m-math_2025_002

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЧЕРНІВЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ЮРІЯ ФЕДЬКОВИЧА

Факультет математики та інформатики
кафедра математичного моделювання

Онлайн сервіс для організації корпоративного висаджування дерев

Кваліфікаційна робота
Рівень вищої освіти – другий (магістерський)

Виконав:
студент 2 курсу, 601 групи
Дем'янчук Іван Васильович
Керівник:
кандидат технічних наук Шкільнюк Д.В.

До захисту допущено
на засіданні кафедри
протокол № 5 від 26 листопада 2024 р.
Зав. кафедрою _____ проф. Черевко І.М.

Чернівці – 2024

АНОТАЦІЯ

Магістерська робота присвячена створенню онлайн-сервісу для корпоративної висадки дерев у містах, що об'єднує бізнес, благодійні організації та місцеві громади. Сервіс спрощує організацію озеленення міських територій та дозволяє користувачам відстежувати результати проєкту. Продукт розроблено з використанням REST API, Angular та Node.js, забезпечуючи зручний і функціональний інтерфейс.

ANNOTATION

The master’s thesis is dedicated to creating an online service for corporate tree planting in urban areas, bringing together businesses, charitable organizations, and local communities. The service simplifies the organization of urban greening and allows users to track project outcomes. The product is developed using REST API, Angular, and Node.js, providing a convenient and functional interface.

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів наукових досліджень інших авторів мають посилання на відповідне джерело.

_____ І.В. Дем’янчук
(підпис)

ЗМІСТ

ВСТУП

Зелені зони є важливим компонентом міського середовища, що сприяє формуванню здорового та комфортного життєвого простору для мешканців. Однак через активну урбанізацію, інтенсивну забудову та втручання людини площа зелених насаджень у містах постійно скорочується. Це веде до погіршення якості повітря, збільшення теплового навантаження та зростання негативного впливу на фізичне і психічне здоров'я населення. Відновлення та збільшення кількості зелених насаджень стало важливим екологічним завданням, спрямованим на покращення міського середовища та боротьбу з наслідками зміни клімату.

Сучасний, інноваційний підхід до цієї проблеми передбачає використання онлайн-сервісів для організації висадки дерев, які забезпечують зручність та прозорість процесу озеленення. Такі майданчики дозволяють приватним особам, компаніям та організаціям замовляти висадку дерев в обраних локаціях онлайн, координуючи свої зусилля з місцевими громадами та екологічними ініціативами. Це не лише гарантує, що дерева будуть висаджені в потрібних місцях, але й допомагає уникнути хаотичних або несанкціонованих посадок, забезпечуючи сталість проєкт з озеленення.

Завдяки таким сервісам мешканці можуть не тільки обирати вид дерева та місце його висадки, а й активно долучатися до екологічного розвитку свого міста. При цьому, висадка дерев відіграє ключову роль у зменшенні викидів вуглецю, природному очищенні повітря та регулюванні температури, що особливо важливо в умовах зростання глобальних екологічних загроз.

1. ОГЛЯД ТЕМИ, СУЧАСНІ ПІДХОДИ ДО РЕАЛІЗАЦІЇ, РОЗШИРЕНА ПОСТАНОВКА ЗАДАЧІ

1.1. Сучасні підходи до реалізації

Розробка програмного продукту розпочалася з детального аналізу предметної області, включаючи поглиблене вивчення сучасних тенденцій у сфері екологічних ініціатив, корпоративного озеленення та взаємодії між користувачами та організаціями. Після цього відбулося ретельне обговорення ключових вимог, які були розділені на три основні категорії: бізнес-вимоги, вимоги користувачів та вимоги до програмного забезпечення. Цей аналіз дозволив нам створити чіткий план реалізації проєкт, орієнтований на задоволення потреб усіх зацікавлених сторін.

За результатами аналізу однією з ключових вимог до розробки дизайну стало впровадження мінімалістичного підходу. Застосування яскравих кольорів, великих шрифтів та чітко структурованих елементів дизайну мало на меті зробити інтерфейс інтуїтивно зрозумілим та привабливим для користувачів. Важливо було забезпечити зручну навігацію, де всі основні функції додатка знаходяться в легкому доступі. Це мінімізувало складність використання системи, зробивши її доступною для користувачів з різним рівнем технічної підготовки.

Для реалізації клієнт-серверної архітектури ми обрали технології REST API на базі Node.js та Angular для користувацької частини. REST API забезпечує ефективну взаємодію вебдодатків з базою даних та іншими сторонніми сервісами, що робить систему гнучкою та легко масштабованою. Node.js забезпечує швидку роботу серверних додатків завдяки асинхронній обробці запитів, що знижує навантаження і підвищує продуктивність. Angular, своєю чергою, є потужним інструментом для розробки інтерактивних інтерфейсів, що дозволяє створювати динамічні та зрозумілі вебдодатки.

Важливим елементом системи є впровадження карти, яка дозволяє користувачам обирати конкретні місця для висадки дерев. Для цього ми обрали Google Maps API, який забезпечує точність і простоту взаємодії з

картою. Користувач може легко визначити, де саме планується висадити дерево, а система надає всю необхідну інформацію про обрану локацію.

Фінальним кроком було визначення стабільного рішення для зберігання даних. Було проведено аналіз сучасних баз даних, і в результаті було обрано PostgreSQL через її надійність, високу продуктивність та підтримку стандартів SQL. Це рішення гарантує сумісність з іншими базами даних і додатками, забезпечуючи легкий перехід на нові технології в майбутньому. PostgreSQL також відома своєю масштабованістю та гнучкістю в роботі з великими обсягами даних, що важливо для довгострокового функціонування та розвитку проєкт.

1.2. Призначення та область застосування

Розроблений онлайн-сервіс для організації корпоративної висадки дерев покликаний спростити процес озеленення міських територій шляхом активної взаємодії бізнесу, благодійних організацій та громадськості. Основна мета цього сервісу - надати зручний інструмент для організації та координації кампаній з висадки дерев, що дозволить компаніям та організаціям долучитися до екологічних ініціатив та зробити свій внесок у відновлення зелених зон у містах.

У сферу дії сервісу входять:

1. Корпоративне озеленення: Послуга дозволяє компаніям брати участь у благодійних екологічних проєктів, роблячи свій внесок у покращення міського середовища. Компанії можуть замовляти висадку дерев, координувати свої екологічні ініціативи та відстежувати їхній вплив.

2. Благодійні організації та екологічні ініціативи: Сервіс надає платформу для благодійних організацій, які займаються висадкою дерев. Вони можуть використовувати його для збору коштів, залучення нових членів та координації проєкт з озеленення.

3. Громадські ініціативи: Місцеві громади та окремі люди можуть долучитися до висадки дерев за допомогою сервісу, обираючи місця для посадки та підтримуючи ініціативи у своїх містах.

4. Екологічна освіта: Сервіс також можна використовувати для поширення знань про важливість озеленення, роль дерев у боротьбі зі зміною клімату та покращенні якості життя в містах.

Таким чином, сервіс розрахований на різних користувачів, об'єднуючи зусилля компаній, громадян та благодійних організацій на єдиній платформі для досягнення екологічних цілей та створення сталого міського середовища.

1.3. Опис основних алгоритмів

Онлайн-сервіс для замовлення висадки дерев містить низку специфічних алгоритмів, які забезпечують ефективну взаємодію користувача з системою, від реєстрації до завершення процесу висадки дерев. Основні алгоритми та вимоги до них включають:

1. Реєстрація користувача: Для користування додатком користувач повинен створити обліковий запис, який дозволить йому розміщувати замовлення на висадку дерев та відстежувати їх статус. Алгоритм реєстрації включає аутентифікацію даних користувача (через електронну пошту або соціальні мережі) та створення особистого профілю з історією замовлень.

2. Вибір місця та виду дерева: Користувач має можливість обрати місце для посадки дерева за допомогою інтегрованої карти (Google Maps API) та вибрати тип дерева зі списку доступних варіантів. Цей алгоритм використовує геолокацію для визначення відповідних ділянок для посадки та забезпечує динамічну взаємодію з базою даних для відображення доступних видів дерев.

3. Підтвердження замовлення: Після вибору місця та типу дерева користувач підтверджує своє замовлення, обираючи дату та час посадки. Алгоритм гарантує, що вибране дерево і час будуть доступні, щоб уникнути конфліктів замовлень.

4. Обробка замовлення: Після підтвердження замовлення обробляється сервером, який реєструє його в базі даних, генерує унікальний номер і готує відповідні повідомлення для користувача та підрядників.

5. Статус замовлення: Користувач може в будь-який момент перевірити статус замовлення. Алгоритм відстежує процес виконання замовлення від його створення до висадки дерева, оновлюючи статус в режимі реального часу.

6. Оплата: Додаток надає можливість безпечно оплатити замовлення через інтегровані платіжні сервіси. Алгоритм взаємодіє з платіжною системою, перевіряє транзакції та підтверджує оплату для користувача.

Окрім основних користувацьких алгоритмів, важливою частиною розробки стала робота з REST API, який забезпечує взаємодію між клієнтською та серверною частинами додатка. Основні алгоритми REST API:

1. Аутентифікація та авторизація: REST API забезпечує безпечний доступ до даних за допомогою механізмів аутентифікації (з використанням токенів або OAuth) та авторизації користувачів, що гарантує безпеку персональної інформації.
2. Отримання списку вільних місць: REST API надає функцію отримання актуального списку доступних місць для висадки дерев. Це дозволяє користувачам бачити лише ті ділянки, де можлива висадка дерев.
3. Отримання списку типів дерев: API надає можливість отримати інформацію про типи дерев, доступних для висадки, включаючи деталі про екологічні переваги кожного виду.
4. Отримання статусу замовлення: API дозволяє користувачам отримувати поточний статус своїх замовлень, надаючи інформацію про етапи виконання (обробка, підтвердження, виконання).
5. Створення замовлення: REST API реалізує функцію створення нового замовлення, включаючи вибір місця, типу дерева, дати і часу посадки. Цей запит генерує новий запис в базі даних з деталями замовлення.
6. Оновлення замовлення: Користувач має можливість оновити параметри свого замовлення через API, наприклад, змінити дату або час посадки, якщо це необхідно.
7. Видалення замовлення: API дозволяє користувачам скасувати замовлення на висадку дерев, що видаляє запис з бази даних та оновлює інформацію для підрядників.

Пояснення REST API та типи HTTP запитів:

REST API (Representational State Transfer Application Programming Interface) - це архітектурний стиль взаємодії між клієнтами та серверами, який

використовує стандартні HTTP-запити для обробки та передачі даних. Основними типами запитів, що використовуються в REST API, є

GET: Запит на отримання даних з сервера (наприклад, список доступних локацій або типів дерев). GET-запити не змінюють стан системи.

POST: Використовується для відправки даних на сервер, наприклад, при створенні нового замовлення.

PATCH: Використовується для часткового оновлення ресурсу, наприклад, для зміни дати або часу замовлення.

DELETE: Запит на видалення ресурсу, наприклад, скасування замовлення.

PUT: Використовується для повного оновлення ресурсу або його створення, якщо він не існує.

Ці алгоритми забезпечують ефективну, гнучку і безпечну роботу програми, дозволяючи користувачам взаємодіяти з усіма функціональними можливостями системи.

2. КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Використанні трехнології реалізації

Зі зростанням урбанізації та змінами клімату питання озеленення міських територій набуває особливого значення. Розробка онлайн-сервісу для корпоративного висаджування дерев є інноваційним рішенням, яке спрямоване на ефективну організацію цього процесу. Такий сервіс покликаний об'єднати зусилля бізнесу, благодійних організацій та громадськості, пропонуючи інструмент для зручної взаємодії та реалізації екологічних ініціатив. У цьому розділі розглянемо використані підходи й технології, які лягли в основу цього проєкту, та їхню роль у забезпеченні функціональності сервісу.

Вибір архітектури та загальні принципи

Основою сервісу є архітектура клієнт-сервер, яка передбачає розділення програмного продукту на фронтенд (клієнтську частину) і бекенд (серверну частину). Така архітектура обрана для забезпечення гнучкості, масштабованості та простоти інтеграції з іншими сервісами. Для взаємодії між клієнтом і сервером було обрано підхід REST API (Representational State Transfer Application Programming Interface), який базується на стандартних HTTP-запитах. Використання REST API дозволяє сервісу обробляти запити від клієнтів, повертати дані або результати операцій, а також гарантує можливість масштабування і легкості підтримки.

REST API став стандартом для побудови вебдодатків завдяки простоті і гнучкості. Він підтримує всі основні типи HTTP-запитів, що дозволяє розробникам легко інтегрувати додатковий функціонал або змінювати існуючий.

Технології фронтенду: Angular

Для розробки клієнтської частини було обрано фреймворк Angular. Angular — це потужний інструмент для побудови односторінкових додатків (SPA), який забезпечує високу продуктивність та зручність управління інтерфейсом. Переваги Angular полягають у:

- **Модульній структурі**: Angular дозволяє розбивати додаток на окремі модулі, що спрощує його розробку, тестування та підтримку.
- **Механізмі двостороннього зв'язку (Two-way Data Binding)**: Забезпечує автоматичне оновлення інтерфейсу користувача при зміні даних, що підвищує зручність і швидкість роботи.
- **Dependency Injection**: Angular має вбудовану систему ін'єкції залежностей, яка спрощує управління сервісами і даними всередині додатка.

У цьому проєкті Angular використовується для реалізації інтерактивного інтерфейсу з такими елементами, як інтеграція карти (Google Maps API) для вибору місць висадки дерев, панель для перегляду статусу замовлень, вибір видів дерев та іншими функціями. Angular також забезпечує адаптивність додатка для різних типів пристроїв, включаючи комп'ютери, планшети та смартфони.

Технології бекенду: Node.js і Express

Серверна частина додатка побудована на платформі Node.js з використанням фреймворка Express.js. Node.js — це платформа для виконання JavaScript-коду на сервері, що забезпечує високу продуктивність завдяки асинхронній обробці запитів. Express.js — це фреймворк для Node.js, який надає інструменти для створення API, налаштування маршрутів і обробки запитів.

Node.js і Express.js були обрані через наступні причини:

1. **Асинхронна обробка запитів**: Node.js дозволяє обробляти велику кількість одночасних запитів завдяки асинхронній природі, що значно покращує продуктивність.

2. **Легка інтеграція з базами даних**: Express.js забезпечує просту інтеграцію з різними базами даних, такими як PostgreSQL, MongoDB та ін.
3. **Широкий набір бібліотек та модулів**: Завдяки розвиненій екосистемі, розробники мають доступ до широкого спектра бібліотек для обробки запитів, аутентифікації користувачів, роботи з даними та багато іншого.

У цьому проєкті Express.js використовується для реалізації API, яке підтримує операції створення, оновлення, видалення та отримання даних про замовлення, місця посадки дерев і користувачів. Express.js також дозволяє швидко налаштувати маршрути для взаємодії з фронтендом.

База даних PostgreSQL

Для зберігання даних обрано реляційну базу даних PostgreSQL. PostgreSQL — це потужна система управління базами даних з підтримкою SQL, яка відома своєю надійністю, стабільністю та високою продуктивністю. Основними перевагами PostgreSQL є:

- **Підтримка складних SQL-запитів**: Це дозволяє легко витягувати, фільтрувати та маніпулювати даними за різними критеріями, що особливо важливо для моніторингу і звітування.
- **Надійність та відновлення**: PostgreSQL підтримує транзакції, що робить її надійним вибором для зберігання даних.
- **Масштабованість**: База даних може легко обробляти великі обсяги даних, що важливо для майбутнього зростання системи.

Використання PostgreSQL в цьому проєкті дозволяє зберігати дані про замовлення, види дерев, користувачів та місця посадки. Завдяки цій базі даних адміністратор сервісу може отримувати звіти про кількість висаджених дерев, види дерев, що користуються найбільшою популярністю, та відстежувати успіх проектів озеленення.

Інтеграція Google Maps API

Однією з важливих функцій сервісу є можливість вибору місця для висадки дерев на інтерактивній карті. Для цього було використано Google Maps API, що забезпечує:

- **Доступ до картографічних даних**: Користувачі можуть вибирати місця посадки дерев на карті, переглядати райони, доступні для озеленення, та отримувати дані про навколишні території.
- **Точність та надійність**: Google Maps API є одним із найбільш надійних інструментів для роботи з картами, що забезпечує точне відображення локацій.
- **Легка інтеграція з Angular**: Google Maps API має добре розроблену документацію для роботи з Angular, що спрощує налаштування і використання.

В цьому сервісі Google Maps API інтегровано для відображення місць посадки дерев та взаємодії з користувачами, що надає сервісу зручність і підвищує його функціональність.

Безпека та аутентифікація

Для забезпечення безпеки додатка було впроваджено механізми аутентифікації та авторизації користувачів. В основі аутентифікації — система токенів JWT (JSON Web Token), яка дозволяє надійно захищати доступ до особистих даних користувачів та API додатка.

JWT — це відкритий стандарт, що дозволяє передавати інформацію між сторонами у вигляді JSON-об'єкта. Основні переваги JWT:

- **Безпека**: Токени JWT підписані секретним ключем, що робить їх захищеними від підробки.
- **Простота використання**: Токени передаються у HTTP-заголовках, що полегшує аутентифікацію.
- **Масштабованість**: JWT може використовуватись на різних серверах без

необхідності зберігати сесію користувача.

У цьому проєкті JWT використовується для захисту API, що забезпечує безпеку.

Розміщення проекту за допомогою Docker

Для забезпечення простоти розгортання, масштабування та зручності розміщення проєкту на хостингу використовуються контейнери Docker. Docker дозволяє упаковувати застосунок разом з усіма залежностями, бібліотеками та конфігураціями в ізольовані контейнери. Це означає, що проєкт можна запустити в будь-якому середовищі, незалежно від його специфіки, що усуває можливі проблеми сумісності та полегшує розгортання.

Переваги використання Docker:

- **Легкість розгортання**: Оскільки Docker-файли містять повний стек додатка (бекенд, фронтенд, базу даних та інші сервіси), розгортання стає простим і швидким процесом.
- **Масштабованість**: Docker дозволяє легко збільшувати або зменшувати ресурси для різних компонентів проєкту. Це означає, що можна гнучко масштабувати фронтенд чи бекенд відповідно до навантаження.
- **Консистентність середовищ**: Docker гарантує, що додаток буде функціонувати однаково на всіх середовищах, від локального комп'ютера до хмарного серверу.
- **Швидке оновлення та відкат версій**: Використання Docker дозволяє легко оновлювати додаток, зберігаючи попередні версії для швидкого відкату в разі потреби.

Структура Docker

У проєкті створено декілька Docker-файлів:

- **Dockerfile для бекенду (Node.js/Express)**: У цьому файлі вказано інструкції для створення образу сервера з усіма залежностями Node.js, необхідними для

запуску REST API.

- **Dockerfile для фронтенду (Angular)**: Цей файл визначає конфігурацію для побудови та запуску Angular-додатка.
- **Docker Compose**: Використовується для керування багатокомпонентними застосунками. За допомогою docker-compose.yml можна одночасно запускати фронтенд, бекенд та базу даних PostgreSQL, що полегшує роботу з проєктом і робить його більш структурованим.

Docker забезпечує надійне середовище для розгортання, яке можна легко перенести на хмарні сервіси, такі як AWS, Google Cloud Platform, чи DigitalOcean, що дозволяє використовувати ресурси гнучко і економічно.

Підходи до забезпечення SEO-оптимізації

Для підвищення видимості онлайн-сервісу в пошукових системах використовуються сучасні методи SEO (Search Engine Optimization). Оскільки сайт створено за допомогою Angular, особливу увагу приділено питанням оптимізації односторінкових додатків (SPA), які традиційно важко індексуються пошуковими системами.

Основні підходи до SEO:

1. **Серверний рендеринг із використанням Angular Universal**: Angular Universal дозволяє рендерити сторінки додатка на сервері, а не на клієнтському боці, що забезпечує коректне відображення вмісту для пошукових систем. Це особливо важливо для SPA-додатків, де більшість вмісту динамічно завантажується за допомогою JavaScript.
2. **Оптимізація метаданих**: Правильне налаштування мета-тегів (title, description, keywords) для кожної сторінки допомагає пошуковим системам зрозуміти зміст сторінок і підвищує ймовірність їх правильного індексування. Для цього Angular використовує MetaService, що дозволяє програмно оновлювати метадані на кожній сторінці.

3. **Схема розмітки (Schema Markup)**: Використання розмітки Schema.org дозволяє додавати структуровані дані на сторінки, які полегшують пошуковим системам інтерпретацію контенту. Це забезпечує відображення інформації у вигляді розширених результатів, що збільшує видимість сервісу в пошукових результатах.
4. **Оптимізація швидкості завантаження**: Швидкість завантаження сайту є важливим фактором ранжування. Використання технік, таких як **ліниве завантаження зображень** (lazy loading), **зменшення розміру CSS та JavaScript** файлів та **кешування**, допомагає забезпечити високу продуктивність і швидкість роботи додатка, що позитивно впливає на SEO.
5. **Створення XML Sitemap**: XML-карта сайту містить інформацію про всі сторінки додатка, що дозволяє пошуковим системам легко знаходити і індексувати їх. Генерація sitemap.xml автоматично забезпечує оновлення карти при додаванні чи зміні сторінок.
6. **Кросплатформна оптимізація**: Оскільки пошукові системи також враховують зручність для мобільних користувачів, особливу увагу приділено адаптивності дизайну. Вебдодаток оптимізовано для коректного відображення на різних пристроях, забезпечуючи зручний інтерфейс як на мобільних пристроях, так і на комп'ютерах.
7. **Створення оптимізованих URL**: Використання чітких і зрозумілих URL-адрес допомагає пошуковим системам краще індексувати сторінки. Замість динамічних і довгих адрес, що генеруються автоматично, застосовується формат “людського читання”, що включає ключові слова, які описують вміст сторінки.
8. **Переваги SEO-оптимізації для проєкту**

Завдяки впровадженню описаних методів SEO, додаток має змогу досягти більш високих позицій у результатах пошуку, що підвищує його видимість серед потенційних користувачів і залучає нову аудиторію. Правильна SEO-

оптимізація не лише покращує ранжування, а й забезпечує кращий користувацький досвід за рахунок швидкого завантаження сторінок, зручної навігації та адаптивного дизайну, що позитивно впливає на тривалість перебування на сайті та зменшення показника відмов.я

2.2 Вимоги до програмного забезпечення

Для успішної реалізації проєкту онлайн-сервісу з корпоративного озеленення міст було визначено низку вимог до програмного забезпечення, які забезпечують його стабільну роботу, безпеку та зручність використання. Ці вимоги можна розподілити на функціональні та нефункціональні, кожна з яких має своє важливе значення для забезпечення відповідності сервісу очікуванням користувачів та бізнес-цілям.

Функціональні вимоги

Функціональні вимоги визначають конкретний функціонал і взаємодії користувача з додатком:

1. **Реєстрація та аутентифікація користувачів**: Система повинна забезпечувати можливість реєстрації нових користувачів, а також надійну автентифікацію та авторизацію. Для захисту особистих даних повинні використовуватися токени доступу, такі як JWT.
2. **Замовлення висадки дерев**: Користувачі мають змогу вибирати місце, тип дерева, дату та час посадки. Система повинна підтримувати взаємодію з інтерактивною картою, де користувач може обирати доступні місця для висадки.
3. **Статус замовлень та відстеження процесу**: Користувачі повинні мати можливість переглядати статус своїх замовлень у реальному часі, отримуючи актуальні дані щодо етапів обробки.
4. **Оплата**: Додаток повинен підтримувати інтеграцію з платіжними сервісами для безпечної та зручної оплати замовлень. Після завершення оплати користувач повинен отримати підтвердження транзакції.
5. **Адміністративна панель**: Адміністратори мають доступ до функціоналу управління замовленнями, перегляду статистики та звітності, а також можливості керування користувачами та контентом.

