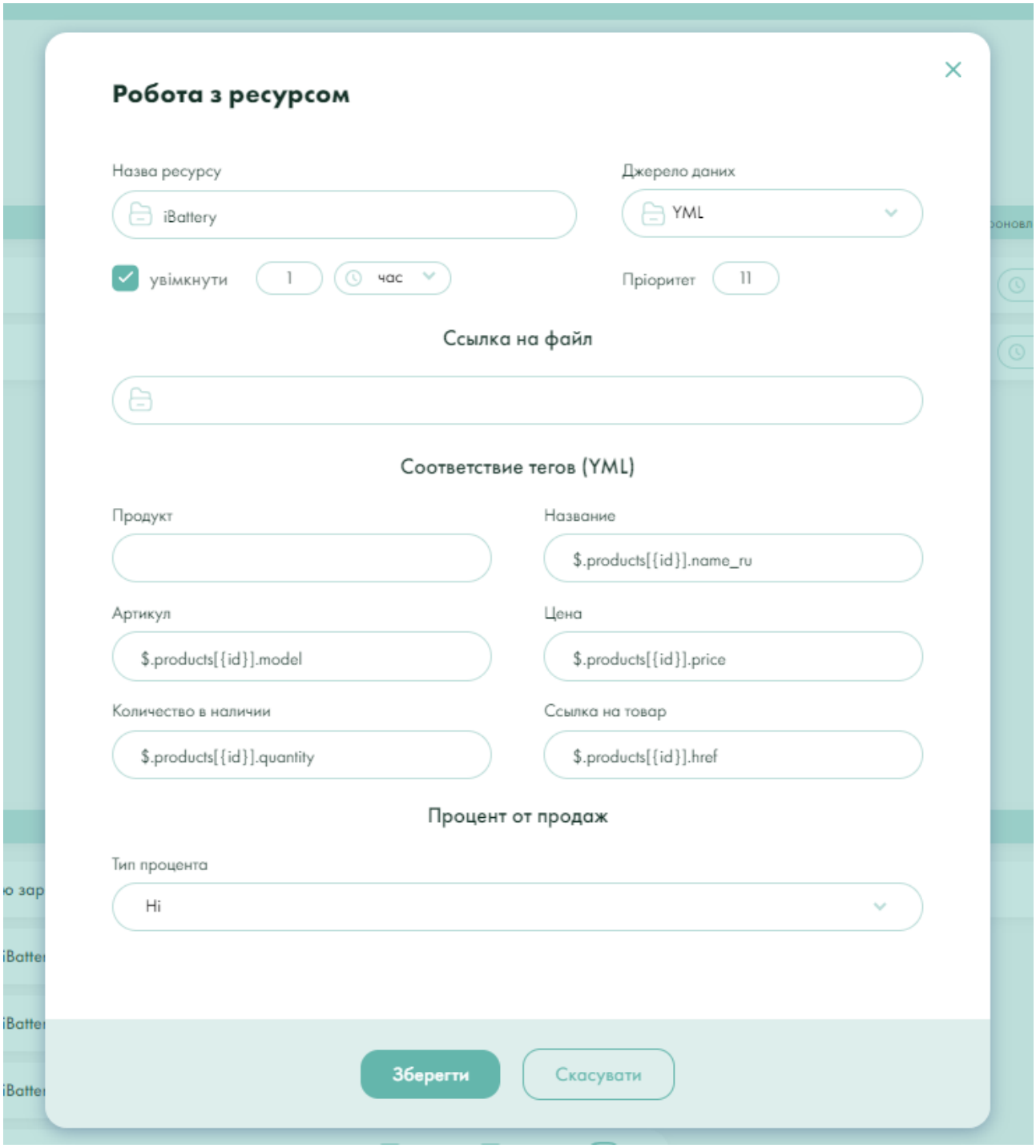

Рисунок 3.7 – Створення нового ресурсу продажу

ВИСНОВКИ

В результаті роботи було написано технічне завдання на розробку системи автоматизації зміни ціни та наявності товарів на ресурсах продажу (магазини, маркетплейси). Для створення цієї системи були обрані такі технології:

- .NET та Visual Studio для розробки back end;
- C# та ASP .NET Core для реалізації API;
- MSSQL DB для зберігання даних;
- Entity Framework для доступу к базі даних;
- Rect Native для розробки front end.

Згідно з поставленими завданнями, були виконані такі етапи створення системи:

- сформульовані вимоги до системи (функціональні та нефункціональні (інтерфейс, кросбраузерність, безпека, продуктивність)), також проведено огляд предметної області та інструментів розробки;
- спроєктована та побудована структура системи (розроблені діаграми прецедентів, діяльності та послідовності; надано детальний опис прецедентів);
- реалізована система автоматизації зміни ціни та наявності товарів на ресурсах продажу;
- надано інструкцію по створенню компонентів системи, керівництво користувача та структуру проєкту;
- протестована робота системи.

ПЕРЕЛІК ПОСИЛАНЬ

1. JSON Web Token Authentication. Readthedocs. URL: https://jwt-auth.readthedocs.io/en/develop/ (дата звернення: 03.03.2023).
2. Офіційна документація ASP.NET Core. Microsoft Developer Portal. URL: https://docs.microsoft.com/en-us/aspnet/core/ (дата звернення: 14.03.2023).
3. ASP.NET Core Repository. GitHub. URL: https://github.com/dotnet/aspnetcore (дата звернення: 11.03.2023).
4. Freeman A. Pro ASP.NET Core MVC 2. Apress, 2016. 1018 p.
5. Офіційна документація Microsoft SQL Server. Microsoft Developer Portal. URL: https://docs.microsoft.com/en-us/sql/sql-server/ (дата звернення: 05.04.2023).
6. SQL Server Central. URL: https://www.sqlservercentral.com/ (дата звернення: 06.05.2023).
7. Petkovic D. Microsoft SQL Server 2019: A Beginner's Guide. McGraw Hill, 2020. 864 p.
8. CSharp Complete Guide. Metanit. URL: https://metanit.com/sharp/tutorial/ (дата звернення: 15.02.2023).
9. Introduction to ASP .NET Identity. Codeguru. URL: https://www.codeguru.com/dotnet/asp-net-identity-library/ (дата звернення: 02.03.2023).
10. Entity Framework Overview. Microsoft Developer Portal. URL: https://learn.microsoft.com/en-us/ef/core/ (дата звернення: 13.04.2023).
11. XPath Documentation. Mozilla Developer Portal. URL: https://developer.mozilla.org/en-US/docs/Web/XPath (дата звернення: 23.04.2023).
12. JSONPath Syntax. Smartbear. URL: https://support.smartbear.com/alertsite/docs/monitors/api/endpoint/jsonpath.htmlв (дата звернення: 25.04.2023).