Нефункціональні вимоги

Нефункціональні вимоги забезпечують якісні характеристики програмного забезпечення, роблячи його продуктивним, зручним і надійним.

1. **Продуктивність**: Додаток повинен забезпечувати швидкий час відгуку для основних функцій, таких як завантаження карти, обробка замовлень та відображення статусу. Максимальний час очікування для користувача повинен бути не більше 2-3 секунд.
2. **Масштабованість**: Програмне забезпечення має підтримувати зростання навантаження та кількості користувачів. Для цього передбачено використання контейнеризації за допомогою Docker, що забезпечує легке масштабування системи на хмарних платформах.
3. **Безпека**: Захист даних користувачів є пріоритетом, тому вимагається захищений канал передачі даних (HTTPS), шифрування токенів доступу та надійна авторизація. Також передбачена захист від атак, таких як CSRF та XSS.
4. **Зручність користування (юзабіліті)**: Інтерфейс повинен бути зрозумілим і інтуїтивно доступним для всіх категорій користувачів. Важливо забезпечити простоту навігації, доступ до основних функцій за декілька кліків та адаптивність для різних пристроїв.
5. **Сумісність**: Додаток повинен коректно працювати на основних браузерах (Chrome, Firefox, Safari) та бути оптимізованим для мобільних пристроїв. Це гарантує, що користувачі зможуть зручно використовувати сервіс незалежно від платформи.
6. **Надійність та стійкість до збоїв**: Система повинна бути стійкою до збоїв, а в разі несправностей — швидко відновлювати роботу без втрати даних. Для цього передбачені регулярні резервні копії бази даних і моніторинг стану сервера.
7. **Можливість підтримки та оновлення**: Програмне забезпечення має бути

розроблено з урахуванням майбутніх оновлень та розширення функціоналу. Для цього використовується модульна архітектура, що дозволяє легко вносити зміни та адаптувати сервіс до нових вимог.

Інтеграційні вимоги

1. **Інтеграція з картографічними сервісами**: Додаток має підтримувати інтеграцію з Google Maps API для точного визначення місць висадки дерев.
2. **Платіжні сервіси**: Для підтримки онлайн-оплати передбачено інтеграцію з платіжними шлюзами, що забезпечують безпечну обробку фінансових транзакцій.
3. **Можливість API-інтеграцій**: Для майбутнього розширення функціоналу додатка, наприклад, інтеграції з іншими екологічними сервісами, додаток повинен мати добре документований REST API, який забезпечить гнучкість у розширенні можливостей.

2.3 Загальна структура розробленого програмного продукту

Для успішної реалізації онлайн-сервісу з корпоративного висаджування дерев важливою складовою є ефективно спроектована архітектура додатка. Враховуючи специфіку завдань і функціональність, яку має виконувати цей сервіс, було обрано клієнт-серверну архітектуру з розподілом на фронтенд і бекенд частини. Така структура забезпечує незалежність клієнтської частини від серверної, що підвищує гнучкість, масштабованість та зручність підтримки продукту.

Додаток побудовано за модульним підходом, де кожен компонент виконує певні завдання та взаємодіє з іншими модулями через чітко визначені інтерфейси. Клієнтська частина (фронтенд), реалізована за допомогою Angular, відповідає за взаємодію користувачів із додатком, відображення даних та підтримку інтуїтивного інтерфейсу. Серверна частина (бекенд), побудована на Node.js і Express, обробляє запити, виконує бізнес-логіку та взаємодіє з базою даних PostgreSQL.

REST API виступає ключовим інтерфейсом для зв’язку між фронтендом і бекендом, забезпечуючи ефективну взаємодію та обмін даними між компонентами. Використання окремих модулів для таких аспектів, як управління станом (Store модуль), обробка запитів (Services модуль) і відображення інтерфейсу (UI модуль), дозволяє зберігати чітку організацію та спрощує процес внесення змін і розширення функціоналу додатка.

Frontend частина

Frontend частина додатка реалізована за допомогою Angular та складається з трьох основних модулів: Services, Store, та UI. Кожен модуль відповідає за певну логіку роботи з інтерфейсом, обробку даних та взаємодію з сервером.

1. **Services модуль**

Services модуль включає підмодулі, які відповідають за взаємодію з API бекенду. Цей модуль дозволяє здійснювати CRUD-операції над різними

ресурсами, наприклад, замовленнями чи типами дерев. Ключовим підмодулем є ProductService, який реалізовано як сервіс для модифікації даних продуктів і зв'язку з основними endpoint'ами (рис. 2.3.1.).

Рис. 2.3.1. Frontend ProductService

Основні методи ProductService забезпечують наступні операції:

- **getProductById** та **getAllProducts** — отримання даних про продукти.
- **createProduct** — створення нового продукту через запит до бекенду.
- **updateProductVisibility** та **updateProductById** — оновлення даних продукту.
- **deleteProductById** — видалення продукту.

2. **Store модуль**

Store модуль забезпечує управління станом додатка, використовуючи схему стейт-менеджменту. Він виконує функцію клієнтської бази даних для зберігання всіх даних, що відображаються в компонентах. Основні складові цього модуля — ефекти та редюсери, що відповідають за обробку дій у стейті.

- **Effects** (наприклад, ProductEffects) обробляють асинхронні запити до API, зберігаючи отримані дані у стані додатка.
- **Reducers** (наприклад, ProductActionReducer) записують оновлені дані в стейт базу, що дозволяє компонувати нові дані на основі дій користувача.

3. **UI модуль**

UI модуль відповідає за основні інтерфейсні компоненти додатка, такі як кнопки, форми, модальні вікна. До прикладу, ModalService та SpinnerService відповідають за управління модальними вікнами та анімацією підвантаження даних відповідно.

Backend (API) частина

Backend частина реалізована за допомогою Node.js та Express і також структурована за модульним принципом, що забезпечує організованість та легкість доступу до функціоналу. Основні модулі: Services, Repository, та API.

1. **Services модуль**

Services модуль обробляє дані, які надходять від API, і взаємодіє з репозиторієм для доступу до бази даних. Наприклад, ProductService у бекенді забезпечує обробку запитів на створення, оновлення та видалення замовлень, а також передає дані до Repository модуля (рис. 2.3.2.).

```
export class ProductService {
  public getAllProducts(): Promise<IProductDto[]> {
    return productRepository.getAll();
  }
  public getById(id: number): Promise<IProductDto> {
    return productRepository.getById(id);
  }
  public createNewProduct(product: IProductDto): Promise<IProductDto> {···
  }

  public async updateProduct(···
  }

  public deleteProduct(id: number): Promise<number> {···
  }
}
```

Рис. 2.3.2. Backend ProductService

2. **Repository модуль**

Repository модуль є основою для запитів до бази даних. Він виконує побудову SQL-запитів та керує операціями над даними. Наприклад, ProductRepository використовує методи для створення, оновлення та видалення продуктів у базі даних (рис. 2.3.3.).

```
export class ProductRepository {
  public getById(id: number): Promise<IProductDto | null> {
    return productModule.findByPk(id, getProductParams());
  }

  public getAll(): Promise<Array<IProductDto>> {
    return productModule.findAll(getProductParams());
  }

  public createProduct(product: any): Promise<IProductDto> {···
  }

  public async updateById(···
  }

  public deleteById(id: number): Promise<number> {···
  }
}
```

Рис. 2.3.3. Backend ProductRepository

3. API модуль

API модуль забезпечує кінцеву точку взаємодії з вебдодатком (рис. 2.3.4.). Він обробляє запити від клієнтів, перевіряє їхню валідність та виконує відповідні CRUD-операції. Після обробки даних API повертає відповідь клієнту або повідомлення про помилку у випадку некоректних запитів. Додатково, тут реалізовано UploadImageService, що працює із хмарними сервісами для зберігання зображень (рис. 2.3.5.).

```
export const initProductApi = (apiRouter: Router): Router => {
  const productRouter = Router();
  const upload = multerUploadFile();

  apiRouter.use(rootApiPathEnum.Products, productRouter);

  productRouter.get(
    productApiPathEnum.All,
    verifyTokenMiddleware,
    authMiddleware([UserRolesEnum.Admin]),
    async (_req, res) => {
      try {
        let products = await productService.getAllProducts();

        res.status(checkIsFound(products)).json(products);
      } catch (error) {
        res.status(HttpCode.INTERNAL_SERVER_ERROR).json([]);
      }
    },
  );

  productRouter.get(···
  );

  productRouter.post(···
  );

  productRouter.patch(···
  );

  productRouter.delete(···
  );

  return productRouter;      You, 18 months ago • added logic for adding products
};
```

Рис. 2.3.4. Backend initProductApi

```
productRouter.post(
  userApiPathEnum.ROOT,
  verifyTokenMiddleware,
  authMiddleware([UserRolesEnum.Admin]),
  upload.single('image'),
  async (_req, res) => {
    try {
      const image = await uploadImageService.uploadImage(_req.file);
      const product = await productService.createNewProduct({
        ...JSON.parse(_req?.body?.data),
        image: image,
      });

      res.status(HttpCode.OK).json(product);
    } catch (err) {
      const error = err?.errors?.[0];
      console.log(err);      You, 17 months ago • Added migration

      await uploadImageService.deleteUploadedImage(
        error?.instance?.image || '',
      );
      res.status(HttpCode.BAD_REQUEST).json({ message: error?.message });
    }
  },
);
```

Рис. 2.3.5. Backend UploadImageService виклик

Загалом, структура додатка дозволяє легко керувати логікою як на клієнтській, так і на серверній частинах. Модульний підхід сприяє розподіленню відповідальності між різними компонентами та забезпечує легкість розширення функціоналу, що робить додаток стійким до змін та зручним у підтримці.

2.4 Реалізація інтерейсу корстувача

Інтерфейс програмного продукту побудований з урахуванням усіх вимог щодо зручності та інтуїтивної взаємодії, надаючи користувачам усі необхідні функціональні можливості для роботи з додатком. Основні функціональні можливості доступні через головну сторінку, сторінки профілю, замовлень і карту для вибору місця посадки.

Користувачі можуть розпочати роботу з додатком, перейшовши на **головну сторінку**, де зібрано ключову інформацію про проєкт і його екологічні ініціативи (рис. 2.4.1.). На головній сторінці доступні також навігаційні елементи, які дозволяють швидко перейти до основних функціональних розділів, таких як вибір місця посадки дерев за допомогою інтерактивної карти (рис. 2.4.2).

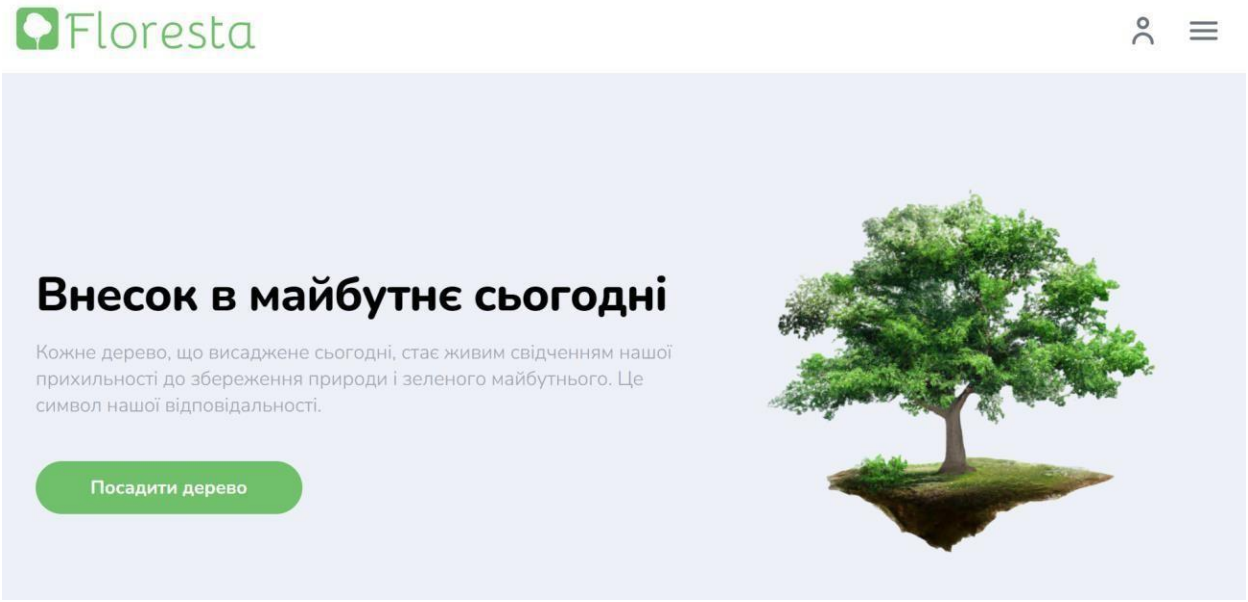

Рис. 2.4.1 Головна сторінка проекту

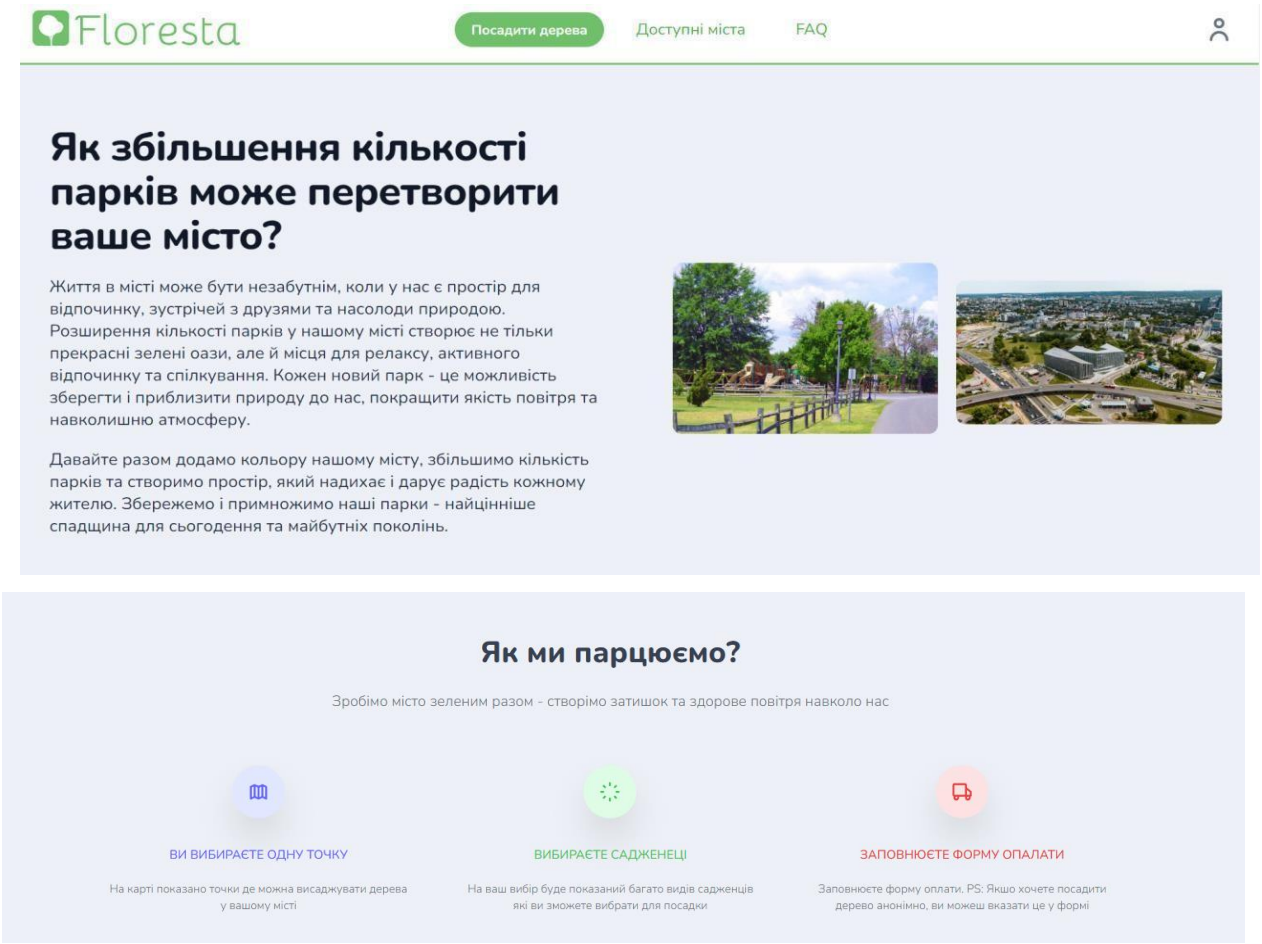

Рис. 2.4.2 Головна сторінка проекту секція ознайомлення

Авторизація в додатку

Користувачі мають можливість увійти в систему через сторінку авторизації, де передбачено поля для введення електронної пошти та пароля (рис. 2.4.3). Після введення даних відбувається верифікація облікового запису, і користувач отримує унікальний токен доступу, що надсилається на його пристрій.

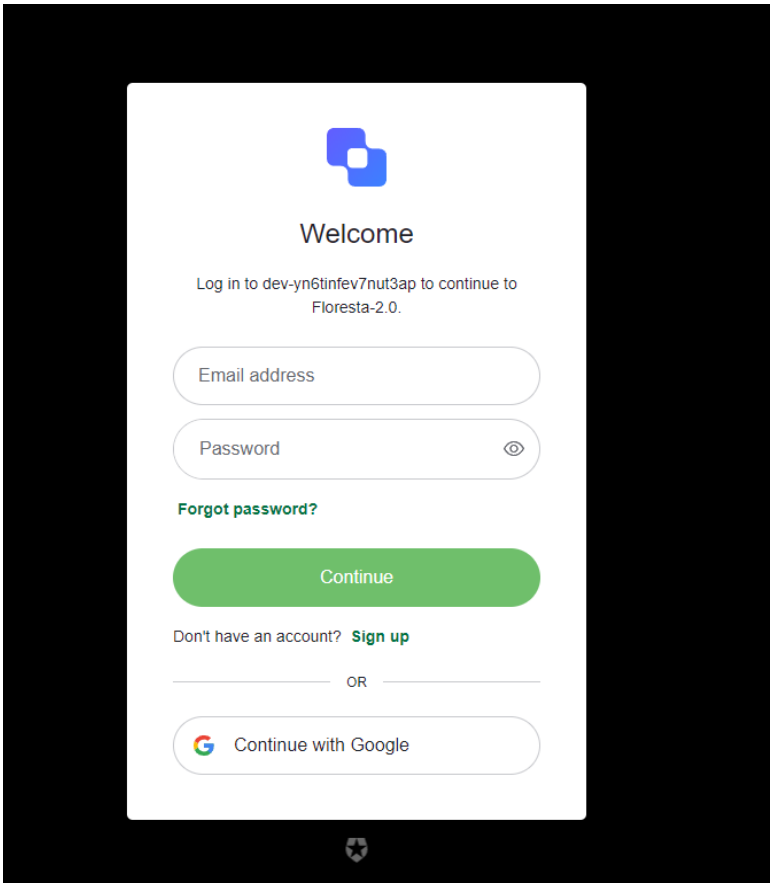

Рис. 2.4.3 Сторінка авторзації в додаток

Інтерактивна карта для вибору місця посадки дерев

На сторінці з інтерактивною картою користувачі можуть обирати місця для посадки дерев (рис. 2.4.4). Інтеграція Google Maps API дозволяє користувачам знаходити доступні для висадки ділянки, переглядати їхні координати та зручність розташування, що робить процес вибору інтуїтивним і зручним.

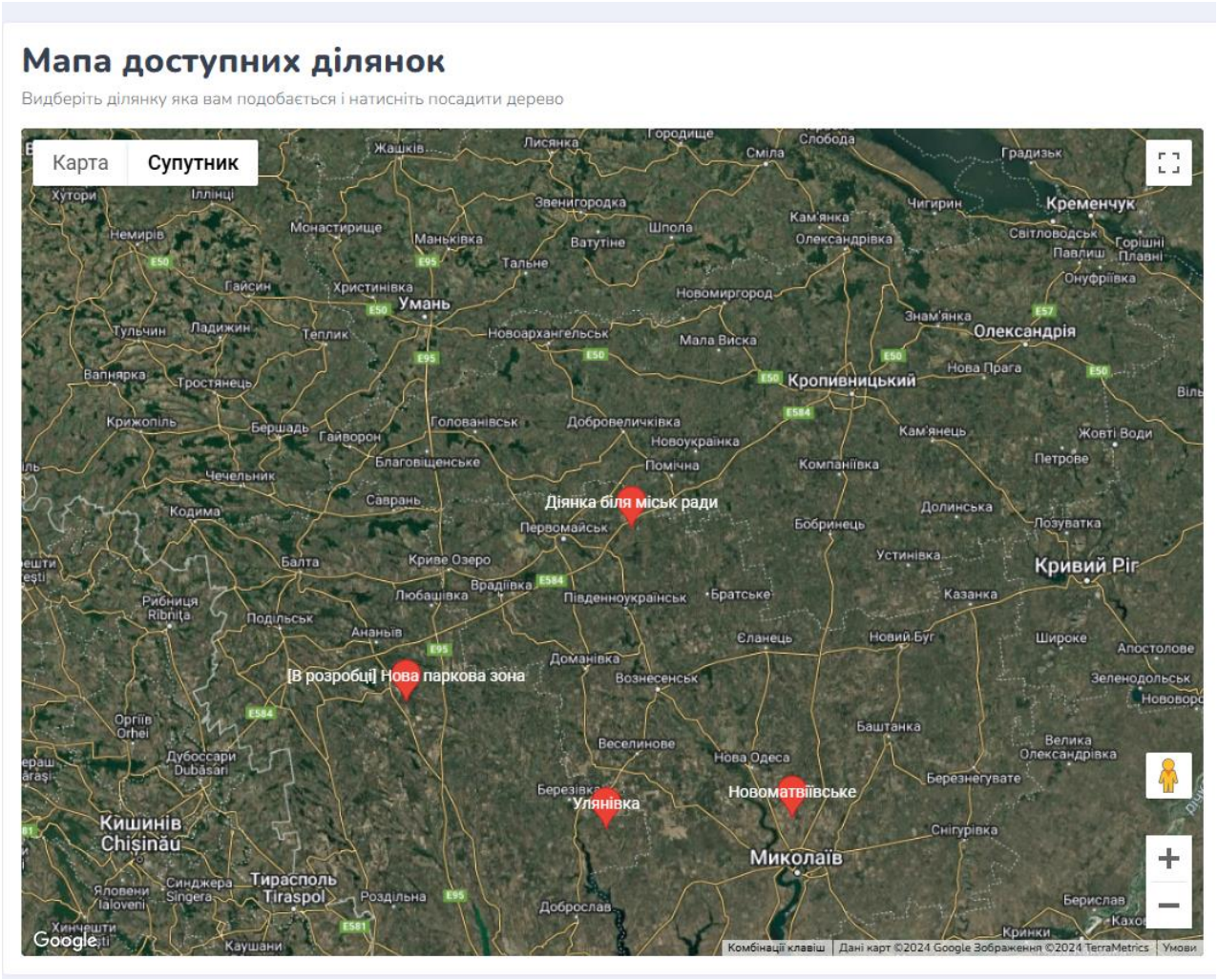

Рис. 2.4.4 Мапа з можливістю посадки дерев

Каталог дерев і опція "Власноруч"

Після вибору місця посадки користувачам доступний **каталог видів дерев**, де можна переглядати інформацію про доступні для замовлення дерева (рис. 2.4.5). Додаток пропонує вибір серед кількох видів дерев, а також дає можливість обрати опцію “Власноруч” (рис. 2.4.6)., що дозволяє користувачеві самостійно висадити дерево. Вибір типу дерева та підтвердження замовлення забезпечують максимальну персоналізацію процесу.

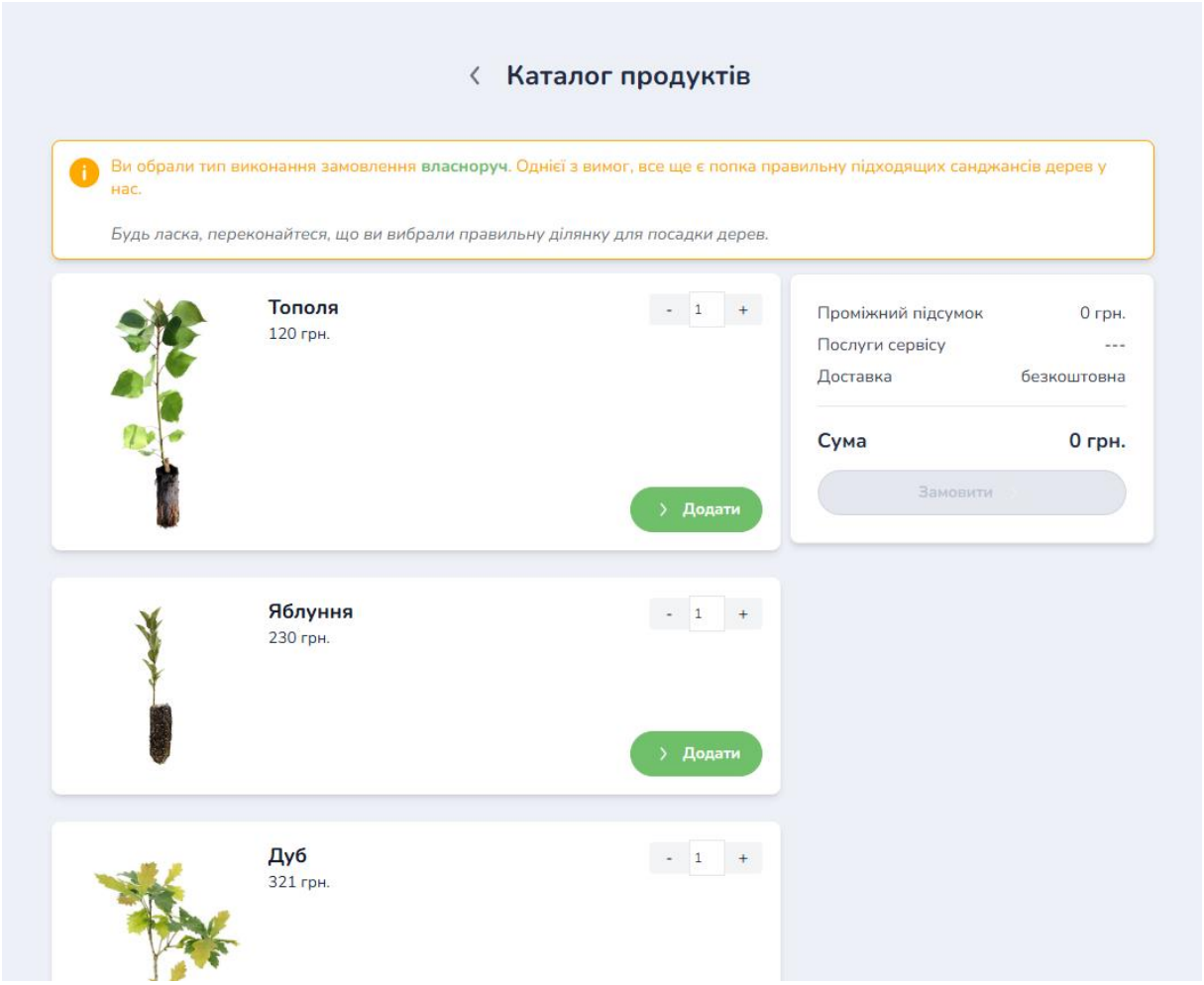

Рис. 2.4.5 Каталог продуктів

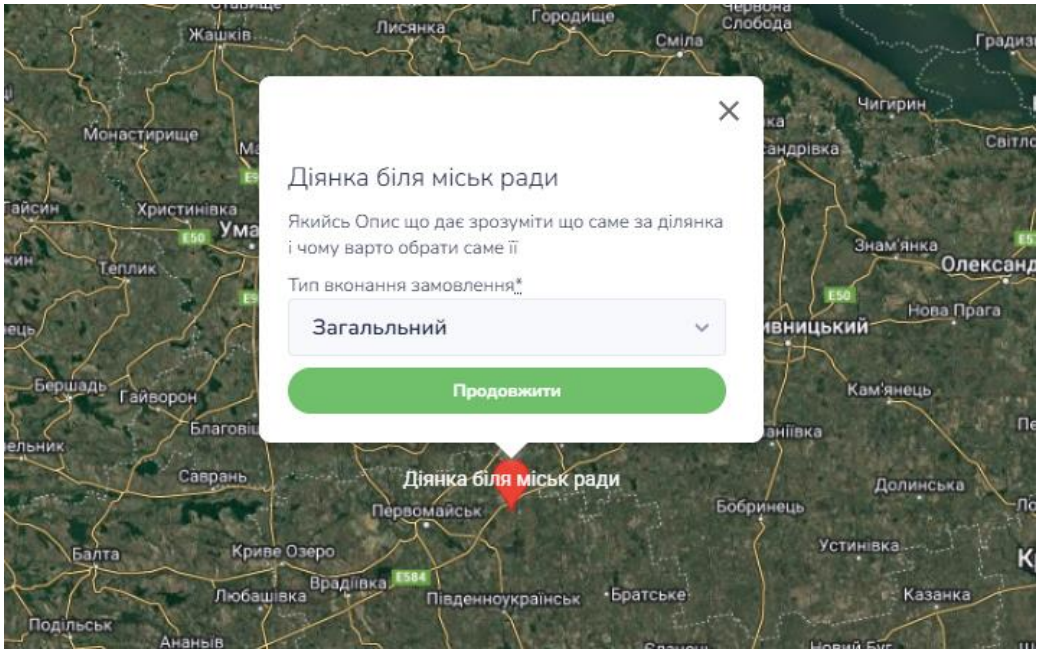

Рис. 2.4.6 Вибір типу виконання замовлення

Оформлення замовлення та форма оплати

Після вибору дерева та підтвердження локації користувач може перейти до оформлення замовлення. У випадку, якщо жодне дерево не обрано, кнопка "Замовити" залишається неактивною, що дозволяє уникнути помилок (рис. 2.4.7). Після вибору дерева користувач натискає кнопку "Замовити" і переходить до **форми оплати** (рис. 2.4.8). Платіжна інтеграція забезпечує безпечну обробку платежів, а після успішного завершення транзакції користувач потрапляє на сторінку підтвердження замовлення з подякою (рис. 2.4.9).

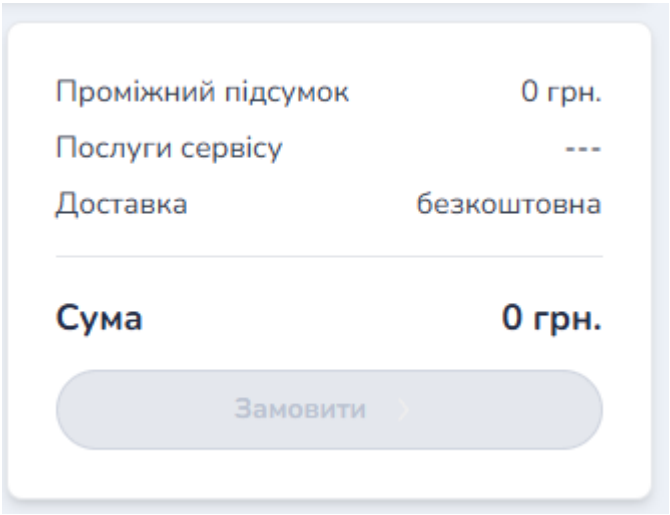

Рис. 2.4.7 Блок з кнопкою замовити

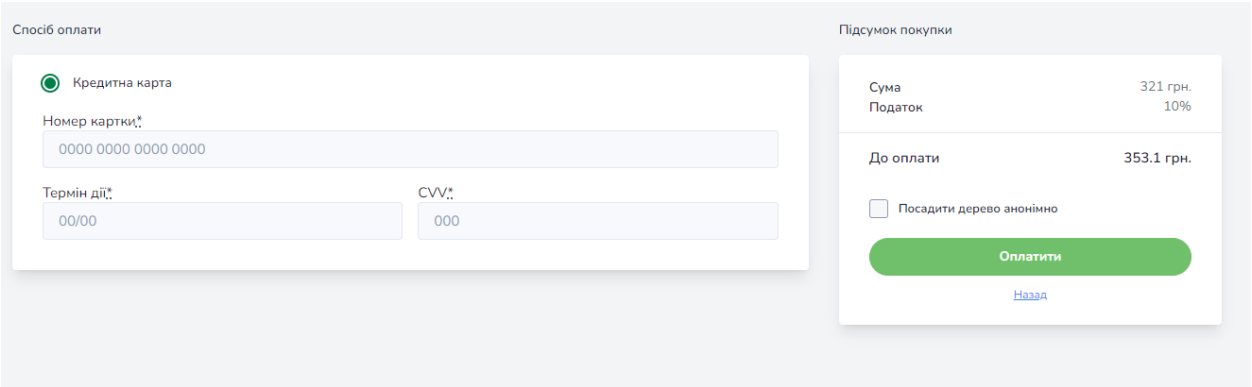

Рис. 2.4.8. Форма оплати

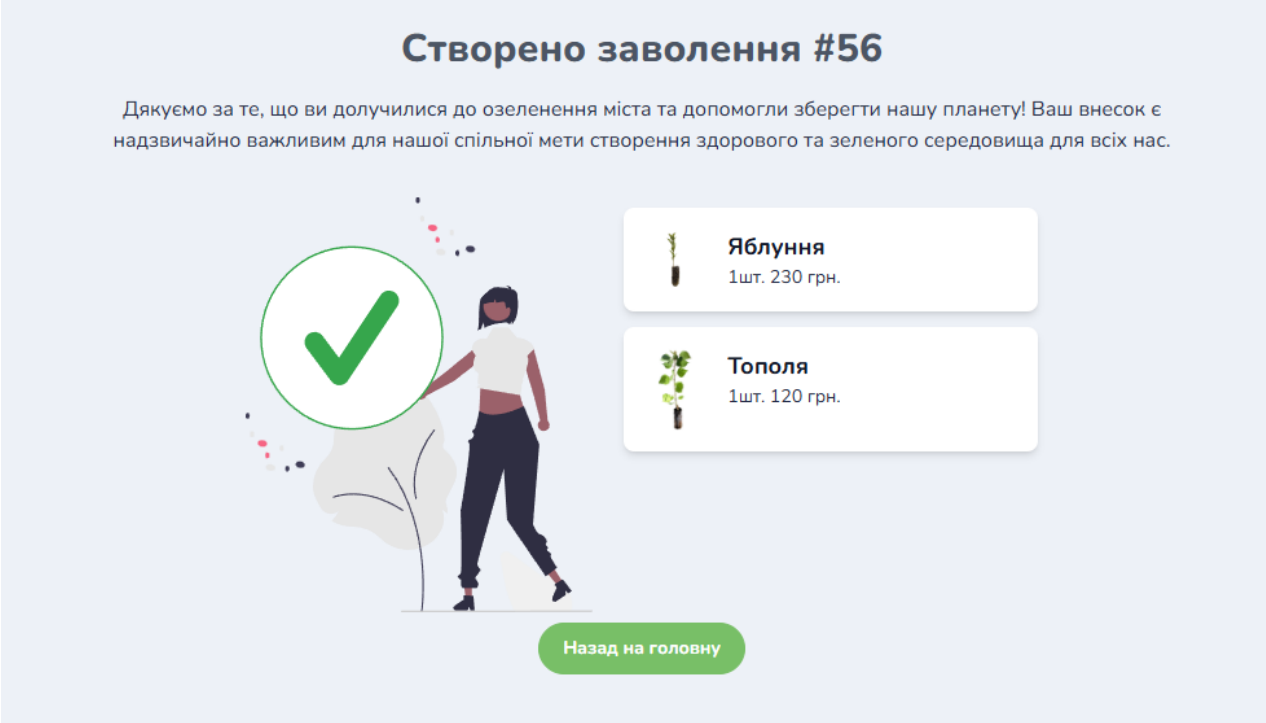

Рис. 2.4.9 Сторінка подяки

Сторінка "Мій профіль" і замовлення

Користувач має доступ до власної сторінки "Мій профіль", де можна змінювати особисту інформацію та переглядати історію замовлень (рис. 2.4.10). Зі сторінки "Мій профіль" користувач може перейти до розділу "Мої замовлення" для перегляду статусу всіх замовлень (рис. 2.4.11). Тут доступна інформація про кожне замовлення та його поточний статус.

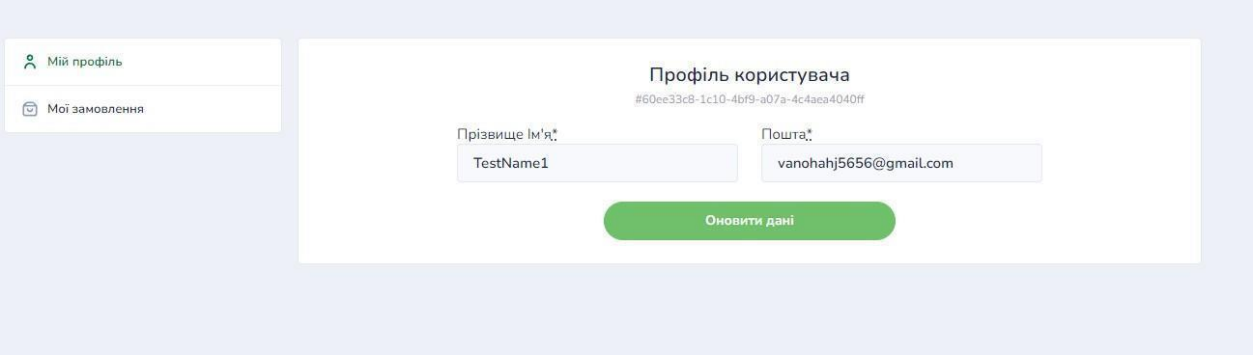

Рис. 2.4.10 Сторінку Мій профіль

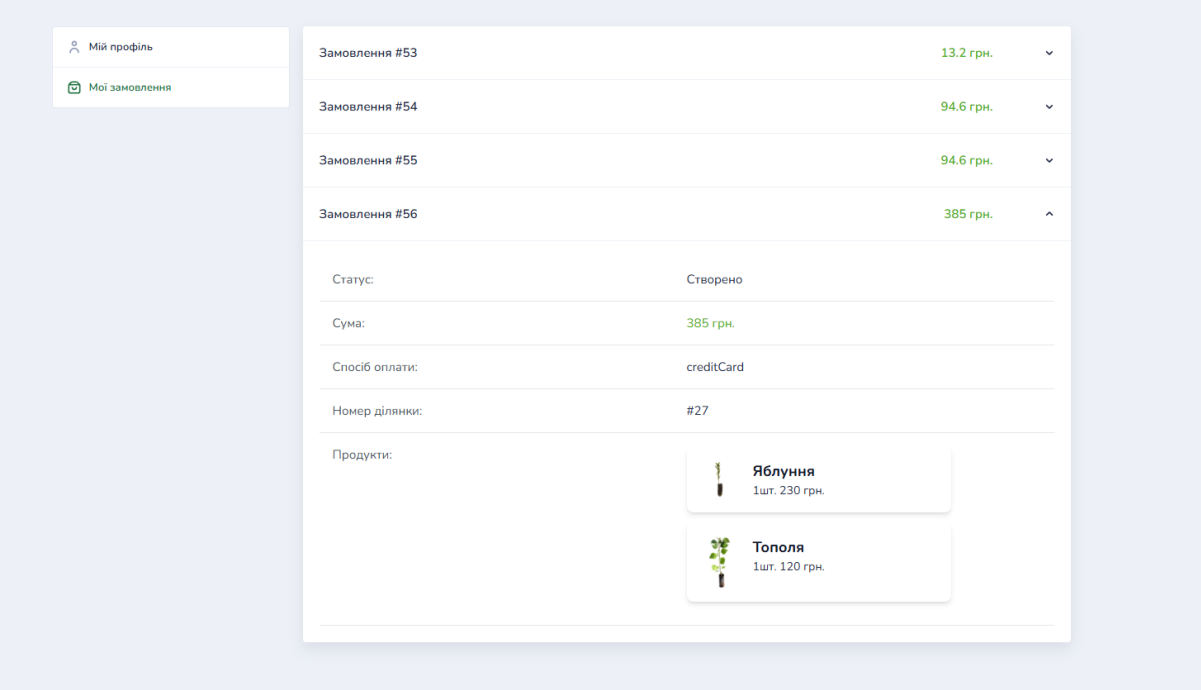

Рис. 2.4.11. Сторінку Мої замовлення

Адмінпанель

Для адміністраторів передбачена **адмінпанель**, доступ до якої надається через меню (рис. 2.4.12). Адмінпанель дозволяє керувати мітками, продуктами, користувачами та замовленнями, забезпечуючи можливості редагування, приховування та видалення продуктів (рис. 2.4.13, 2.4.14). Кожен продукт має свою панель управління з функціями редагування, приховування або видалення. Натискання кнопки "Приховати" приховує продукт від користувачів (рис. 2.4.15), а кнопка "Редагувати" відкриває модальне вікно редагування продукту (рис. 2.4.16).

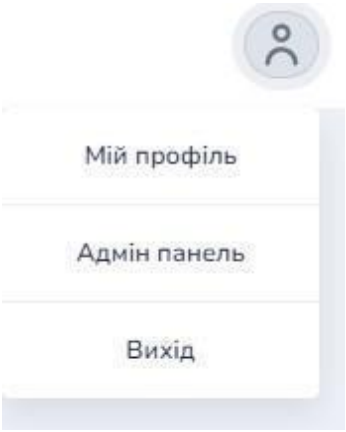

Рис. 2.4.12. Меню

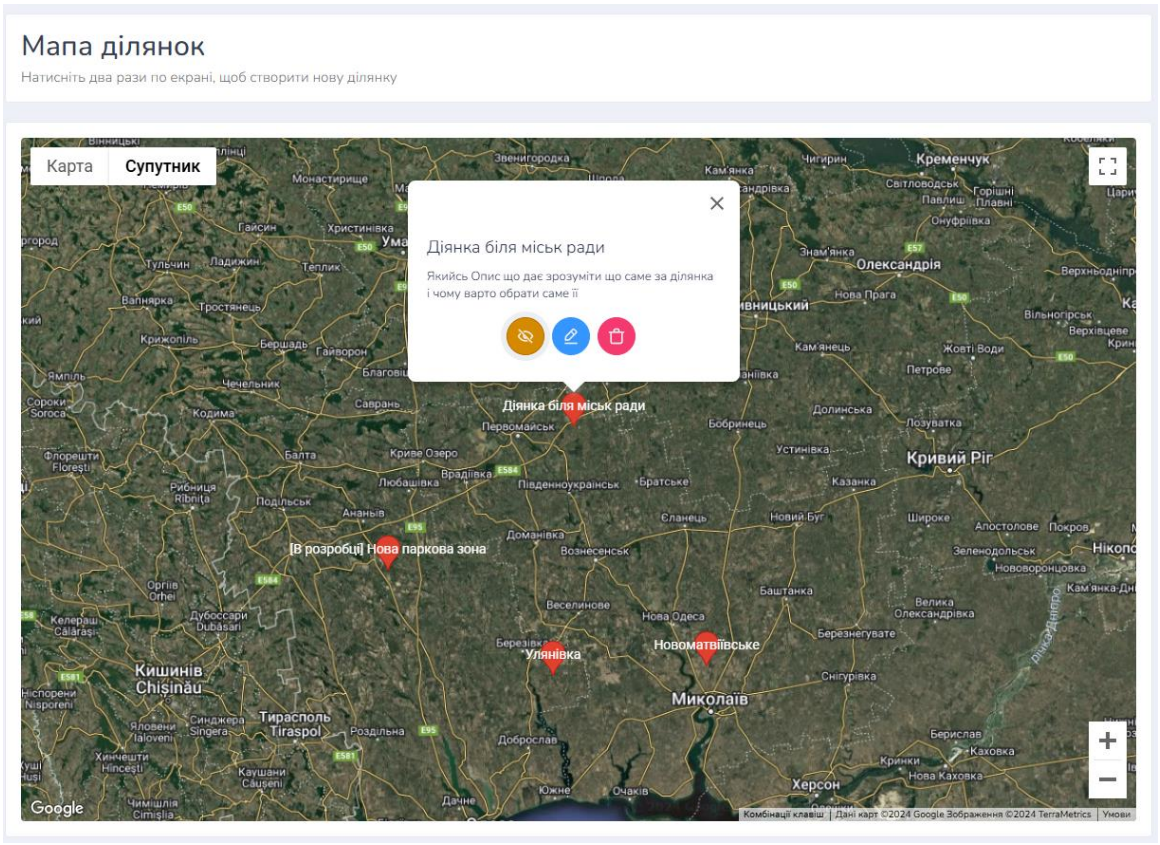

Рис. 2.4.13. Сторінка адміністрування міток

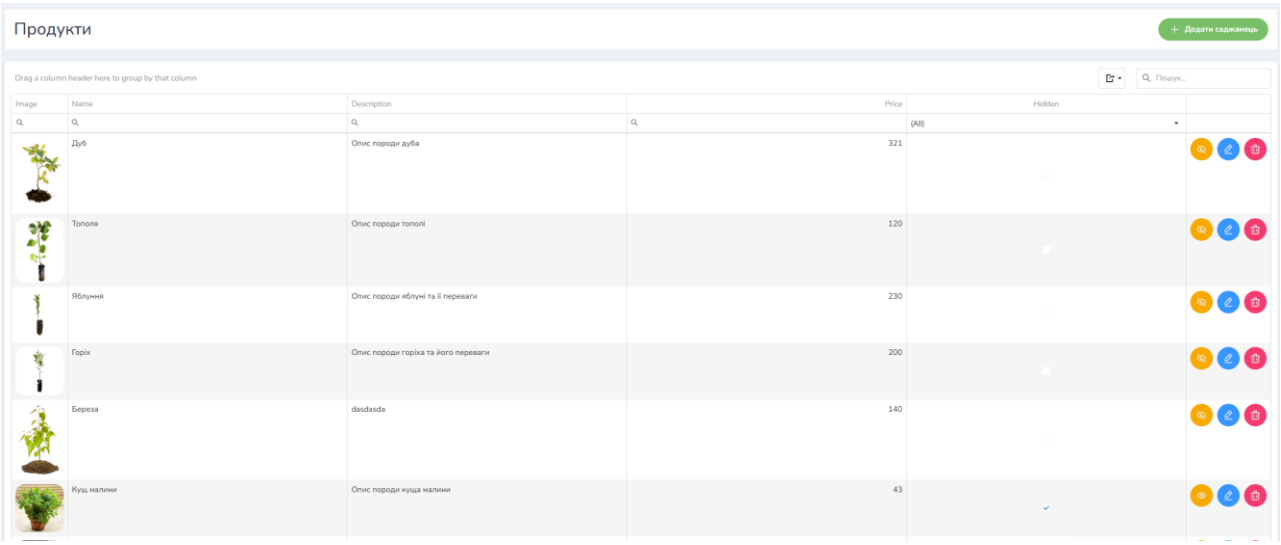

Рис. 2.4.14. Сторінка адміністрування продуктів

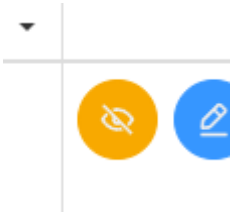

Рис. 2.4.15. Кнопки взаємодії приховати продукт

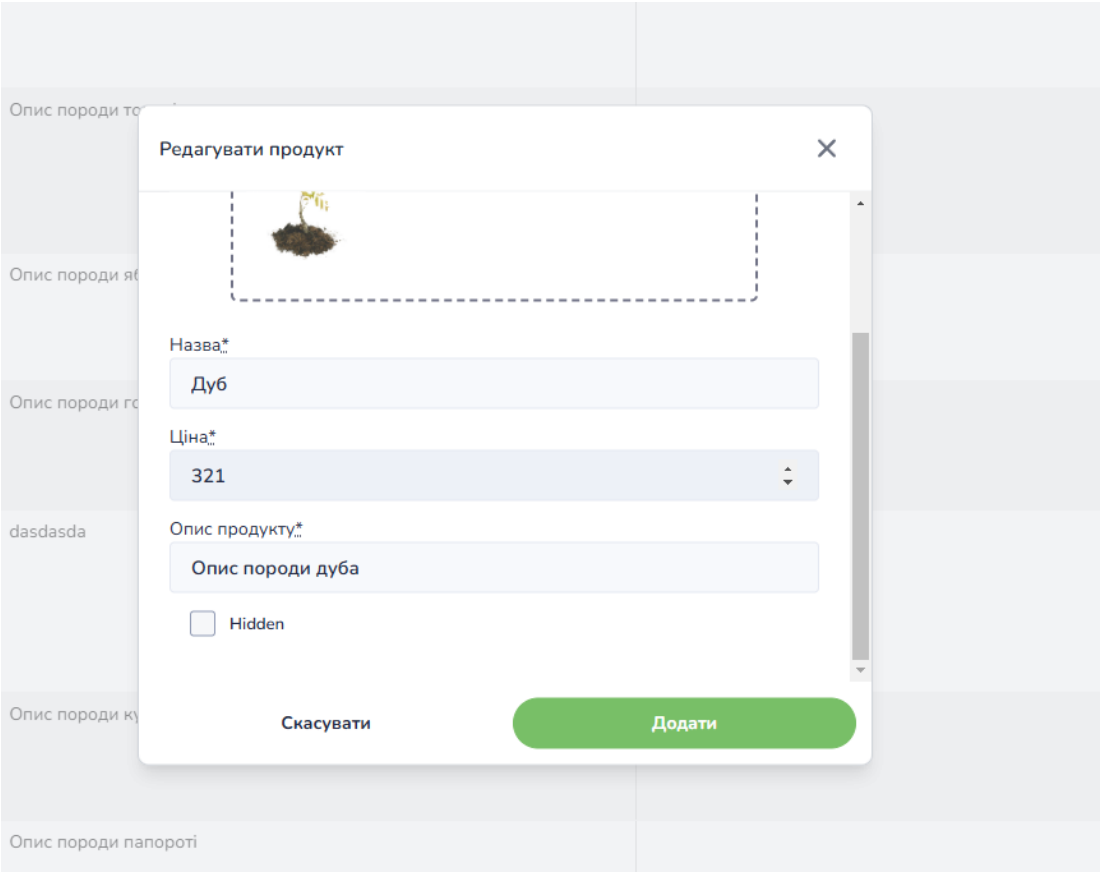

Рис. 2.4.16. Модальне вікно редагування продукту

При на тисканні на кнопку видалення адміністратор побачить модальне вікно підтвердження видалення (Рис. 2.4.17.).

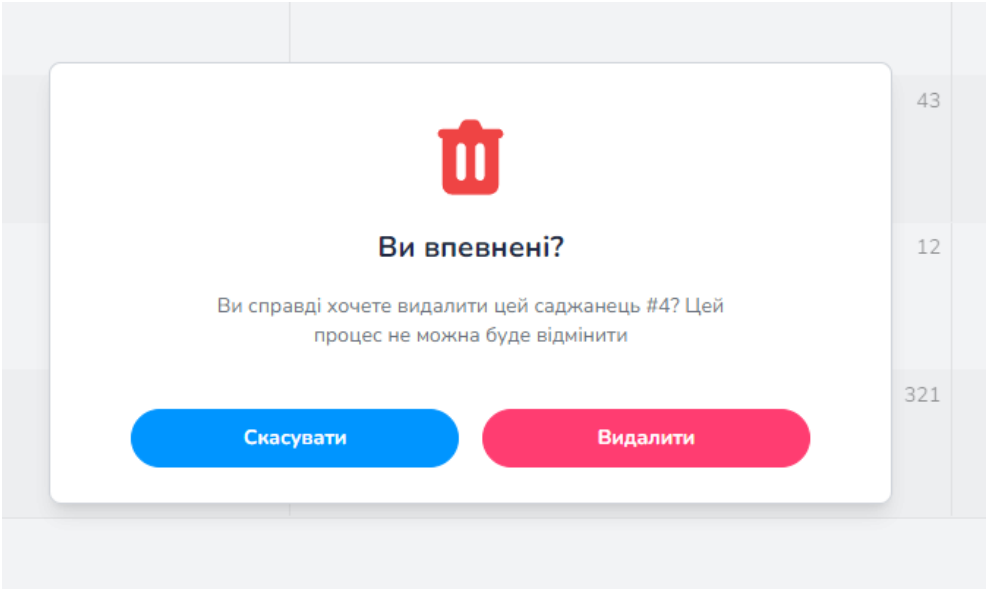

Рис. 2.4.17. Модальне підтвердження видалення

У адмінпанелі також реалізовано **діалогове вікно редагування замовлень**, що дозволяє адміністратору змінювати склад і параметри замовлення (рис. 2.4.18).

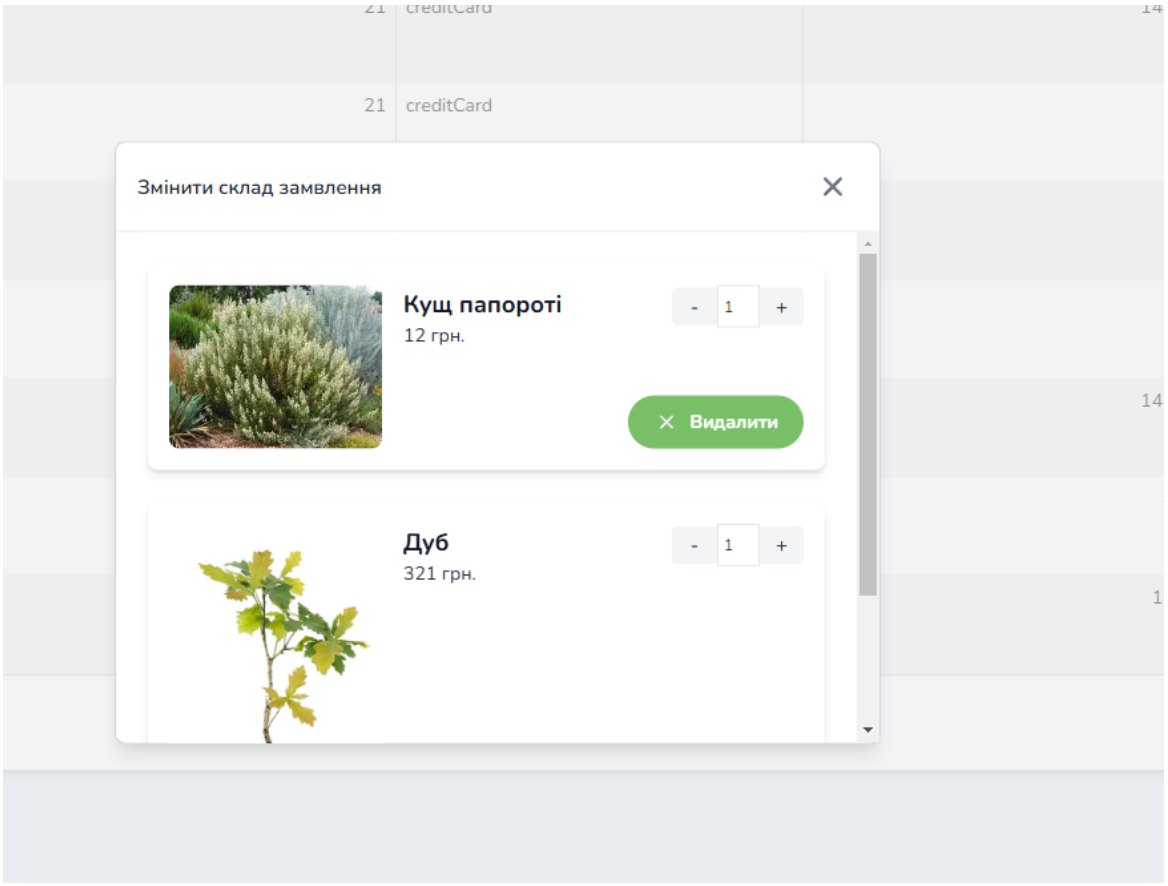

Рис. 2.4.18. Модальне вікно редагування складу замовлення

Таким чином, інтерфейс системи спроектований з урахуванням зручності для кінцевого користувача, забезпечуючи всі функціональні можливості, необхідні для оформлення, оплати та контролю замовлень, а також ефективне управління для адміністратора.

2.6 Методика та результати випробувань

У процесі розробки та впровадження онлайн-сервісу для корпоративного озеленення було проведено серію випробувань, щоб оцінити функціональність, продуктивність та зручність користування системою. Основні випробування включали: тестування функціональності, навантажувальне тестування, тестування користувацького інтерфейсу, перевірку безпеки та зручності. Окрему увагу було приділено перевірці коректності роботи REST API, інтеграції з Google Maps API та платіжними сервісами.

Тестування функціональності

Функціональне тестування проводилося для перевірки коректності реалізації основних можливостей сервісу. Серед ключових функцій перевірялися такі аспекти:

1. **Реєстрація та аутентифікація користувачів** — перевірка реєстрації нових користувачів, відновлення пароля, входу та виходу з системи.
2. **Замовлення висадки дерев** — оцінювалися всі етапи процесу: вибір локації на карті, вибір типу дерева, підтвердження замовлення та його оплата.
3. **Статус замовлень** — тестувалися оновлення статусів замовлень у реальному часі для забезпечення актуальності інформації.
4. **Адміністративний інтерфейс** — здійснювалася перевірка доступу адміністраторів до функцій редагування, видалення замовлень та взаємодії з користувачами.

Результати тестів свідчать про стабільність і коректність роботи основних функцій: у 98% випадків функції виконувалися відповідно до технічних вимог.

Навантажувальне тестування

Навантажувальне тестування виконувалося для визначення продуктивності системи під час значного навантаження. Було змодельовано ситуацію, коли до сервісу одночасно зверталися 500, 1000 та 2000 користувачів. Навантаження створювалося через багаторазові запити на сервер, що включали операції реєстрації, замовлення та перегляд статусів.

Висновки показали, що сервіс здатен ефективно обробляти до 1500 одночасних запитів без суттєвого збільшення часу відгуку. При навантаженні понад 2000 запитів спостерігалося певне зниження швидкості обробки, що вказує на можливість оптимізації для масштабування.

Тестування користувацького інтерфейсу

Для забезпечення зручності користування проводилося тестування інтерфейсу за участю потенційних користувачів з різним рівнем технічної підготовки. Учасники тестування отримували завдання, серед яких були реєстрація, вибір локації, замовлення висадки та перевірка статусу замовлення.

Результати тестування вказали на високу інтуїтивність інтерфейсу: 85% користувачів змогли виконати всі завдання без додаткових пояснень, а середній час на виконання кожного завдання був оптимальним. Деякі користувачі зазначили, що поліпшення навігації могло б пришвидшити доступ до окремих функцій, що було враховано під час подальшої розробки.

Тестування безпеки

Безпека сервісу оцінювалася шляхом перевірки захищеності даних користувачів. Було впроваджено кілька методів, зокрема перевірка на стійкість до атак типу SQL Injection, XSS, CSRF. Особливу увагу приділили тестуванню аутентифікації за допомогою JWT-токенів та безпечній передачі даних через HTTPS.

Під час тестування не було виявлено критичних вразливостей. Усі запити та дані передавалися через захищені канали, що підтвердило безпечність доступу до персональної інформації.

Результати та висновки

Проведені випробування підтвердили функціональність та стабільність роботи онлайн-сервісу для корпоративного озеленення. Система успішно пройшла тестування за всіма критеріями: забезпечення функціональності, продуктивності, зручності та безпеки. Отримані результати свідчать про можливість масштабування продукту та подальше його впровадження для підтримки екологічних ініціатив у міському середовищі.

ЗАГАЛЬНІ ВИСНОВКИ ДО РОБОТИ

Дипломна робота присвячена розробці онлайн-сервісу для підтримки корпоративного озеленення, який забезпечує ефективну організацію процесу висадки дерев у міських умовах. Метою проєкту було створення інструменту, що сприятиме об’єднанню зусиль бізнесу, громадських організацій та локальних громад для поліпшення міського середовища.

В ході розробки було проведено аналіз сучасних тенденцій в екологічній сфері та озелененні, що дозволило визначити ключові потреби користувачів. На основі цього аналізу було побудовано чітку структуру сервісу з функціями для різних категорій користувачів: бізнесів, що бажають підтримати екологічні ініціативи, благодійних організацій, які координують озеленення, та місцевих жителів, які можуть долучатися до цих заходів.

Програмне забезпечення побудовано на основі клієнт-серверної архітектури, що забезпечує гнучкість, масштабованість і простоту у підтримці. На серверній стороні було використано платформу Node.js з фреймворком Express для реалізації REST API, який обробляє запити та забезпечує доступ до даних. На клієнтській стороні застосовано Angular, що дозволило створити інтерактивний інтерфейс, адаптований до різних пристроїв.

Проєкт також використовує сучасні рішення для зручної інтеграції та масштабування. Зокрема, завдяки Docker контейнери забезпечують стабільне та легке розгортання системи на хмарних платформах, а база даних PostgreSQL відповідає за надійне зберігання інформації. Важливим компонентом є інтеграція Google Maps API, що дозволяє користувачам легко обирати місця для висадки дерев на інтерактивній карті, додаючи елемент інтуїтивності в користування.

Проведене тестування підтвердило надійність та стабільність системи. Функціональні випробування засвідчили коректність роботи всіх основних можливостей, а навантажувальні тести показали, що система здатна обробляти

велику кількість одночасних запитів без значного погіршення продуктивності. Тестування безпеки підтвердило стійкість системи до різних видів атак, що гарантує захист персональних даних користувачів.

Отже, розроблений онлайн-сервіс пропонує зручний інструмент для організації та моніторингу корпоративного озеленення, що сприяє покращенню екологічної ситуації у містах, зменшенню викидів вуглекислого газу та покращенню якості повітря. Завдяки гнучкості архітектури та інтеграції з популярними API сервіс має потенціал для подальшого розвитку і масштабування, що відкриває нові можливості для підтримки екологічних ініціатив.

ВИКОРИСТАНА ЛІТЕРАТУРА

1. Браун Е. Вивчаємо javascript: керівництво для створення сучасних веб-сайтів. 3-тє вид. 2020. 368 с.
2. Керні Б. Програмування TypeScript: масштабування ваших програм JavaScript. 7-ме вид. Orelly, 2020. 321 с.
3. Річардсон Л. Веб-API RESTful: служби для світу, що змінюється. O'Reilly Media, 2013. 406 с.
4. Стоян С. JavaScript. Шаблони. 2-ге вид. SNL, 2018. 272 с.
5. Express - Node.js web application framework. *Express - Node.js web application framework*. URL: https://expressjs.com/ (дата звернення: 11.05.2023).
6. Index | node.js v15.14.0 документація. *Node.js*. URL: https://nodejs.org/docs/latest-v15.x/api/ (дата звернення: 10.05.2023).
7. JavaScript with syntax for types. *TypeScript: JavaScript With Syntax For Types*. URL: https://www.typescriptlang.org/ (дата звернення: 11.05.2023).
8. MDN web документація. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/ (дата звернення: 01.06.2023).
9. Tailwind CSS - rapidly build modern websites without ever leaving your HTML. *Tailwind CSS - Rapidly build modern websites without ever leaving your HTML*. URL: https://tailwindcss.com/ (дата звернення: 20.05.2023).

ДОДАТКИ

Frontend:

frontend/angular.json

```
{
  "$schema":
"./node_modules/@angular/cli/lib/config/schema.json",
  "version": 1,
  "newProjectRoot": "projects",
  "projects": {
    "frontend": {
      "projectType": "application",
      "schematics": {
        "@schematics/angular:component": {
          "style": "scss"
        }
      },
      "root": "",
      "sourceRoot": "src",
      "prefix": "app",
      "architect": {
        "build": {
          "builder": "@angular-devkit/build-
angular:browser",
          "options": {
            "outputPath": "dist/frontend",
            "index": "src/index.html",
            "main": "src/main.ts",
            "polyfills": [
              "zone.js"
            ],
            "tsConfig": "tsconfig.app.json",
            "inlineStyleLanguage": "scss",
            "assets": [
              "src/favicon.ico",
              "src/assets"
            ],
            "styles": [

"node_modules/devextreme/dist/css/dx.light.css",

"node_modules/@nebular/theme/styles/prebuilt/default.
css",
              "src/styles.scss"
            ],
            "scripts": []
          },
          "configurations": {
            "production": {
              "budgets": [
                {
                  "type": "initial",
                  "maximumWarning": "5mb",
                  "maximumError": "15mb"
                },
                {
                  "type": "anyComponentStyle",
                  "maximumWarning": "8kb",
                  "maximumError": "16kb"
                }
              ],
              "outputHashing": "all"
            },
            "development": {
              "buildOptimizer": false,
              "optimization": false,
              "vendorChunk": true,
              "extractLicenses": false,
              "sourceMap": true,
              "namedChunks": true
            }
          },
          "defaultConfiguration": "production"
        },
        "serve": {
          "builder": "@angular-devkit/build-
angular:dev-server",
          "configurations": {
            "production": {
              "browserTarget":
"frontend:build:production"
            },
            "development": {
              "browserTarget":
"frontend:build:development"
            }
          },
          "defaultConfiguration": "development"
        },
        "extract-i18n": {
          "builder": "@angular-devkit/build-
angular:extract-i18n",
          "options": {
            "browserTarget": "frontend:build"
          }
        },
        "test": {
          "builder": "@angular-devkit/build-
angular:karma",
          "options": {
            "polyfills": [
              "zone.js",
              "zone.js/testing"
```

```
            ],
            "tsConfig": "tsconfig.spec.json",
            "inlineStyleLanguage": "scss",
            "assets": [
              "src/favicon.ico",
              "src/assets"
            ],
            "styles": [
              "src/styles.scss"
            ],
            "scripts": []
          }
        }
      }
    }
  },
  "cli": {
    "analytics": "088d84d1-02cb-4403-975a-
be625ff2a8d2"
  }
}
```

frontend/Dockerfile

```
# Use the official Node.js image as the base image
FROM node:16

# Set the working directory
WORKDIR /app

# Copy package.json and package-lock.json to the
working directory
COPY package*.json ./

# Install dependencies
RUN npm install

# Copy the rest of the application code to the
working directory
COPY . .

# Expose the port the app runs on
EXPOSE 4200

# Command to run the application
CMD ["npm", "start"]
```

frontend/package.json

```
{
  "name": "frontend",
  "version": "0.0.0",
  "scripts": {
    "ng": "ng",
    "start": "ng serve",
    "build": "ng build",
    "watch": "ng build --watch --configuration
development",
    "test": "ng test"
  },
  "private": true,
  "dependencies": {
    "@angular/animations": "^15.1.0",
    "@angular/cdk": "^15.0.0",
    "@angular/common": "^15.1.0",
    "@angular/compiler": "^15.1.0",
    "@angular/core": "^15.1.0",
    "@angular/forms": "^15.1.0",
    "@angular/google-maps": "^15.2.8",
    "@angular/platform-browser": "^15.1.0",
    "@angular/platform-browser-dynamic": "^15.1.0",
    "@angular/router": "^15.1.0",
    "@auth0/auth0-angular": "^2.1.0",
    "@nebular/eva-icons": "^11.0.0",
    "@nebular/theme": "^11.0.0",
    "@ngrx/effects": "^15.4.0",
    "@ngrx/store": "^15.4.0",
    "@ngrx/store-devtools": "^15.4.0",
    "@types/lodash": "^4.14.194",
    "angular-cc-library": "^3.1.0",
    "devextreme": "22.2.6",
    "devextreme-angular": "22.2.6",
    "eva-icons": "^1.1.3",
    "jspdf": "^2.5.1",
    "lodash": "^4.17.21",
    "ngx-dropzone": "^3.1.0",
    "rxjs": "~7.8.0",
    "tslib": "^2.3.0",
    "zone.js": "~0.12.0"
  },
  "devDependencies": {
    "@angular-devkit/build-angular": "^15.1.2",
    "@angular/cli": "~15.1.2",
    "@angular/compiler-cli": "^15.1.0",
    "@schematics/angular": "~15.1.2",
    "@types/jasmine": "~4.3.0",
    "@types/jspdf": "^2.0.0",
    "autoprefixer": "^10.4.14",
    "devextreme-cli": "latest",
    "devextreme-themebuilder": "^22.2.4",
    "jasmine-core": "~4.5.0",
    "karma": "~6.4.0",
    "karma-chrome-launcher": "~3.1.0",
    "karma-coverage": "~2.2.0",
    "karma-jasmine": "~5.1.0",
    "karma-jasmine-html-reporter": "~2.0.0",
    "postcss": "^8.4.21",
    "prettier": "2.8.7",
    "tailwindcss": "^3.3.1",
```

```
    "typescript": "~4.9.4"
  }
}
```

frontend/src/app/app.component.ts

```
import { ChangeDetectorRef, Component, OnDestroy,
OnInit } from '@angular/core';
import { combineLatest, Subject, takeUntil } from
'rxjs';
import {
  AuthWrapperService,
  LocalStorageService,
  SpinnerService,
  UserService,
} from '../services';
import { Store } from '@ngrx/store';
import { CartActions, ProfileActions } from
'../store/actions';
import { ProfileSelectors } from
'../store/selectors';
import { isEmpty } from 'lodash';
import { AUTH_TOKEN, USER_PROFILE } from
'../common/local-storage-keys';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.scss'],
})
export class AppComponent implements OnDestroy,
OnInit {
  isShowLoadingSpinner = false;

  private readonly unsubscribe$ = new Subject();
  constructor(
    private readonly authWrapperService:
AuthWrapperService,
    private readonly store: Store<any>,
    private readonly userService: UserService,
    private readonly localStorageService:
LocalStorageService,
    private readonly cdr: ChangeDetectorRef,
    private readonly spinnerService: SpinnerService,
  ) {}

  ngOnInit() {
    combineLatest([
      this.store.select(ProfileSelectors.getProfile),

this.authWrapperService.authService.idTokenClaims$,
    ])
      .pipe(takeUntil(this.unsubscribe$))
      .subscribe(([profile, idToken]) => {
        if (isEmpty(profile) && idToken?.['sub']) {
          this.store.dispatch(
            ProfileActions.createUserProfile({
              user:
this.userService.userTokenToProps(idToken),
            }),
          );
        }

        if (profile?.id) {

this.localStorageService.setItem(USER_PROFILE,
profile);

this.localStorageService.setItem(AUTH_TOKEN,
idToken?.['__raw']);

this.store.dispatch(CartActions.getCartByUserId({ id:
profile?.id }));
        }
        this.cdr.detectChanges();
      });


this.authWrapperService.authService.idTokenClaims$
      .pipe(takeUntil(this.unsubscribe$))
      .subscribe((token) => {
        this.store.dispatch(
          ProfileActions.getProfileInfoById({ id:
token?.['sub'] }),
        );
        this.cdr.detectChanges();
      });


    this.spinnerService.visibility
      .pipe(takeUntil(this.unsubscribe$))
      .subscribe((visibility) => {
        this.isShowLoadingSpinner = visibility;
        this.cdr.detectChanges();
      });
  }

  ngOnDestroy() {
    this.unsubscribe$.complete();
    this.unsubscribe$.unsubscribe();
  }
}
```

frontend/src/app/app.component.html

```
<!--<div class="nb-theme-corporateTheme"></div>-->
<nb-layout>
  <nb-layout-column>
    <ng-container *ngIf="isShowLoadingSpinner">
      <section
        class="absolute flex justify-center items-
center z-50 w-full h-full bg-gray-500 bg-opacity-50"
      >
        <div class="lds-ellipsis">
```

```
          <div></div>
          <div></div>
          <div></div>
          <div></div>
        </div>
      </section>
    </ng-container>

    <router-outlet></router-outlet>
  </nb-layout-column>
</nb-layout>
```

frontend/src/app/app.component.scss

```
.nb-theme-default
  nb-layout
  .layout
  .layout-container
  .content
  .columns
  nb-layout-column {
  @apply p-0;
}

.lds-ellipsis {
  display: inline-block;
  position: relative;
  width: 80px;
  height: 80px;
}
.lds-ellipsis div {
  position: absolute;
  top: 33px;
  width: 13px;
  height: 13px;
  border-radius: 50%;
  background: #fff;
  animation-timing-function: cubic-bezier(0, 1, 1,
0);
}
.lds-ellipsis div:nth-child(1) {
  left: 8px;
  animation: lds-ellipsis1 0.6s infinite;
}
.lds-ellipsis div:nth-child(2) {
  left: 8px;
  animation: lds-ellipsis2 0.6s infinite;
}
.lds-ellipsis div:nth-child(3) {
  left: 32px;
  animation: lds-ellipsis2 0.6s infinite;
}
.lds-ellipsis div:nth-child(4) {
  left: 56px;
  animation: lds-ellipsis3 0.6s infinite;
}
@keyframes lds-ellipsis1 {
  0% {
    transform: scale(0);
  }
  100% {
    transform: scale(1);
  }
}
@keyframes lds-ellipsis3 {
  0% {
    transform: scale(1);
  }
  100% {
    transform: scale(0);
  }
}
@keyframes lds-ellipsis2 {
  0% {
    transform: translate(0, 0);
  }
  100% {
    transform: translate(24px, 0);
  }
}
```

frontend/src/app/app-routing.module.ts

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from
'@angular/router';
import {
  AdminPageComponent,
  HomePageComponent,
  MapMarkSettingPageComponent,
  NotFoundPageComponent,
  OrderHistoryPageComponent,
  OrdersPageComponent,
  ProductsPageComponent,
  ProfilePageComponent,
  UsersPageComponent,
} from '../pages';
import { RoleGuard } from
'../common/guards/role.guard';
import { RouterPathEnum } from '../common/enums';
import { CatalogPageComponent } from
'../pages/catalog-page/catalog-page.component';
import { CheckoutPageComponent } from
'../pages/checkout-page/checkout-page.component';
import { ThankYouPageComponent } from
'../pages/thank-you-page/thank-you-page.component';
import { IdGuard } from '../common/guards/id.guard';
import { orderNumberGuard } from
'../common/guards/orderNumber.guard';
```

```
import { LogoutPageComponent } from '../pages/logout-page/logout-page.component';


const routes: Routes = [
  { path: '', redirectTo: RouterPathEnum.Home, pathMatch: 'full' },
  {
    path: RouterPathEnum.Home,
    component: HomePageComponent,
  },
  {
    path: RouterPathEnum.Profile,
    component: ProfilePageComponent,
  },
  {
    path: RouterPathEnum.Catalog,
    canActivate: [IdGuard],
    component: CatalogPageComponent,
  },
  {
    path: RouterPathEnum.Logout,
    component: LogoutPageComponent,
  },
  {
    path: RouterPathEnum.Checkout,
    canActivate: [IdGuard],
    component: CheckoutPageComponent,
  },
  {
    path: RouterPathEnum.ThankYou,
    canActivate: [orderNumberGuard],
    component: ThankYouPageComponent,
  },
  {
    path: RouterPathEnum.Orders,
    component: OrderHistoryPageComponent,
  },
  {
    path: RouterPathEnum.Admin,
    canActivate: [RoleGuard],
    component: AdminPageComponent,
    data: {
      expectedRole: 'admin',
    },
    children: [
      {
        path: RouterPathEnum.MapMarks,
        component: MapMarkSettingPageComponent,
      },
      {
        path: RouterPathEnum.Users,
        component: UsersPageComponent,
      },
      {
        path: RouterPathEnum.Products,
        component: ProductsPageComponent,
      },
      {
        path: RouterPathEnum.Orders,
        component: OrdersPageComponent,
      },
    ],
  },


  { path: '**', component: NotFoundPageComponent },
];


@NgModule({
  imports: [RouterModule.forRoot(routes, { anchorScrolling: 'enabled' })],
  exports: [RouterModule],
})
export class AppRoutingModule {}
```

frontend/src/app/app.module.ts

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { BrowserAnimationsModule } from '@angular/platform-browser/animations';
import { StoreDevtoolsModule } from '@ngrx/store-devtools';
import { StoreModule } from '@ngrx/store';
import {
  NbAccordionModule,
  NbButtonModule,
  NbCardModule,
  NbCheckboxModule,
  NbContextMenuModule,
  NbIconModule,
  NbInputModule,
  NbLayoutModule,
  NbMenuModule,
  NbSelectModule,
  NbSidebarModule,
  NbSpinnerModule,
  NbThemeModule,
} from '@nebular/theme';
import { FormsModule, ReactiveFormsModule } from '@angular/forms';
import { HTTP_INTERCEPTORS, HttpClientModule } from '@angular/common/http';
import { NavbarComponent } from
```

```
'src/components/navbar';
import { ButtonComponent } from
'../components/button/button.component';
import { NavigationItemComponent } from
'../components/navbar/navigation-item/navigation-
item.component';
import { NbEvaIconsModule } from '@nebular/eva-
icons';
import {
  AdminPageComponent,
  HomePageComponent,
  MapMarkSettingPageComponent,
  NotFoundPageComponent,
  OrderHistoryPageComponent,
  OrdersPageComponent,
  ProductsPageComponent,
  ProfilePageComponent,
  UsersPageComponent,
} from '../pages';
import { AuthModule } from '@auth0/auth0-angular';
import {
  cartReducer,
  mapMarkerReducer,
  orderReducer,
  productReducer,
  profileReducer,
  userReducer,
} from '../store/reducers';
import { EffectsModule } from '@ngrx/effects';
import {
  CartEffects,
  MapMarkersEffects,
  OrderEffects,
  ProductEffects,
  ProfileEffects,
  UserEffects,
} from '../store/effects';
import { GoogleMapsModule } from '@angular/google-
maps';
import { GoogleMapComponent } from
'../components/google-map/google-map.component';
import { AdminPanelSidebarComponent } from
'../components/admin-panel-sidebar/admin-panel-
sidebar.component';
import { ModalComponent } from
'../components/modal/modal.component';
import {
  AddMapMarkerFormComponent,
  EditUserFormComponent,
} from '../components/forms';
import { FormInputComponent } from
'../components/form-input/form-input.component';
import { CheckboxComponent } from
'../components/checkbox/checkbox.component';
import { DeleteMapMarkerComponent } from
'../components/delete-dialog';
import { DataTableComponent } from
'../components/data-table/data-table.component';
import {
  DxBulletModule,
  DxDataGridModule,
  DxTemplateModule,
} from 'devextreme-angular';
import { DeleteUserComponent } from
'../components/delete-dialog/delete-user.component';
import { SelectComponent } from
'../components/select/select.component';
import { NgxDropzoneModule } from 'ngx-dropzone';
import { AddProductComponent } from
'../components/forms/product/add-product/add-
product.component';
import { DeleteProductComponent } from
'../components/delete-dialog/delete-
product.component';
import { EditMapMarkerFormComponent } from
'../components/forms/map-merker/edit-map-marker-
form/edit-map-marker-form.component';
import { EditProductComponent } from
'../components/forms/product/edit-product/edit-
product.component';
import { CatalogPageComponent } from
'../pages/catalog-page/catalog-page.component';
import { CatalogItemComponent } from
'../components/catalog-item/catalog-item.component';
import { NgOptimizedImage } from '@angular/common';
import { FeatureSectionComponent } from
'../components/feature-section/feature-
section.component';
import { ClearCartItemsWarningComponent } from
'../components/info-dialog/clear-cart-items-
warning.component';
import { CheckoutPageComponent } from
'../pages/checkout-page/checkout-page.component';
import { ThankYouPageComponent } from
'../pages/thank-you-page/thank-you-page.component';
import { OrderDetailsProductComponent } from
'../components/order-details-product/order-details-
product.component';
import { LogoutPageComponent } from '../pages/logout-
page/logout-page.component';
import { EditOrderComponent } from
'../components/forms/order/edit-order/edit-
order.component';
import { OrderItemsComponent } from
'../components/forms/order/edit-order-items/order-
items/order-items.component';
import { EditOrderItemsComponent } from
'../components/forms/order/edit-order-items/edit-
order-items.component';
import { HeroSectionComponent } from
'../components/hero-section/hero-section.component';
import { FooterComponent } from
'../components/footer/footer.component';
import { LoadingInterceptor } from
'./loading.interseptor';

@NgModule({
  declarations: [
```

```
  // Components
  AppComponent,
  NavbarComponent,
  NavigationItemComponent,
  ButtonComponent,
  GoogleMapComponent,
  AdminPanelSidebarComponent,
  CheckboxComponent,
  FormInputComponent,
  SelectComponent,
  ModalComponent,
  DataTableComponent,
  CatalogItemComponent,
  FeatureSectionComponent,
  OrderDetailsProductComponent,
  OrderItemsComponent,
  HeroSectionComponent,
  FooterComponent,
  //Forms
  EditUserFormComponent,
  AddMapMarkerFormComponent,
  DeleteMapMarkerComponent,
  DeleteUserComponent,
  DeleteProductComponent,
  AddProductComponent,
  EditMapMarkerFormComponent,
  EditProductComponent,
  ClearCartItemsWarningComponent,
  EditOrderComponent,
  EditOrderItemsComponent,
  // Pages
  HomePageComponent,
  ProfilePageComponent,
  NotFoundPageComponent,
  AdminPageComponent,
  MapMarkSettingPageComponent,
  UsersPageComponent,
  ProductsPageComponent,
  OrdersPageComponent,
  CatalogPageComponent,
  CheckoutPageComponent,
  ThankYouPageComponent,
  LogoutPageComponent,
  OrderHistoryPageComponent,
],
imports: [
  BrowserModule,
  AppRoutingModule,
  GoogleMapsModule,
  BrowserAnimationsModule,
  HttpClientModule,
  ReactiveFormsModule,
  FormsModule,
    NgxDropzoneModule,
    NbButtonModule,
    NbSpinnerModule,
    NbIconModule,
    NbLayoutModule,
    NbEvaIconsModule,
    NbInputModule,
    NbCheckboxModule,
    NbSelectModule,
    NbSidebarModule,
    // DevExpress
    DxDataGridModule,
    DxTemplateModule,
    DxBulletModule,
    NbContextMenuModule,
    NbAccordionModule,
    NbCardModule,
    NbMenuModule.forRoot(),
    StoreModule.forRoot({
      ...profileReducer,
      ...cartReducer,
      ...mapMarkerReducer,
      ...userReducer,
      ...productReducer,
      ...orderReducer,
    }),
    NbThemeModule.forRoot({ name: 'corporateTheme'
}),
    AuthModule.forRoot({
      domain: 'dev-yn6tinfev7nut3ap.eu.auth0.com',
      clientId: 'XRauFMnZ8PkPNK3NKnNIIOaxWS63UNL3',
      useRefreshTokens: true,
      cacheLocation: 'localstorage',
      authorizationParams: {
        redirect_uri: window.location.origin,
      },
    }),
    EffectsModule.forRoot([
      ProfileEffects,
      CartEffects,
      MapMarkersEffects,
      UserEffects,
      ProductEffects,
      OrderEffects,
    ]),
    StoreDevtoolsModule.instrument({
      maxAge: 25, // Retains last 25 states
      logOnly: !true, // Restrict extension to log-
only mode
      autoPause: true, // Pauses recording actions
and state changes when the extension window is not
open
      trace: false, //  If set to true, will include
stack trace for every dispatched action, so you can
see it in trace tab jumping directly to that part of
```

code

```
      traceLimit: 75, // maximum stack trace frames
to be stored (in case trace option was provided as
true)
    }),
    NgOptimizedImage,
  ],
  providers: [
    {
      provide: HTTP_INTERCEPTORS,
      useClass: LoadingInterceptor,
      multi: true,
    },
  ],
  bootstrap: [AppComponent],
})
export class AppModule {}
```

frontend/src/app/loading.interseptor.ts

```
import { Injectable } from '@angular/core';
import {
  HttpEvent,
  HttpHandler,
  HttpInterceptor,
  HttpRequest,
} from '@angular/common/http';
import { finalize, Observable } from 'rxjs';
import { SpinnerService } from '../services';

@Injectable()
export class LoadingInterceptor implements
HttpInterceptor {
  constructor(private spinnerService: SpinnerService)
{}

  intercept(
    request: HttpRequest<unknown>,
    next: HttpHandler,
  ): Observable<HttpEvent<unknown>> {
    this.spinnerService.show();

    return next
      .handle(request)
      .pipe(finalize(() =>
this.spinnerService.hide()));
  }
}
```

frontend/src/pages/home-page/home-component-page.component.html

```
<app-navbar> </app-navbar>

<section class="mt-16">
  <app-hero-section></app-hero-section>

  <section class="my-4">
    <div
      class="gap-16 items-center py-8 px-8 mx-auto
max-w-screen-lg sm:grid sm:grid-cols-2 sm:py-16"
    >
      <div class="font-light text-gray-500 sm:text-lg
dark:text-gray-400">
        <h2 class="mb-4 text-4xl tracking-tight font-
extrabold text-gray-900">
          Як збільшення кількості парків може
перетворити ваше місто?
        </h2>
        <p class="mb-4">
          Життя в місті може бути незабутнім, коли у
нас є простір для
          відпочинку, зустрічей з друзями та насолоди
природою. Розширення
          кількості парків у нашому місті створює не
тільки прекрасні зелені
          оази, але й місця для релаксу, активного
відпочинку та спілкування.
          Кожен новий парк - це можливість зберегти і
приблизити природу до нас,
          покращити якість повітря та навколишню
атмосферу.
        </p>
        <p>
          Давайте разом додамо кольору нашому місту,
збільшимо кількість парків
          та створимо простір, який надихає і дарує
радість кожному жителю.
          Збережемо і примножимо наші парки -
найцінніше спадщина для сьогодення
          та майбутніх поколінь.
        </p>
      </div>
      <div class="grid grid-cols-2 gap-4 mt-8">
        <img
          class="w-full rounded-lg"

src="https://res.cloudinary.com/dykctctkb/image/uploa
d/v1684624067/playground_nejvzr.jpg"
          alt="office content 1"
        />
        <img
          class="mt-4 w-full lg:mt-10 rounded-lg"

src="https://res.cloudinary.com/dykctctkb/image/uploa
d/v1684624060/atidarytas-antrasis-verslo-centro-park-
town-pastatas-darnu-group-nuotr-82419087_b4phxx.jpg"
          alt="office content 2"
        />
      </div>
    </div>
  </section>

  <section id="google-map" class="max-w-screen-lg mt-
4 mx-auto">
```

```
    <nb-card>
      <div class="px-4 pt-4">
        <div class="flex justify-between items-
center">
          <h1 class="text-3xl font-extrabold text-
gray-700">
            Мапа доступних ділянок
          </h1>
        </div>
        <p class="mt-1 text-md font-light text-gray-
500">
          Видберіть ділянку яка вам подобається і
натисніть посадити дерево
        </p>
      </div>

      <div class="rounded overflow-hidden m-4">
        <app-google-map
          [options]="mapsOptions"

[markerInfoWindowTemplate]="markerInfoWindowTemplate"
        >
        </app-google-map>
      </div>
    </nb-card>
  </section>

  <app-feature-section id="FAQ" class="my-4"></app-
feature-section>

  <ng-template #markerInfoWindowTemplate let-
selectedMarker>
    <form
      class="flex flex-col p-2 gap-2 max-w-xs"
      [formGroup]="markerForm"
      novalidate
    >
      <h1 class="text-lg">{{
selectedMarker?.label?.text }}</h1>
      <span>{{ selectedMarker?.description }}</span>
      <app-select
        placeholder="Продукти"
        formControlName="orderType"
        label="Тип вконання замовлення"
        [selectOptions]="orderTypeOptions"
      >
      </app-select>
      <app-button
        label="Продовжити"
        size="small"

(onClick)="catalogRedirection(selectedMarker)"
      >
      </app-button>
    </form>
  </ng-template>
</section>

<app-footer />
```

frontend/src/pages/home-page/home-component-page.component.ts

```
import { Component, OnDestroy, OnInit } from
'@angular/core';
import { ukraineBounds } from
'../../common/consts/map-ukraine-borders.const';
import { Subject } from 'rxjs';
import { Store } from '@ngrx/store';
import { MapMarkerActions } from
'../../store/actions';
import {
  RouterPathEnum,
  UserRolesEnum,
  OrderTypeEnum,
} from '../../common/enums';
import { Router } from '@angular/router';
import { FormGroup, FormControl } from
'@angular/forms';

@Component({
  selector: 'app-home-page',
  templateUrl: './home-component-
page.component.html',
  styleUrls: ['./home-component-
page.component.scss'],
})
export class HomePageComponent implements OnInit,
OnDestroy {
  markerForm: FormGroup;

  orderTypeOptions = [
    { id: 'general', name: 'Загальльний' },
    { id: 'custom', name: 'Власноруч' },
  ];
  mapsOptions: google.maps.MapOptions = {
    center: ukraineBounds.getCenter(),
    zoom: 7,
    restriction: {
      latLngBounds: ukraineBounds,
      strictBounds: true,
    },
    zoomControl: true,
    scrollwheel: true,
    mapTypeId: 'hybrid',
  };

  private readonly unsubscribe$ = new Subject();

  constructor(
```

```
    private readonly store: Store<any>,
    private readonly router: Router,
  ) {
    this.markerForm = new FormGroup({
      orderType: new
FormControl(OrderTypeEnum.General, []),
    });
  }


  ngOnInit(): void {
    this.store.dispatch(
      MapMarkerActions.getMapMarkerByRoleId({ roleId:
UserRolesEnum.Customer }),
    );
  }


  catalogRedirection(selectedMarker: any): void {
    const formData = this.markerForm.value;


    this.router.navigate([RouterPathEnum.Catalog], {
      queryParams: { id: selectedMarker?.id,
orderType: formData.orderType },
    });
  }


  ngOnDestroy(): void {
    this.unsubscribe$.complete();
    this.unsubscribe$.unsubscribe();
  }
}
```

frontend/src/pages/catalog-page/catalog-page.component.html

```
<app-navbar></app-navbar>


<section class="catalog-page-layout">
  <div>
    <div class="pt-10">
      <div class="flex justify-center items-center h-
full mb-10">
        <app-button
          icon="arrow-ios-back-outline"
          iconStatus="basic"
          size="medium"
          [isGhost]="true"
          (onClick)="back()"
        ></app-button>
        <h1 class="text-2xl font-bold ml-2">Каталог
продуктів</h1>
      </div>
      <div
        class="mx-auto max-w-5xl w-full justify-
center flex-col px-6 md:flex xl:px-0 gap-3"
      >
        <div
          class="h-full w-full items-start flex
rounded-lg border bg-white border-[#ffaa00] p-3 gap-2
shadow-md"
        >
          <nb-icon class="h-8 w-8" icon="info"
status="warning"></nb-icon>
          <span *ngIf="orderType === 'custom'"
class="text-[#ffaa00]">
            Ви обрали тип виконання замовлення
            <span class="font-bold text-
[#6fbf6b]">власноруч</span>. Однієї з
            вимог, все ще є попка правильну
підходящих санджансів дерев у нас.
            <br />
            <br />
            <span class="italic text-gray-500">
              Будь ласка, переконайтеся, що ви
вибрали правильну ділянку для
              посадки дерев.
            </span>
          </span>
        </div>


        <div class="max-w-5xl justify-center md:flex
xl:px-0 gap-2">
          <div class="rounded-lg md:w-2/3">
            <ng-container *ngFor="let product of
productsCatalog">
              <app-catalog-item [product]="product">
</app-catalog-item>
            </ng-container>
          </div>
          <!-- Sub total -->
          <div
            class="mt-6 h-full rounded-lg border bg-
white p-6 shadow-md md:mt-0 md:w-1/3"
          >
            <div class="mb-2 flex justify-between">
              <p class="text-gray-700">Проміжний
підсумок</p>
              <p class="text-gray-700">{{
productsSubTotal }} грн.</p>
            </div>
            <div class="mb-2 flex justify-between">
              <p class="text-gray-700">Послуги
сервісу</p>
              <p class="text-gray-700"
*ngIf="orderType === 'custom'">---</p>
              <p class="text-gray-700"
*ngIf="orderType !== 'custom'">
                {{ serviceCost }} грн.
              </p>
            </div>
            <div class="flex justify-between">
              <p class="text-gray-700">Доставка</p>
              <p class="text-gray-
700">безкоштовна</p>
```

```
            </div>
            <hr class="my-4" />
            <div class="flex justify-between">
              <p class="text-lg font-bold">Сума</p>
              <p class="mb-1 text-lg font-bold">{{
productsSubTotal }} грн.</p>
            </div>
            <div class="flex justify-end mt-2 w-
full">
              <app-button
                class="w-full"
                [iconOnStart]="false"
                icon="arrow-ios-forward-outline"
                iconStatus="basic"
                size="medium"
                label="Замовити"
                [disabled]="isDisableCheckoutButton"
                (onClick)="navigateToCheckout()"
              ></app-button>
            </div>
          </div>
        </div>
      </div>
    </div>
  </div>

  <app-footer></app-footer>
</section>

<section *ngIf="modalState.open" class="absolute">
  <app-modal></app-modal>
</section>
```

frontend/src/pages/catalog-page/catalog-page.component.ts

```
import { Component, Injector, OnDestroy, OnInit }
from '@angular/core';
import { Store } from '@ngrx/store';
import { CartActions, ProductActions } from
'../../store/actions';
import { ActivatedRoute, Router } from
'@angular/router';
import { combineLatest, filter, Subject, take,
takeUntil } from 'rxjs';
import { CartSelectors, ProductSelectors } from
'../../store/selectors';
import { ICart, IProduct } from
'../../common/interfaces';
import { RouterPathEnum } from '../../common/enums';
import { LocalStorageService, ModalService } from
'../../services';
import {
  BaseCartItemsForDelete,
  CartItemsForDelete,
} from '../../common/classes';
import { ClearCartItemsWarningComponent } from
'../../components/info-dialog/clear-cart-items-
warning.component';
import { USER_PROFILE } from '../../common/local-
storage-keys';

@Component({
  selector: 'app-catalog-page',
  templateUrl: './catalog-page.component.html',
  styleUrls: ['./catalog-page.component.scss'],
})
export class CatalogPageComponent implements OnInit,
OnDestroy {
  modalState: { content: any; open: boolean;
injector: any };
  productsCatalog: Array<IProduct>;
  idsForCartItemDelete: Array<number>;
  cart: ICart;
  markerId = 0;
  orderType: string = '';
  serviceCost = 100;

  private readonly unsubscribe$ = new Subject();
  constructor(
    private readonly store: Store<any>,
    private readonly activatedRoute: ActivatedRoute,
    private readonly router: Router,
    private readonly modalService: ModalService,
    private readonly localStorageService:
LocalStorageService,
    private inj: Injector,
  ) {}

  ngOnInit(): void {
    this.activatedRoute.queryParams
      .pipe(
        take(1),
        filter((params) => !!params?.['id']),
      )
      .subscribe(({ id, orderType }) => {
        this.markerId = Number(id);
        this.orderType = orderType || 'general'; //
Зберігаємо тип замовлення

this.store.dispatch(ProductActions.getCatalog({ id
}));
      });

    const userId = JSON.parse(
      this.localStorageService.getItem(USER_PROFILE)
|| {},
    )?.id;
    this.store.dispatch(CartActions.getCartByUserId({
id: userId }));

    combineLatest([
```

```
this.store.select(ProductSelectors.getProductsCatalog
),
      this.store.select(CartSelectors.getCart),
    ])
      .pipe(takeUntil(this.unsubscribe$))
      .subscribe(([productsCatalog, cart]) => {
        this.cart = cart;
        this.productsCatalog = productsCatalog;

        if (cart?.items?.length) {
          this.findIdsForCartItemsDelete();

          if (this.isShowingWarningDialog) {
            this.showWarningDialog();
          }
        }
      });

    this.modalService.modalSource.subscribe(
      (value: any) => (this.modalState = value),
    );
  }

  onOrderTypeChange(event: Event) {
    const selectElement = event.target as
HTMLSelectElement;
    this.orderType = selectElement.value;
  }

  showWarningDialog(): void {
    const injector: Injector = Injector.create({
      providers: [
        {
          provide: BaseCartItemsForDelete,
          useValue: new
CartItemsForDelete(this.idsForCartItemDelete),
        },
      ],
      parent: this.inj,
    });


this.modalService.openNewModal(ClearCartItemsWarningC
omponent, injector);
  }

  findIdsForCartItemsDelete(): void {
    const cartItems = this.cart.items;
    this.idsForCartItemDelete =
      cartItems
        ?.filter(
          (item) =>
            !this.productsCatalog?.some((obj) =>
obj.id === item.productId),
        )
        .map((item) => item.id) || [];
  }

  get isShowingWarningDialog(): boolean {
    return !!this.cart?.items?.length &&
!!this.productsCatalog?.length
      ? !!this.idsForCartItemDelete?.length
      : false;
  }

  get productsSubTotal(): number {
    const productsCost = this.cart?.items?.reduce(
      (sum, product) => sum + product.price *
product.quantity,
      0,
    );

    if (this.orderType === 'custom') {
      return productsCost || 0;
    }

    return productsCost + this.serviceCost || 0;
  }

  get isDisableCheckoutButton(): boolean {
    return !this.cart?.items?.length;
  }

  navigateToCheckout(): void {
    this.router.navigate([RouterPathEnum.Checkout], {
      queryParams: { id: this.markerId, orderType:
this.orderType },
    });
  }

  back(): void {
    this.router.navigate([RouterPathEnum.Home]);
  }

  ngOnDestroy(): void {
    this.unsubscribe$.complete();
    this.unsubscribe$.unsubscribe();
  }
}
```

frontend/src/pages/checkout-page/checkout-page.component.html

```
<section
  class="flex justify-center w-screen min-h-screen
bg-gray-100 text-gray-800 p-8"
>
  <div class="grid lg:grid-cols-3 md:grid-cols-2 gap-
8 w-full max-w-screen-lg">
```

```
    <div class="lg:col-span-2">
      <h2 class="text-sm font-medium">Спосіб оплати</h2>
      <div class="bg-white rounded mt-4 shadow-lg">
        <div class="flex items-center px-8 py-5">
          <input
            class="appearance-none w-4 h-4 rounded-full border-2 border-white ring-2 ring-primary-700 ring-opacity-100 bg-primary-700"
            type="radio"
          />
          <label class="text-sm font-medium ml-4">Кредитна карта</label>
        </div>
        <form
          [formGroup]="form"
          novalidate
          class="grid grid-cols-2 gap-4 px-8 pb-8"
        >
          <div class="col-span-2">
            <app-form-input
              type="text"
              placeholder="0000 0000 0000 0000"
              formControlName="creditCard"
              label="Номер картки"
            >
            </app-form-input>
          </div>
          <div class="">
            <app-form-input
              type="text"
              placeholder="00/00"
              formControlName="expirationDate"
              label="Термін дії"
            >
            </app-form-input>
          </div>
          <div class="">
            <app-form-input
              type="text"
              placeholder="000"
              formControlName="cvc"
              label="CVV"
            >
            </app-form-input>
          </div>
        </form>
      </div>
    </div>
    <div>
      <h2 class="text-sm font-medium">Підсумок покупки</h2>
      <div class="bg-white rounded mt-4 shadow-lg py-6">
        <div class="px-8"></div>
        <div class="px-8">
          <div class="flex items-end justify-between">
            <span class="text-sm font-semibold">Сума</span>
            <span class="text-sm text-gray-500 mb-px"
              >{{ orderSubtotal }} грн.</span
            >
          </div>
          <div class="flex items-end justify-between">
            <span class="text-sm font-semibold">Податок</span>
            <span class="text-sm text-gray-500 mb-px">10%</span>
          </div>
        </div>
        <div class="px-8 mt-4 border-t pt-4">
          <div class="flex items-end justify-between">
            <span class="font-semibold">До оплати</span>
            <span class="font-semibold">{{ orderTotal }} грн.</span>
          </div>
        </div>
        <div class="flex items-center px-8 mt-8">
          <app-checkbox
            label="Посадити дерево анонімно"
            [formControl]="anonymousControl"
          >
          </app-checkbox>
        </div>
        <div class="flex flex-col px-8 pt-4">
          <app-button
            class="flex items-center justify-center text-sm font-medium w-full h-10 rounded"
            size="medium"
            label="Оплатити"
            (onClick)="onSubmit()"
          ></app-button>

          <button
            class="text-xs text-blue-500 mt-3 underline"
            (click)="navigateToCatalog()"
          >
            Назад
          </button>
        </div>
      </div>
    </div>
  </div>
</section>
```

frontend/src/pages/checkout-page/checkout-page.component.ts

```
import { Component, OnDestroy, OnInit } from '@angular/core';
import { Store } from '@ngrx/store';
import { filter, Subject, take, takeUntil } from 'rxjs';
import { CartSelectors } from '../../store/selectors';
import { ICart, ICartItem } from '../../common/interfaces';
import {
  FormBuilder,
  FormControl,
  FormGroup,
  Validators,
} from '@angular/forms';
import { CreditCardValidators } from 'angular-cc-library';
import { PaymentProvideEnum, RouterPathEnum } from '../../common/enums';
import { ActivatedRoute, Router } from '@angular/router';
import { OrderActions } from '../../store/actions';

@Component({
  selector: 'app-checkout-page',
  templateUrl: './checkout-page.component.html',
  styleUrls: ['./checkout-page.component.scss'],
})
export class CheckoutPageComponent implements OnInit, OnDestroy {
  orderSubtotal = 0;
  orderTotal = 0;
  form: FormGroup;
  markerId = 0;
  orderItems: Array<ICartItem> = [];
  orderType = '';

  anonymousControl = new FormControl(false);

  private readonly unsubscribe$ = new Subject();
  constructor(
    private readonly store: Store<any>,
    private readonly activatedRoute: ActivatedRoute,
    private readonly fb: FormBuilder,
    private readonly router: Router,
  ) {}
  ngOnInit(): void {
    this.activatedRoute.queryParams
      .pipe(
        take(1),
        filter((params) => !!params?.['id']),
      )
      .subscribe(({ id, orderType }) => {
        this.markerId = Number(id);
        this.orderType = orderType || 'general';
      });

    this.form = this.fb.group({
      creditCard: ['', [CreditCardValidators.validateCCNumber]],
      expirationDate: ['', [CreditCardValidators.validateExpDate]],
      cvc: [
        '',
        [Validators.required, Validators.minLength(3), Validators.maxLength(4)],
      ],
    });

    this.store
      .select(CartSelectors.getCart)
      .pipe(
        filter((cart) => !!cart?.id),
        takeUntil(this.unsubscribe$),
      )
      .subscribe((cart) => {
        this.serOrderTotal(cart);
        this.orderItems = cart?.items;
      });
  }

  serOrderTotal(cart: ICart): void {
    this.orderSubtotal =
      cart?.items?.reduce(
        (sum, product) => sum + product.price * product.quantity,
        0,
      ) || 0;
    const tax10 = this.orderSubtotal * 0.1;
    this.orderTotal = this.orderSubtotal + tax10;
  }

  onSubmit() {
    const orderDto = {
      total: this.orderTotal,
      type: this.orderType,
      mapMarkerId: this.markerId,
      isAnonymous: this.anonymousControl.value || false,
      items: this.orderItems.map(({ productId, name, quantity }) => ({
        productId,
        name,
        quantity,
      })) as any,
      payment: {
        provider: PaymentProvideEnum.CreditCard,
      },
```

```
    };

    this.store.dispatch(OrderActions.createOrder({
orderDto }));
  }

  navigateToCatalog(): void {
    this.router.navigate([RouterPathEnum.Catalog], {
      queryParams: { id: this.markerId },
    });
  }

  ngOnDestroy(): void {
    this.unsubscribe$.complete();
    this.unsubscribe$.unsubscribe();
  }
}
```

frontend/src/pages/not-found-page/not-found-page.component.html

```
<section class="flex items-center h-full p-16">
  <div
    class="container flex flex-col items-center
justify-center px-5 mx-auto my-8"
  >
    <div class="max-w-md text-center">
      <h2 class="mb-8 font-extrabold text-9xl
dark:text-gray-600">
        <span class="sr-only">Error</span>404
      </h2>
      <p class="text-2xl font-semibold md:text-3xl">
        Sorry, we couldn't find this page.
      </p>
      <p class="mt-4 mb-8 dark:text-gray-400">
        But dont worry, you can find plenty of other
things on our homepage.
      </p>
      <a
        rel="noopener noreferrer"
        href="#"
        class="px-8 py-3 font-semibold rounded
dark:bg-violet-400 dark:text-gray-900"
        >Back to homepage</a
      >
    </div>
  </div>
</section>
```

frontend/src/pages/not-found-page/not-found-page.component.ts

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-not-found-page',
  templateUrl: './not-found-page.component.html',
  styleUrls: ['./not-found-page.component.scss'],
})
export class NotFoundPageComponent {}
```

frontend/src/pages/order-history-page/order-history-page.component.html

```
<app-navbar></app-navbar>

<section class="profile-page-layout">
  <div
    class="container flex flex-col sm:flex-row gap-4
items-start justify-center px-5 mx-auto my-8"
  >
    <nb-card class="min-w-full sm:min-w-[18rem]">
      <nb-menu [items]="items"> </nb-menu>
    </nb-card>

    <div class="w-full sm:w-9/12">
      <nb-accordion class="w-full">
        <nb-accordion-item *ngFor="let order of
orderHistory">
          <nb-accordion-item-header>
            <div class="flex justify-between items-
center w-11/12">
              <div>Замовлення #{{ order.id }}</div>
              <div class="text-primary-300">{{
order.total }} грн.</div>
            </div>
          </nb-accordion-item-header>
          <nb-accordion-item-body>
            <div class="w-full">
              <div>
                <div
                  class="md:grid md:grid-cols-2
hover:bg-gray-50 md:space-y-0 space-y-1 p-4 border-b"
                >
                  <p class="text-gray-
600">Статус:</p>
                  <p>{{ getStatusTitle(order.status)
}}</p>
                </div>
                <div
                  class="md:grid md:grid-cols-2
hover:bg-gray-50 md:space-y-0 space-y-1 p-4 border-b"
                >
                  <p class="text-gray-600">Сума:</p>
                  <p class="text-primary-300">{{
order.total }} грн.</p>
                </div>
                <div
                  class="md:grid md:grid-cols-2
hover:bg-gray-50 md:space-y-0 space-y-1 p-4 border-b"
                >
```

```
                  <p class="text-gray-600">Спосіб
оплати:</p>
                  <p>{{ order.paymentDetails.provider
}}</p>
                </div>
                <div
                  class="md:grid md:grid-cols-2
hover:bg-gray-50 md:space-y-0 space-y-1 p-4 border-b"
                >
                  <p class="text-gray-600">Номер
ділянки:</p>
                  <p>#{{ order.mapMarkerId }}</p>
                </div>

                <div
                  class="md:grid md:grid-cols-2
hover:bg-gray-50 md:space-y-0 space-y-1 p-4 border-b"
                >
                  <p class="text-gray-
600">Продукти:</p>
                  <div>
                    <ng-container *ngFor="let product
of order?.items">
                      <app-order-details-product
[product]="product">
                      </app-order-details-product>
                    </ng-container>
                  </div>
                </div>
              </div>
            </div>
          </nb-accordion-item-body>
        </nb-accordion-item>
      </nb-accordion>
    </div>
  </div>
</section>

<app-footer></app-footer>
```

frontend/src/pages/order-history-page/order-history-page.component.ts

```
import { Component, OnDestroy, OnInit } from
'@angular/core';
import { NbMenuItem } from '@nebular/theme';
import { profileMenuConstants } from
'../../common/consts/profile-menu.constants';
import { filter, Subject, takeUntil } from 'rxjs';
import { Store } from '@ngrx/store';
import { ProfileActions } from '../../store/actions';
import { ProfileSelectors } from
'../../store/selectors';
import { IOrderDetails } from
'../../common/interfaces';
import { OrderStatusEnum, OrderStatusTitleEnum } from
'../../common/enums';

@Component({
  selector: 'app-order-history-page',
  templateUrl: './order-history-page.component.html',
  styleUrls: ['./order-history-page.component.scss'],
})
export class OrderHistoryPageComponent implements
OnInit, OnDestroy {
  items: NbMenuItem[] = profileMenuConstants;
  orderHistory: Array<IOrderDetails>;

  private readonly unsubscribe$ = new Subject();
  constructor(private readonly store: Store<any>) {}
  ngOnInit(): void {

this.store.dispatch(ProfileActions.getOrderHistory())
;


    this.store
      .select(ProfileSelectors.getOrderHistory)
      .pipe(
        filter((orderHistory) =>
!!orderHistory.length),
        takeUntil(this.unsubscribe$),
      )
      .subscribe((orderHistory) => {
        this.orderHistory = orderHistory;
      });
  }

  getStatusTitle(status: number): string {
    switch (status) {
      case OrderStatusEnum.Created:
        return OrderStatusTitleEnum.Created;
      case OrderStatusEnum.Canceled:
        return OrderStatusTitleEnum.Canceled;
      case OrderStatusEnum.InProcess:
        return OrderStatusTitleEnum.InProcess;
      case OrderStatusEnum.Completed:
        return OrderStatusTitleEnum.Completed;
      default:
        return OrderStatusTitleEnum.Created;
    }
  }

  ngOnDestroy() {
    this.unsubscribe$.complete();
    this.unsubscribe$.unsubscribe();
  }
}
```

frontend/src/pages/profile-page/profile-component-page.component.html

```
<app-navbar></app-navbar>

<section class="profile-page-layout">
  <div
    class="container flex flex-col sm:flex-row gap-4
items-start justify-center px-5 mx-auto my-8"
  >
    <nb-card class="min-w-full sm:min-w-[18rem]">
      <nb-menu [items]="items"> </nb-menu>
    </nb-card>

    <nb-card class="w-full">
      <div class="px-4 sm:px-8 py-6">
        <div class="text-center">
          <h1 class="text-xl font-semibold">Профіль
користувача</h1>
          <small class="text-gray-400"> #{{
profile.id }} </small>
        </div>

        <div class="flex items-center justify-center
mx-auto w-full mt-4">
          <form [formGroup]="userForm" novalidate>
            <div class="-mx-3 flex flex-wrap">
              <div class="w-full px-3 sm:w-1/2">
                <div class="mb-5">
                  <app-form-input
                    placeholder="xxxxxx xxxxx"
                    formControlName="name"
                    label="Прізвище Ім'я"
                  >
                  </app-form-input>
                </div>
              </div>
              <div class="w-full px-3 sm:w-1/2">
                <div class="mb-5">
                  <app-form-input
                    placeholder="xxxxxx@xxx.xxx"
                    formControlName="email"
                    label="Пошта"
                  >
                  </app-form-input>
                </div>
              </div>
            </div>

            <div class="flex justify-center">
              <app-button
                class="w-full sm:w-6/12"
                size="medium"
                buttonStatus="primary"
                iconStatus="basic"
                label="Оновити дані"
                [disabled]="userForm.invalid"
                (onClick)="updateProfile()"
              >
              </app-button>
            </div>
          </form>
        </div>
      </div>
    </nb-card>
  </div>

  <app-footer></app-footer>
</section>
```

frontend/src/pages/profile-page/profile-component-page.component.ts

```
import { Component, OnDestroy, OnInit } from
'@angular/core';
import { Store } from '@ngrx/store';
import { filter, Subject, takeUntil } from 'rxjs';
import { ProfileSelectors } from
'../../store/selectors';
import { NbMenuItem } from '@nebular/theme';
import { profileMenuConstants } from
'../../common/consts/profile-menu.constants';
import { FormControl, FormGroup, Validators } from
'@angular/forms';
import { IUser } from '../../common/interfaces';
import { ProfileActions } from '../../store/actions';

@Component({
  selector: 'app-profile-page',
  templateUrl: './profile-component-
page.component.html',
  styleUrls: ['./profile-component-
page.component.scss'],
})
export class ProfilePageComponent implements OnInit,
OnDestroy {
  items: NbMenuItem[] = profileMenuConstants;
  userForm: FormGroup;
  profile: IUser;

  private readonly unsubscribe$ = new Subject();
  constructor(private readonly store: Store<any>) {
    this.userForm = new FormGroup({
      name: new FormControl('', [
        Validators.required,
        Validators.minLength(3),
        Validators.maxLength(25),
      ]),
      email: new FormControl('', [
        Validators.required,
        Validators.minLength(3),
        Validators.maxLength(25),
```

```
        Validators.email,
      ]),
    });
  }
  ngOnInit(): void {
    this.store
      .select(ProfileSelectors.getProfile)
      .pipe(
        filter((profile) => !!profile),
        takeUntil(this.unsubscribe$),
      )
      .subscribe((profile) => {
        this.profile = profile;
        this.prefillUserForm(profile);
      });
  }

  prefillUserForm(profile: IUser): void {
    this.userForm.patchValue({ name: profile.name,
email: profile.email });
  }

  updateProfile(): void {
    const formData = this.userForm.value;
    this.store.dispatch(
      ProfileActions.updateProfile({
        id: this.profile.id as string,
        user: {
          name: formData?.name,
          email: formData?.email,
        } as any,
      }),
    );
  }

  ngOnDestroy() {
    this.unsubscribe$.complete();
    this.unsubscribe$.unsubscribe();
  }
}
```

frontend/src/pages/thank-you-page/thank-you-page.component.html

```
<app-navbar></app-navbar>

<section class="thank-you-page-layout">
  <div
    class="container flex flex-col items-center
justify-center px-5 mx-auto my-8"
  >
    <div class="max-w-5xl flex flex-col items-center
justify-center gap-2">
      <h2 class="mb-2 font-extrabold text-center
text-3xl dark:text-gray-600">
        Створено заволення #{{ orderNumber }}
      </h2>
      <p class="text-base text-center mb-6 max-w-
4xl">
        Дякуємо за те, що ви долучилися до озеленення
міста та допомогли
        зберегти нашу планету! Ваш внесок є
надзвичайно важливим для нашої
        спільної мети створення здорового та зеленого
середовища для всіх нас.
      </p>

      <div class="flex flex-col md:flex-row gap-6
justify-center">
        <img
          class="h-40 md:h-80 md:block hidden"
          alt="undraw_completing_re_i7ap"

src="assets/completing/undraw_completing_re_i7ap.svg"
        />
        <div>
          <section class="mt-2">
            <ng-container *ngFor="let product of
orderDetails?.items">
              <app-order-details-product
[product]="product">
              </app-order-details-product>
            </ng-container>
          </section>
        </div>
      </div>

      <app-button
        class="min-w-96"
        [iconOnStart]="false"
        iconStatus="basic"
        size="medium"
        label="Назад на головну"
        (onClick)="navigateToHome()"
      ></app-button>
    </div>
  </div>

  <app-footer></app-footer>
</section>
```

frontend/src/pages/thank-you-page/thank-you-page.component.ts

```
import { Component, OnDestroy, OnInit } from
'@angular/core';
import { ActivatedRoute, Router } from
'@angular/router';
import { filter, Subject, take, takeUntil } from
'rxjs';
```

```
import { USER_PROFILE } from '../../common/local-storage-keys';
import { CartActions, OrderActions } from '../../store/actions';
import { Store } from '@ngrx/store';
import { LocalStorageService } from '../../services';
import { OrderSelectors } from '../../store/selectors';
import { IOrderDetails } from '../../common/interfaces';
import { RouterPathEnum } from '../../common/enums';

@Component({
  selector: 'app-thank-you-page',
  templateUrl: './thank-you-page.component.html',
  styleUrls: ['./thank-you-page.component.scss'],
})
export class ThankYouPageComponent implements OnInit, OnDestroy {
  orderNumber: number;
  orderDetails: IOrderDetails;

  private readonly unsubscribe$ = new Subject();
  constructor(
    private readonly activatedRoute: ActivatedRoute,
    private readonly router: Router,
    private readonly store: Store<any>,
    private readonly localStorageService: LocalStorageService,
  ) {}

  ngOnInit(): void {
    const userId = JSON.parse(
      this.localStorageService.getItem(USER_PROFILE) || {},
    )?.id;
    this.store.dispatch(CartActions.getCartByUserId({ id: userId }));

    this.activatedRoute.queryParams
      .pipe(
        take(1),
        filter((params) => !!params?.['orderNumber']),
      )
      .subscribe(({ orderNumber }) => {
        this.orderNumber = Number(orderNumber);
        this.store.dispatch(
          OrderActions.getOrderDetails({ orderNumber: this.orderNumber }),
        );
      });

    this.store
      .select(OrderSelectors.getOrderDetails)
      .pipe(takeUntil(this.unsubscribe$))
      .subscribe((orderDetails) => (this.orderDetails = orderDetails));
  }

  navigateToHome(): void {
    this.router.navigate([RouterPathEnum.Home]);
  }

  ngOnDestroy() {
    this.unsubscribe$.complete();
    this.unsubscribe$.unsubscribe();
  }
}
```

frontend/src/pages/admin-path/admin-page/admin-page.component.html

```
<section class="flex" [ngClass]="{ 'overflow-hidden': isGuardVisible }">
  <section
    *ngIf="isGuardVisible"
    class="absolute flex flex-col justify-center items-center z-50 w-full h-full bg-primary-700"
  >
    <p class="text-white text-base sm:text-xl text-center">
      Адмін панель не доступна на даному розширенні екрану
    </p>
    <app-button
      class="mt-4"
      buttonStatus="control"
      size="medium"
      label="Повернутись на головну"
      (onClick)="backToHomePage()"
    >
    </app-button>
  </section>
  <app-admin-panel-sidebar> </app-admin-panel-sidebar>
  <div class="w-full">
    <router-outlet></router-outlet>
  </div>

  <section *ngIf="modalState.open" class="absolute">
    <app-modal></app-modal>
  </section>
</section>
```

frontend/src/pages/admin-path/admin-page/admin-page.component.ts

```
import { Component, HostListener, OnInit } from '@angular/core';
import { Router } from '@angular/router';
```

```
import { ModalService } from '../../../services';
import { RouterPathEnum } from
'../../../common/enums';

@Component({
  selector: 'app-admin-page',
  templateUrl: './admin-page.component.html',
  styleUrls: ['./admin-page.component.scss'],
})
export class AdminPageComponent implements OnInit {
  modalState: { content: any; open: boolean;
injector: any };
  innerWidth: number;

  constructor(
    private readonly router: Router,
    private readonly modalService: ModalService,
  ) {
    if (router.url === '/admin') {
      this.router.navigate(['admin', 'map-marks']);
    }
  }
  ngOnInit(): void {
    this.innerWidth = window.innerWidth;

    this.modalService.modalSource.subscribe(
      (value: any) => (this.modalState = value),
    );
  }

  @HostListener('window:resize', ['$event'])
  onResize(event: any) {
    this.innerWidth = window.innerWidth;
  }

  get isGuardVisible(): boolean {
    return this.innerWidth <= 680;
  }

  backToHomePage(): void {
    this.router.navigate([RouterPathEnum.Home]);
  }
}
```

frontend/src/pages/admin-path/map-mark-setting-page/map-mark-setting-page.component.html

```
<section class="mx-4 mt-4">
  <nb-card>
    <div class="p-4">
      <div class="flex justify-between items-center">
        <h1 class="text-3xl font-medium text-gray-
700">Мапа ділянок</h1>
      </div>
      <p class="mt-1 text-md font-light text-gray-
500">
        Натисніть два рази по екрані, щоб створити
нову ділянку
      </p>
    </div>
  </nb-card>
  <nb-card>
    <div class="rounded overflow-hidden m-4">
      <app-google-map
        [options]="mapsOptions"
        [mapType]="mapTypeEnum.Admin"

[markerInfoWindowTemplate]="markerInfoWindowTemplate"
      ></app-google-map>
    </div>
  </nb-card>
</section>
<ng-template #markerInfoWindowTemplate let-
selectedMarker>
  <div class="flex flex-col p-2 gap-2 max-w-xs">
    <h1 class="text-lg">{{
selectedMarker?.label?.text }}</h1>
    <span>{{ selectedMarker?.description }}</span>

    <div class="flex p-2 gap-x-2 justify-center">
      <app-button
        [icon]="selectedMarker?.hidden ? 'eye-
outline' : 'eye-off-outline'"
        size="medium"
        buttonStatus="warning"
        iconStatus="basic"

(onClick)="updateMapMarkerVisibility(selectedMarker)"
      />
      <app-button
        icon="edit-2-outline"
        size="medium"
        buttonStatus="info"
        iconStatus="basic"
        (onClick)="editMarker(selectedMarker)"
      />
      <app-button
        icon="trash-outline"
        size="medium"
        buttonStatus="danger"
        iconStatus="basic"
        (onClick)="deleteMapMarker(selectedMarker)"
      />
    </div>
  </div>
</ng-template>
```

frontend/src/pages/admin-path/map-mark-setting-

page/map-mark-setting-page.component.ts

```
import { Component, Injector, OnDestroy, OnInit } from '@angular/core';
import { Store } from '@ngrx/store';
import { MapMarkerActions, ProductActions } from '../../../store/actions';
import { ukraineBounds } from '../../../common/consts/map-ukraine-borders.const';
import { ProfileSelectors } from '../../../store/selectors';
import { filter, Subject, takeUntil } from 'rxjs';
import { BaseMarker, Marker } from '../../../common/classes';
import { EditMapMarkerFormComponent } from '../../../components/forms/map-merker/edit-map-marker-form/edit-map-marker-form.component';
import { DeleteMapMarkerComponent } from '../../../components/delete-dialog';
import { ModalService } from '../../../services';
import { MapTypeEnum } from '../../../common/enums/map';
import { TableTypeEnum } from '../../../common/enums';


@Component({
  selector: 'app-map-mark-setting-page',
  templateUrl: './map-mark-setting-page.component.html',
  styleUrls: ['./map-mark-setting-page.component.scss'],
})
export class MapMarkSettingPageComponent implements OnInit, OnDestroy {
  mapsOptions: google.maps.MapOptions = {
    center: ukraineBounds.getCenter(),
    zoom: 7,
    restriction: {
      latLngBounds: ukraineBounds,
      strictBounds: true,
    },
    zoomControl: true,
    scrollwheel: true,
    mapTypeId: 'hybrid',
    disableDoubleClickZoom: true,
    streetViewControl: false,
  };
  mapTypeEnum = MapTypeEnum;

  private readonly unsubscribe$ = new Subject();

  constructor(
    private readonly store: Store<any>,
    private readonly modalService: ModalService,
    private inj: Injector,
  ) {}

  ngOnInit(): void {
    this.store
      .select(ProfileSelectors.getProfile)
      .pipe(
        filter((profile) => !!profile?.id),
        takeUntil(this.unsubscribe$),
      )
      .subscribe((profile) => {
        this.store.dispatch(
          MapMarkerActions.getMapMarkerByRoleId({ roleId: profile?.role }),
        );
      });


this.store.dispatch(ProductActions.getAllProduct());
  }

  updateMapMarkerVisibility(selectedMarker: any): void {
    this.store.dispatch(
      MapMarkerActions.updateMapMarkerVisibility({
        id: selectedMarker.id,
        isHidden: !selectedMarker.hidden,
      }),
    );
  }

  editMarker(selectedMarker: any) {
    console.log(selectedMarker);
    const injector: Injector = Injector.create({
      providers: [
        {
          provide: BaseMarker,
          useValue: new Marker(
            selectedMarker?.position?.lat,
            selectedMarker?.position?.lng,
            selectedMarker.hidden,
            selectedMarker.id,
            selectedMarker.productIds,
            selectedMarker?.label?.text,
            selectedMarker?.description,
          ),
        },
      ],
      parent: this.inj,
    });

    this.modalService.openNewModal(
      EditMapMarkerFormComponent,
      injector,
      'Редагувати нову',
    );
  }
  deleteMapMarker(selectedMarker: any): void {
```

```
    const injector: Injector = Injector.create({
      providers: [
        {
          provide: BaseMarker,
          useValue: new Marker(0, 0, false,
selectedMarker.id),
        },
      ],
      parent: this.inj,
    });

this.modalService.openNewModal(DeleteMapMarkerCompone
nt, injector);
  }

  ngOnDestroy(): void {
    this.unsubscribe$.complete();
    this.unsubscribe$.unsubscribe();
  }

  protected readonly TableTypeEnum = TableTypeEnum;
}
```

frontend/src/pages/admin-path/orders-page/orders-page.component.html

```
<section class="mx-4 mt-4">
  <nb-card>
    <div class="flex justify-between items-center p-
4">
      <h1 class="text-3xl font-medium text-gray-
700">Замовлення</h1>
    </div>
  </nb-card>
  <app-data-table
    [tableType]="tableTypeEnum.Products"
    [dataSource]="orders"
    [columns]="tableColumns"
    [buttonsTemplate]="buttonsTemplate"
  >
  </app-data-table>
</section>

<ng-template #buttonsTemplate let-
selectedOrderData="data">
  <div class="button-container">
    <app-button
      icon="edit-2-outline"
      buttonStatus="info"
      iconStatus="basic"
      size="medium"

(onClick)="updatedSelectedOrder(selectedOrderData)"
    ></app-button>
    <app-button
      icon="shopping-bag-outline"
      size="medium"
      buttonStatus="warning"
      iconStatus="basic"

(onClick)="openOrderItemDialog(selectedOrderData)"
    />
  </div>
</ng-template>
```

frontend/src/pages/admin-path/orders-page/orders-page.component.ts

```
import { Component, Injector, OnDestroy, OnInit }
from '@angular/core';
import { IOrderDetails } from
'../../../common/interfaces';
import {
  OrderStatusEnum,
  OrderStatusTitleEnum,
  TableTypeEnum,
} from '../../../common/enums';
import { Subject, takeUntil } from 'rxjs';
import { Store } from '@ngrx/store';
import { ModalService } from '../../../services';
import { OrderActions } from
'../../../store/actions';
import { BaseOrder, Order } from
'../../../common/classes';
import { OrderSelectors } from
'../../../store/selectors';
import { EditOrderComponent } from
'../../../components/forms/order/edit-order/edit-
order.component';
import { EditOrderItemsComponent } from
'../../../components/forms/order/edit-order-
items/edit-order-items.component';

@Component({
  selector: 'app-products-page',
  templateUrl: './orders-page.component.html',
  styleUrls: ['./orders-page.component.scss'],
})
export class OrdersPageComponent implements OnInit,
OnDestroy {
  orders: Array<any>;
  tableColumns = [
    { dataField: 'status', dataType: 'string' },
    { dataField: 'type', dataType: 'string' },
    { dataField: 'mapMarker', dataType: 'number' },
    { dataField: 'payment', dataType: 'string' },
    { dataField: 'total', dataType: 'number' },
    { dataField: 'user', dataType: 'string' },
    { dataField: 'created', dataType: 'date' },
  ];
  tableTypeEnum = TableTypeEnum;
```

```
  private readonly unsubscribe$ = new Subject();
  constructor(
    private readonly store: Store<any>,
    private readonly modalService: ModalService,
    private inj: Injector,
  ) {}

  ngOnInit() {

this.store.dispatch(OrderActions.getAllOrderDetails()
);


    this.store
      .select(OrderSelectors.getAllOrderDetails)
      .pipe(takeUntil(this.unsubscribe$))
      .subscribe((orders) => {
        this.orders =
this.mapOrderForDataTable(orders);
      });
  }

  mapOrderForDataTable(orders: Array<IOrderDetails>):
Array<any> {
    return (
      orders?.map((order) => {
        return {
          id: order.id,
          mapMarker: order.mapMarkerId,
          status: this.getStatusTitle(order.status),
          total: Number(order.total),
          payment: order.paymentDetails.provider,
          user: order?.user?.name,
          created: new Date(order.createdAt),
        };
      }) || []
    );
  }

  getStatusTitle(status: number): string {
    switch (status) {
      case OrderStatusEnum.Created:
        return OrderStatusTitleEnum.Created;
      case OrderStatusEnum.Canceled:
        return OrderStatusTitleEnum.Canceled;
      case OrderStatusEnum.InProcess:
        return OrderStatusTitleEnum.InProcess;
      case OrderStatusEnum.Completed:
        return OrderStatusTitleEnum.Completed;
      default:
        return OrderStatusTitleEnum.Created;
    }
  }

  updatedSelectedOrder(order: IOrderDetails): void {
    const injector: Injector = Injector.create({
      providers: [
        {
          provide: BaseOrder,
          useValue: new Order(order.status, order.id,
0, order?.type),
        },
      ],
      parent: this.inj,
    });
    this.modalService.openNewModal(
      EditOrderComponent,
      injector,
      'Змінити статус замовлення',
    );
  }

  openOrderItemDialog(order: any): void {
    const injector: Injector = Injector.create({
      providers: [
        {
          provide: BaseOrder,
          useValue: new Order(
            order.status,
            order.id,
            order.mapMarker,
            order?.type,
          ),
        },
      ],
      parent: this.inj,
    });
    this.modalService.openNewModal(
      EditOrderItemsComponent,
      injector,
      'Змінити склад замвлення',
    );
  }

  ngOnDestroy() {
    this.unsubscribe$.complete();
    this.unsubscribe$.unsubscribe();
  }
}
```

frontend/src/pages/admin-path/products-page/products-page.component.html

```
<section class="mx-4 mt-4">
  <nb-card>
    <div class="flex justify-between items-center p-
4">
      <h1 class="text-3xl font-medium text-gray-
```

```
700">Продукти</h1>
      <app-button
        icon="plus-outline"
        iconStatus="basic"
        size="medium"
        label="Додати саджанець"
        (onClick)="createNewProduct()"
      ></app-button>
    </div>
  </nb-card>

  <app-data-table
    [tableType]="tableTypeEnum.Products"
    [dataSource]="products"
    [columns]="tableColumns"
    [buttonsTemplate]="buttonsTemplate"
  >
  </app-data-table>
</section>

<ng-template #buttonsTemplate let-
selectedProductData="data">
  <div class="button-container">
    <app-button
      [icon]="selectedProductData?.hidden ? 'eye-
outline' : 'eye-off-outline'"
      size="medium"
      buttonStatus="warning"
      iconStatus="basic"

(onClick)="updateProductVisibility(selectedProductDat
a)"
    />
    <app-button
      icon="edit-2-outline"
      buttonStatus="info"
      iconStatus="basic"
      size="medium"

(onClick)="updatedSelectedProduct(selectedProductData
)"
    ></app-button>
    <app-button
      icon="trash-2-outline"
      buttonStatus="danger"
      iconStatus="basic"
      size="medium"

(onClick)="deleteSelectedUser(selectedProductData)"
    ></app-button>
  </div>
</ng-template>
```

frontend/src/pages/admin-path/products-page/products-page.component.ts

```
import { Component, Injector, OnDestroy, OnInit }
from '@angular/core';
import { IProduct } from
'../../../common/interfaces';
import { TableTypeEnum } from
'../../../common/enums';
import { Subject, takeUntil } from 'rxjs';
import { Store } from '@ngrx/store';
import { ModalService } from '../../../services';
import { ProductActions } from
'../../../store/actions';
import { BaseProduct, Product } from
'../../../common/classes';
import { AddProductComponent } from
'../../../components/forms/product/add-product/add-
product.component';
import { ProductSelectors } from
'../../../store/selectors';
import { DeleteProductComponent } from
'../../../components/delete-dialog/delete-
product.component';
import { EditProductComponent } from
'../../../components/forms/product/edit-product/edit-
product.component';

@Component({
  selector: 'app-products-page',
  templateUrl: './products-page.component.html',
  styleUrls: ['./products-page.component.scss'],
})
export class ProductsPageComponent implements OnInit,
OnDestroy {
  products: Array<IProduct>;
  tableColumns = [
    { dataField: 'image', dataType: 'picture' },
    { dataField: 'name', dataType: 'string' },
    { dataField: 'description', dataType: 'string' },
    { dataField: 'price', dataType: 'number' },
    { dataField: 'hidden', dataType: 'boolean' },
  ];
  tableTypeEnum = TableTypeEnum;

  private readonly unsubscribe$ = new Subject();
  constructor(
    private readonly store: Store<any>,
    private readonly modalService: ModalService,
    private inj: Injector,
  ) {}

  ngOnInit() {

this.store.dispatch(ProductActions.getAllProduct());

    this.store
      .select(ProductSelectors.getAllProducts)
      .pipe(takeUntil(this.unsubscribe$))
```

```
      .subscribe((products) => {
        this.products = products;
      });
  }

  createNewProduct(): void {
    const injector: Injector = Injector.create({
      providers: [
        {
          provide: BaseProduct,
          useValue: {} as Product,
        },
      ],
      parent: this.inj,
    });
    this.modalService.openNewModal(
      AddProductComponent,
      injector,
      'Створити новий продукт',
    );
  }

  updatedSelectedProduct(productData: IProduct): void
{
    const injector: Injector = Injector.create({
      providers: [
        {
          provide: BaseProduct,
          useValue: new Product(
            productData.id,
            productData.name,
            productData.image,
            productData.hidden,
            productData.price,
            productData?.description,
          ),
        },
      ],
      parent: this.inj,
    });
    this.modalService.openNewModal(
      EditProductComponent,
      injector,
      'Редагувати продукт',
    );
  }

  deleteSelectedUser(product: IProduct): void {
    const injector: Injector = Injector.create({
      providers: [
        {
          provide: BaseProduct,
          useValue: new Product(product.id,
product.name, '', false, 0),
        },
      ],
      parent: this.inj,
    });

this.modalService.openNewModal(DeleteProductComponent
, injector);
  }

  updateProductVisibility(productData: IProduct):
void {
    this.store.dispatch(
      ProductActions.updateProductVisibility({
        id: productData.id || 0,
        isHidden: !productData.hidden,
      }),
    );
  }

  ngOnDestroy() {
    this.unsubscribe$.complete();
    this.unsubscribe$.unsubscribe();
  }
}
```

frontend/src/pages/admin-path/users-page/users-page.component.html

```
<section class="mx-4 mt-4">
  <nb-card>
    <div class="flex justify-between items-center p-
4">
      <h1 class="text-3xl font-medium text-gray-
700">Користувачі</h1>
    </div>
  </nb-card>
  <app-data-table
    [tableType]="tableTypeEnum.Users"
    [dataSource]="users"
    [columns]="tableColumns"
    [buttonsTemplate]="buttonsTemplate"
  >
  </app-data-table>
</section>

<ng-template #buttonsTemplate let-
selectedUserData="data">
  <div class="button-container">
    <app-button
      icon="edit-2-outline"
      buttonStatus="info"
      iconStatus="basic"
      size="medium"
```

```
(onClick)="updatedSelectedUser(selectedUserData)"
    ></app-button>
    <app-button
      icon="shopping-cart-outline"
      size="medium"
      buttonStatus="danger"
      iconStatus="basic"
      (onClick)="cleanUserCart(selectedUserData)"
    />
    <app-button
      icon="trash-2-outline"
      buttonStatus="danger"
      iconStatus="basic"
      size="medium"

(onClick)="deleteSelectedUser(selectedUserData)"
    ></app-button>
  </div>
</ng-template>
```

frontend/src/pages/admin-path/users-page/users-page.component.ts

```
import { Component, Injector, OnDestroy, OnInit } from '@angular/core';
import { Store } from '@ngrx/store';
import { CartActions, UserActions } from '../../../store/actions';
import { UserSelectors } from '../../../store/selectors';
import { Subject, takeUntil } from 'rxjs';
import { IUser } from '../../../common/interfaces';
import { TableTypeEnum } from '../../../common/enums';
import { BaseUser, User } from '../../../common/classes';
import { ModalService } from '../../../services';
import { DeleteUserComponent } from '../../../components/delete-dialog/delete-user.component';
import { EditUserFormComponent } from '../../../components/forms';

@Component({
  selector: 'app-user-page',
  templateUrl: './users-page.component.html',
  styleUrls: ['./users-page.component.scss'],
})
export class UsersPageComponent implements OnInit, OnDestroy {
  users: Array<IUser>;
  tableColumns = [
    { dataField: 'name', dataType: 'string' },
    { dataField: 'email', dataType: 'string' },
    { dataField: 'role', dataType: 'string' },
  ];
  tableTypeEnum = TableTypeEnum;

  protected readonly console = console;

  private readonly unsubscribe$ = new Subject();
  constructor(
    private readonly store: Store<any>,
    private readonly modalService: ModalService,
    private inj: Injector,
  ) {}

  ngOnInit() {
    this.store.dispatch(UserActions.getAllUsers());
    this.store.dispatch(UserActions.getUserRoles());

    this.store
      .select(UserSelectors.getAllUsers)
      .pipe(takeUntil(this.unsubscribe$))
      .subscribe((users) => {
        this.users = users;
      });
  }

  updatedSelectedUser(userData: IUser): void {
    const injector: Injector = Injector.create({
      providers: [
        {
          provide: BaseUser,
          useValue: new User(
            userData.id,
            userData.subId,
            userData.email,
            userData.name,
            userData.role || '',
          ),
        },
      ],
      parent: this.inj,
    });
    this.modalService.openNewModal(
      EditUserFormComponent,
      injector,
      'Редагувати користувача',
    );
  }

  deleteSelectedUser(userData: IUser): void {
    const injector: Injector = Injector.create({
      providers: [
        {
          provide: BaseUser,
          useValue: new User(userData.id, '', '', userData.name),
```

```
        },
      ],
      parent: this.inj,
    });

this.modalService.openNewModal(DeleteUserComponent,
injector);
  }

  cleanUserCart(userData: IUser): void {
    this.store.dispatch(
      CartActions.deleteCartByUserId({ id:
userData.id || '' }),
    );
  }

  ngOnDestroy() {
    this.unsubscribe$.complete();
    this.unsubscribe$.unsubscribe();
  }
}
```

frontend/src/components/google-map/google-map.component.html

```
<google-map
  class="google-map"
  height="720px"
  width="100%"
  [center]="center"
  [options]="options"
  (mapDblclick)="mapType === mapTypeEnum.Admin ?
addMarker($event) : null"
>
  <map-marker
    #someMarker="mapMarker"
    *ngFor="let marker of markers"
    [position]="marker.position"
    [label]="marker.label"
    [title]="marker.title"
    [options]="marker.options"
    (mapClick)="openInfo(someMarker, marker)"
  >
  </map-marker>

  <map-info-window>
    <ng-container
      *ngTemplateOutlet="markerInfoWindowTemplate;
context: selectedMarker"
    ></ng-container>
  </map-info-window>
</google-map>
```

frontend/src/components/google-map/google-map.component.ts

```
import {
  Component,
  Injector,
  Input,
  OnDestroy,
  OnInit,
  ViewChild,
} from '@angular/core';
import { GoogleMap, MapInfoWindow } from
'@angular/google-maps';
import { Store } from '@ngrx/store';
import { MapMarkerSelectors } from
'../../store/selectors';
import { Subject, takeUntil } from 'rxjs';
import { IMapMarker } from '../../common/interfaces';
import { BaseMarker, Marker } from
'../../common/classes';
import { AddMapMarkerFormComponent } from '../forms';
import { ModalService } from '../../services';
import { MapTypeEnum } from '../../common/enums/map';

@Component({
  selector: 'app-google-map',
  templateUrl: './google-map.component.html',
  styleUrls: ['./google-map.component.scss'],
})
export class GoogleMapComponent implements OnInit,
OnDestroy {
  @Input() options: google.maps.MapOptions;
  @Input() markerInfoWindowTemplate: any;
  @Input() mapType: MapTypeEnum = MapTypeEnum.User;

  @ViewChild(GoogleMap, { static: false }) map:
GoogleMap;
  @ViewChild(MapInfoWindow, { static: false }) info:
MapInfoWindow;

  mapTypeEnum = MapTypeEnum;
  center: google.maps.LatLngLiteral;
  markers: any[] = [];
  selectedMarker = { $implicit: {} };

  private readonly unsubscribe$ = new Subject();
  constructor(
    private readonly store: Store<any>,
    private readonly modalService: ModalService,
    private inj: Injector,
  ) {}

  ngOnInit() {
    this.store
      .select(MapMarkerSelectors.getMapMarkers)
      .pipe(takeUntil(this.unsubscribe$))
      .subscribe((mapMarkers) => {
        this.markers = mapMarkers?.map((mapMarker:
```

```
IMapMarker) => ({
          id: mapMarker.id,
          hidden: mapMarker.hidden,
          productIds: mapMarker.productIds,
          description: mapMarker.description,
          position: {
            lat: mapMarker.lat,
            lng: mapMarker.lng,
          },
          label: {
            color: 'white',
            text: this.generateMarkerName(mapMarker),
          },
          options: {
            animation: google.maps.Animation.DROP,
          },
        }));
      });
  }

  generateMarkerName(marker: IMapMarker) {
    if (!marker?.name) {
      return `Мітку #${marker.id}`;
    }
    return marker.name;
  }

  addMarker(event: google.maps.MapMouseEvent) {
    const injector: Injector = Injector.create({
      providers: [
        {
          provide: BaseMarker,
          useValue: new Marker(
            event?.latLng?.lat() || 0,
            event?.latLng?.lng() || 0,
            false,
          ),
        },
      ],
      parent: this.inj,
    });

    this.modalService.openNewModal(
      AddMapMarkerFormComponent,
      injector,
      'Додати нову мітку',
    );
  }

  openInfo(marker: any, selectedMarker: any) {
    this.selectedMarker.$implicit = selectedMarker;
    this.info.open(marker);
  }
```

```
  ngOnDestroy() {
    this.unsubscribe$.complete();
    this.unsubscribe$.unsubscribe();
  }
}
```

frontend/src/components/modal/modal.component.html

```
<section class="modal-section">
  <div class="modal-container">
    <div *ngIf="modalState?.title?.length"
class="modal-header">
      <p class="modal-title">{{ modalState.title
}}</p>

      <app-button
        icon="close-outline"
        size="medium"
        iconStatus="info"
        [isGhost]="true"
        (onClick)="closeModal()"
      />
    </div>
    <div
      class="modal-body"
      [ngClass]="{ 'rounded-lg':
!modalState.title?.length }"
    >
      <ng-container
        *ngComponentOutlet="modalState.content;
injector: modalState.injector"
      >
      </ng-container>
    </div>
  </div>
</section>
```

frontend/src/components/modal/modal.component.ts

```
import { Component, OnInit } from '@angular/core';
import { ModalService } from '../../services';
import { IModalDto } from '../../common/interfaces';

@Component({
  selector: 'app-modal',
  templateUrl: './modal.component.html',
  styleUrls: ['./modal.component.scss'],
})
export class ModalComponent implements OnInit {
  modalState!: IModalDto;
  constructor(private readonly modalService:
ModalService) {}
```

```
  ngOnInit(): void {
    this.modalService.modalCurrentContent.subscribe(
      (value) => (this.modalState = value),
    );
  }

  public closeModal(): void {
    this.modalService.changeOpenState(false);
  }
}
```

frontend/src/components/navbar/navbar.component.html

```
<header>
  <nav class="navbar">
    <div class="flex flex-wrap justify-between items-center mx-auto">
      <a href="/" class="flex items-center">
        <img
          src="assets/logo/Floresta_small_resolution.png"
          class="mr-3 h-10"
          alt="Floresta Logo"
        />
      </a>

      <div
        class="justify-between w-full md:flex md:w-auto order-1 md:order-none"
        id="mobile-menu-2"
      >
        <ul
          [ngClass]="{ hidden: !isSubMenuOpen && isTabletView }"
          class="flex flex-col items-center mt-4 font-medium md:flex-row md:space-x-4 md:mt-0"
        >
          <li *ngFor="let navigationItem of navigationMenuItems">
            <app-navigation-item [data]="navigationItem"> </app-navigation-item>
          </li>
        </ul>
      </div>

      <div class="flex items-center justify-end gap-2 lg:order-2 md:w-44">
        <ng-container *ngIf="isUserAuthenticated; else LoginButton">
          <app-button
            nbContextMenuTag="user-context-menu"
            icon="person-outline"
            size="medium"
            iconStatus="info"
            [isGhost]="true"
            [nbContextMenu]="contextMenuList"
          />

          <app-button
            class="md:hidden"
            [icon]="isSubMenuOpen ? 'close-outline' : 'menu-outline'"
            size="medium"
            iconStatus="info"
            [isGhost]="true"
            (onClick)="isSubMenuOpen = !isSubMenuOpen"
          />
        </ng-container>
      </div>
    </div>
  </nav>
</header>

<ng-template #LoginButton>
  <app-button
    icon="log-in-outline"
    iconStatus="info"
    size="medium"
    [isGhost]="true"
    (onClick)="authWrapperService.logIn()"
  >
  </app-button>
</ng-template>
```

frontend/src/components/navbar/navbar.component.ts

```
import { Component, HostListener, OnDestroy, OnInit } from '@angular/core';
import { INavigationItem } from '../../common/interfaces';
import { AuthWrapperService, LocalStorageService } from '../../services';
import { Subject, takeUntil } from 'rxjs';
import { Router } from '@angular/router';
import { USER_PROFILE } from '../../common/local-storage-keys';
import { RouterPathEnum, UserRolesEnum } from '../../common/enums';

@Component({
  selector: 'app-navbar',
  templateUrl: './navbar.component.html',
  styleUrls: ['./navbar.component.scss'],
})
export class NavbarComponent implements OnInit, OnDestroy {
  isUserAuthenticated = false;
  isSubMenuOpen = false;
  isTabletView = false;
```

```
  navigationMenuItems: Array<INavigationItem> = [
    {
      label: 'Посадити дерева',
      link: `/${RouterPathEnum.Home}`,
      fragment: 'google-map',
      asButton: true,
    },
    {
      label: 'Доступні міста',
      link: `/${RouterPathEnum.Home}`,
      fragment: 'google-map',
      asButton: false,
    },
    {
      label: 'FAQ',
      link: `/${RouterPathEnum.Home}`,
      fragment: 'FAQ',
      asButton: false,
    },
  ];

  contextMenuList: Array<any>;

  private readonly unsubscribe$ = new Subject();

  constructor(
    public authWrapperService: AuthWrapperService,
    private readonly router: Router,
    private readonly localStorageService:
LocalStorageService,
  ) {}

  ngOnInit() {
    this.isTabletView = this.isTabletViewCalc;
    this.authWrapperService.isAuthenticated$
      .pipe(takeUntil(this.unsubscribe$))
      .subscribe((isAuthenticated) => {
        this.isUserAuthenticated = isAuthenticated;
        this.setContextMenuList();
      });
  }

  setContextMenuList(): void {
    const userRole = JSON.parse(
      this.localStorageService.getItem(USER_PROFILE)
|| {},
    )?.role;

    this.contextMenuList =
      userRole === UserRolesEnum.Admin
        ? [
            { title: 'Мій профіль', link:
RouterPathEnum.Profile },
            { title: 'Адмін панель', link:
RouterPathEnum.Admin },
            { title: 'Вихід', link:
RouterPathEnum.Logout },
          ]
        : [
            { title: 'Мій профіль', link:
RouterPathEnum.Profile },
            { title: 'Вихід', link:
RouterPathEnum.Logout },
          ];
  }

  @HostListener('window:resize', ['$event'])
  onResize(_: any) {
    this.isTabletView = this.isTabletViewCalc;
  }

  @HostListener('window:scroll', ['$event'])
  onWindowScroll() {
    let element = document.querySelector('.navbar')
as HTMLElement;
    if (window.pageYOffset > element.clientHeight) {
      element.classList.add('navbar-inverse');
    } else {
      element.classList.remove('navbar-inverse');
    }
  }

  get isTabletViewCalc(): boolean {
    return window.innerWidth <= 1024;
  }

  ngOnDestroy() {
    this.unsubscribe$.complete();
    this.unsubscribe$.unsubscribe();
  }
}
```

frontend/src/services/map-market.service.ts

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { IMapMarker, IMapMarkerWithProductInfo } from
'../common/interfaces';
import { ApiSidePathEnum, RootApiPathEnum } from
'../common/enums';
import { Observable } from 'rxjs';
import { AUTH_TOKEN } from '../common/local-storage-
keys';
import { LocalStorageService } from './local-
storage.service';

@Injectable({
  providedIn: 'root',
```

```
})
export class MapMarketService {
  constructor(
    private http: HttpClient,
    private readonly localStorageService:
LocalStorageService,
  ) {}

  public getMapMarkerByRoleId(roleId: number):
Observable<Array<IMapMarker>> {
    const token =
this.localStorageService.getItem(AUTH_TOKEN);

    return this.http.get<Array<IMapMarker>>(
`http://localhost:3001${RootApiPathEnum.Api}${RootApi
PathEnum.Marker}?roleId=${roleId}`,
      {
        headers: {
          token: token || '',
        },
      },
    );
  }

  public getMarkerProducts(id: number):
Observable<IMapMarkerWithProductInfo> {
    const token =
this.localStorageService.getItem(AUTH_TOKEN);

    return this.http.get<IMapMarkerWithProductInfo>(
`http://localhost:3001${RootApiPathEnum.Api}${RootApi
PathEnum.Marker}${ApiSidePathEnum.GetProducts}?id=${i
d}`,
      {
        headers: {
          token: token || '',
        },
      },
    );
  }

  public createMapMarker(body: IMapMarker):
Observable<IMapMarker> {
    const token =
this.localStorageService.getItem(AUTH_TOKEN);

    return this.http.post<IMapMarker>(
`http://localhost:3001${RootApiPathEnum.Api}${RootApi
PathEnum.Marker}`,
      body,
      {
        headers: {
          token: token || '',
        },
      },
    );
  }

  public updateMapMarker(id: number, body:
IMapMarker): Observable<IMapMarker> {
    const token =
this.localStorageService.getItem(AUTH_TOKEN);

    return this.http.patch<IMapMarker>(
`http://localhost:3001${RootApiPathEnum.Api}${RootApi
PathEnum.Marker}/${id}`,
      body,
      {
        headers: {
          token: token || '',
        },
      },
    );
  }

  public updateMapMarkerVisibility(
    id: number,
    newVisibility: boolean,
  ): Observable<IMapMarker> {
    const token =
this.localStorageService.getItem(AUTH_TOKEN);

    return this.http.patch<IMapMarker>(
`http://localhost:3001${RootApiPathEnum.Api}${RootApi
PathEnum.Marker}/${id}`,
      { hidden: newVisibility },
      {
        headers: {
          token,
        },
      },
    );
  }

  public deleteMapMarkerById(id: number):
Observable<void> {
    const token =
this.localStorageService.getItem(AUTH_TOKEN);

    return this.http.delete<void>(
`http://localhost:3001${RootApiPathEnum.Api}${RootApi
PathEnum.Marker}/${id}`,
      {
        headers: {
          token,
        },
      },
```

```
    );
  }
}
```

frontend/src/services/modal-service.service.ts

```
import { Injectable, Injector } from '@angular/core';
import { BehaviorSubject } from 'rxjs';

@Injectable({
  providedIn: 'root',
})
export class ModalService {
  modalSource = new BehaviorSubject({
    title: '',
    open: false,
    content: null,
    injector: Injector,
  });
  modalCurrentContent =
this.modalSource.asObservable();

  get isOpen(): boolean {
    return this.modalSource.value.open;
  }

  public changeModalContent(newContent: any) {
    this.modalSource.next({
...this.modalSource.value, content: newContent });
  }
  public changeOpenState(newState: boolean) {
    this.modalSource.next({
...this.modalSource.value, open: newState });
  }

  public closeModal() {
    this.modalSource.next({
...this.modalSource.value, open: false });
  }

  public changeInjector(newInjector: any) {
    this.modalSource.next({
      ...this.modalSource.value,
      injector: newInjector,
    });
  }
  public openNewModal(
    newContent: any,
    newInjector: any = null,
    newTitle: string = '',
  ) {
    this.modalSource.next({
      title: newTitle,
      open: true,
      content: newContent,
      injector: newInjector,
    });
  }
}
```

frontend/src/services/user.service.ts

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { IUser, IUserRoles } from
'../common/interfaces';
import { ApiSidePathEnum, RootApiPathEnum } from
'../common/enums';
import { Observable } from 'rxjs';
import { AUTH_TOKEN } from '../common/local-storage-
keys';
import { LocalStorageService } from './local-
storage.service';

@Injectable({
  providedIn: 'root',
})
export class UserService {
  constructor(
    private http: HttpClient,
    private readonly localStorageService:
LocalStorageService,
  ) {}

  public userTokenToProps(token: any): IUser {
    return {
      name: token.nickname,
      email: token.email,
      subId: token?.['sub'],
    };
  }

  public getUserById(id: string): Observable<IUser> {
    const token =
this.localStorageService.getItem(AUTH_TOKEN);

    return this.http.get<IUser>(

`http://localhost:3001${RootApiPathEnum.Api}${RootApi
PathEnum.User}/${id}`,
      {
        headers: {
          token: token || '',
        },
      },
    );
  }

  public getAllUsers(): Observable<Array<IUser>> {
```

```
    const token =
this.localStorageService.getItem(AUTH_TOKEN);

    return this.http.get<Array<IUser>>(
`http://localhost:3001${RootApiPathEnum.Api}${RootApi
PathEnum.User}${ApiSidePathEnum.All}`,
      {
        headers: {
          token: token || '',
        },
      },
    );
  }

  public getUserRoles():
Observable<Array<IUserRoles>> {
    const token =
this.localStorageService.getItem(AUTH_TOKEN);

    return this.http.get<Array<IUserRoles>>(
`http://localhost:3001${RootApiPathEnum.Api}${RootApi
PathEnum.Role}${ApiSidePathEnum.All}`,
      {
        headers: {
          token: token || '',
        },
      },
    );
  }

  public createUser(body: any): Observable<IUser> {
    const token =
this.localStorageService.getItem(AUTH_TOKEN);

    return this.http.post<IUser>(
`http://localhost:3001${RootApiPathEnum.Api}${RootApi
PathEnum.User}`,
      body,
      {
        headers: {
          token: token || '',
        },
      },
    );
  }

  public updateUser(id: string, body: IUser):
Observable<IUser> {
    const token =
this.localStorageService.getItem(AUTH_TOKEN);

    return this.http.patch<IUser>(
`http://localhost:3001${RootApiPathEnum.Api}${RootApi
PathEnum.User}/${id}`,
      body,
      {
        headers: {
          token: token || '',
        },
      },
    );
  }

  public deleteUserById(id: string): Observable<void>
{
    const token =
this.localStorageService.getItem(AUTH_TOKEN);

    return this.http.delete<void>(
`http://localhost:3001${RootApiPathEnum.Api}${RootApi
PathEnum.User}/${id}`,
      {
        headers: {
          token: token || '',
        },
      },
    );
  }
}
```

frontend/src/store/reducers/map-marker.reducer.ts

```
import { createReducer, on } from '@ngrx/store';
import { MapMarkerActions } from '../actions';
import { IMapMarker } from '../../common/interfaces';

type mapMarkersState = {
  mapMarkers: Array<IMapMarker>;
};
export const mapMarkersState: mapMarkersState = {
  mapMarkers: [],
};

const mapMarkersActionReducer = createReducer(
  mapMarkersState,
  on(MapMarkerActions.getMapMarkerByRoleIdSuccess,
(state, action) => ({
    ...state,
    mapMarkers: action.mapMarkers,
  })),
  on(MapMarkerActions.deleteMapMarkerSuccess, (state,
action) => {
    const mapMarkers = [...state.mapMarkers];
    const indexToDelete = mapMarkers.findIndex(
      (element) => element.id === action.id,
    );
```

```
    if (indexToDelete !== -1) {
      mapMarkers.splice(indexToDelete, 1);
    }

    return { ...state, mapMarkers };
  }),

on(MapMarkerActions.updateMapMarkerVisibilitySuccess,
(state, action) => {
    let mapMarkers = [...state.mapMarkers];

    return {
      ...state,
      mapMarkers: mapMarkers.map((element) =>
        element.id !== action.id
          ? element
          : {
              ...element,
              hidden: action.isHidden,
            },
      ),
    };
  }),
);

export const mapMarkerReducer = { mapMarkers:
mapMarkersActionReducer };
```

frontend/src/store/reducers/order.reducer.ts

```
import { createReducer, on } from '@ngrx/store';
import { OrderActions } from '../actions';
import { IOrderDetails } from
'../../common/interfaces';

type orderState = {
  orderDetails: IOrderDetails | null;
  allOrderDetails: Array<IOrderDetails> | null;
};
export const orderInitialState: orderState = {
  orderDetails: null,
  allOrderDetails: null,
};

const orderActionReducer = createReducer(
  orderInitialState,
  on(OrderActions.getOrderDetailsSuccess, (state,
action) => ({
    ...state,
    orderDetails: action.orderDetails,
  })),
  on(OrderActions.getAllOrderDetailsSuccess, (state,
action) => ({
    ...state,
    allOrderDetails: action.allOrderDetails,
  })),
  on(OrderActions.updateOrderItemSuccess, (state,
action) => {
    const orderDetailsArray =
[...(state?.allOrderDetails || [])];
    const orderDetails = orderDetailsArray.find(
      (order) => order.id === action.orderId,
    );
    const arrayWithoutCurrentOrder =
orderDetailsArray.filter(
      (order) => order.id !== action.orderId,
    );

    if (orderDetails?.id) {
      return {
        ...state,
        allOrderDetails: [
          ...arrayWithoutCurrentOrder,
          {
            ...orderDetails,
            items: orderDetails?.items.map((element)
=>
              element.id !== action.id
                ? element
                : {
                    ...element,
                    quantity: action.quantity,
                  },
            ),
          },
        ] as Array<IOrderDetails>,
      };
    }
    return state;
  }),

  on(OrderActions.deleteOrderItemByIdSuccess, (state,
action) => {
    const orderDetailsArray =
[...(state?.allOrderDetails || [])];
    const orderDetails = orderDetailsArray.find(
      (order) => order.id === action.orderId,
    );
    const arrayWithoutCurrentOrder =
orderDetailsArray.filter(
      (order) => order.id !== action.orderId,
    );

    if (orderDetails?.id) {
      return {
        ...state,
        allOrderDetails: [
          ...arrayWithoutCurrentOrder,
          {
            ...orderDetails,
```

```
            items: orderDetails?.items.filter(
              (element) => element.id !== action.id,
            ),
          },
        ] as Array<IOrderDetails>,
      };
    }
    return state;
  }),
);

export const orderReducer = { order:
orderActionReducer };
```

frontend/src/store/reducers/product.reducer.ts

```
import { createReducer, on } from '@ngrx/store';
import { IProduct } from '../../common/interfaces';
import { ProductActions } from '../actions';

type productState = {
  allProducts: Array<IProduct>;
  productsCatalog: Array<IProduct>;
};
export const productInitialState: productState = {
  allProducts: [],
  productsCatalog: [],
};

const productActionReducer = createReducer(
  productInitialState,
  on(ProductActions.getAllProductsSuccess, (state,
action) => ({
    ...state,
    allProducts: action.products,
  })),
  on(ProductActions.updateProductSuccess, (state,
action) => {
    let products = [...state.allProducts];

    return {
      ...state,
      allProducts: products.map((element) =>
        element.id !== action.product?.id ? element :
action.product,
      ),
    };
  }),
  on(ProductActions.deleteProductByIdSuccess, (state,
action) => {
    const products = [...state.allProducts];
    const indexToDelete = products.findIndex(
      (element) => element.id === action.id,
    );
    if (indexToDelete !== -1) {
      products.splice(indexToDelete, 1);
    }

    return { ...state, allProducts: products };
  }),
  on(ProductActions.updateProductVisibility, (state,
action) => {
    let allProducts = [...state.allProducts];

    return {
      ...state,
      allProducts: allProducts.map((element) =>
        element.id !== action.id
          ? element
          : {
              ...element,
              hidden: action.isHidden,
            },
      ),
    };
  }),
  on(ProductActions.getCatalogSuccess, (state,
action) => ({
    ...state,
    productsCatalog: action.productsCatalog,
  })),
);

export const productReducer = { product:
productActionReducer };
```

frontend/src/store/reducers/user.reducer.ts

```
import { createReducer, on } from '@ngrx/store';
import { IUser, IUserRoles } from
'../../common/interfaces';
import { UserActions } from '../actions';

type userState = {
  users: Array<IUser>;
  roles: Array<IUserRoles>;
};
export const userInitialState: userState = {
  users: [],
  roles: [],
};

const userActionReducer = createReducer(
  userInitialState,
  on(UserActions.getAllUsersSuccess, (state, action)
=> ({
    ...state,
```

```
    users: action.users,
  })),
  on(UserActions.deleteUserByIdSuccess, (state,
action) => {
    const users = [...state.users];
    const indexToDelete = users.findIndex(
      (element) => element.id === action.id,
    );

    if (indexToDelete !== -1) {
      users.splice(indexToDelete, 1);
    }

    return { ...state, users };
  }),
  on(UserActions.getUserRolesSuccess, (state, action)
=> ({
    ...state,
    roles: action.userRoles,
  })),

  on(UserActions.updateUserSuccess, (state, action)
=> {
    let users = [...state.users];

    return {
      ...state,
      users: users.map((element) =>
        element.id !== action.user.id ? element :
action.user,
      ),
    };
  }),
);

export const userReducer = { user: userActionReducer
};
```

frontend/src/store/effects/map-marker.effect.ts

```
import { Injectable } from '@angular/core';
import { Actions, createEffect, ofType } from
'@ngrx/effects';
import { catchError, map, mergeMap, tap } from
'rxjs/operators';
import {
  LocalStorageService,
  MapMarketService,
  ModalService,
} from '../../services';
import { MapMarkerActions } from '../actions';
import { of } from 'rxjs';
import { USER_PROFILE } from '../../common/local-
storage-keys';
import { Store } from '@ngrx/store';

@Injectable()
export class MapMarkersEffects {
  constructor(
    private readonly store: Store<any>,
    private readonly actions$: Actions,
    private readonly mapMarketService:
MapMarketService,
    private readonly modalService: ModalService,
    private readonly localStorageService:
LocalStorageService,
  ) {}

  getMapMarkersByRoleIdId$ = createEffect(() =>
    this.actions$.pipe(
      ofType(MapMarkerActions.getMapMarkerByRoleId),
      mergeMap((action) =>

this.mapMarketService.getMapMarkerByRoleId(action.rol
eId).pipe(
          map((mapMarkers) =>

MapMarkerActions.getMapMarkerByRoleIdSuccess({
mapMarkers }),
          ),
          catchError(() =>
of(MapMarkerActions.getMapMarkerByRoleIdFailed())),
        ),
      ),
    ),
  );

  createMapMarker$ = createEffect(() =>
    this.actions$.pipe(
      ofType(MapMarkerActions.createMapMarker),
      mergeMap(({ mapMarker }) =>

this.mapMarketService.createMapMarker(mapMarker).pipe
(
          map((mapMarker) =>
            MapMarkerActions.createMapMarkerSuccess({
mapMarker }),
          ),
          tap(() => {
            const role = JSON.parse(

this.localStorageService.getItem(USER_PROFILE) || {},
            )?.role;

            if (this.modalService?.isOpen) {

this.modalService.changeOpenState(false);
            }

            this.store.dispatch(
              MapMarkerActions.getMapMarkerByRoleId({
roleId: role }),
```

```
            );
          }),
          catchError(() =>
of(MapMarkerActions.createMapMarkerFailed())),
        ),
      ),
    ),
  );

  updateUser$ = createEffect(() =>
    this.actions$.pipe(
      ofType(MapMarkerActions.updateMarker),
      mergeMap((action) =>
        this.mapMarketService
          .updateMapMarker(action?.id,
action?.mapMarker)
          .pipe(
            map((marker) => {
              if (this.modalService?.isOpen) {

this.modalService.changeOpenState(false);
              }

              return
MapMarkerActions.updateMarkerSuccess({
                mapMarker: marker,
              });
            }),
            tap(() => {
              const role = JSON.parse(

this.localStorageService.getItem(USER_PROFILE) || {},
              )?.role;

              this.store.dispatch(

MapMarkerActions.getMapMarkerByRoleId({ roleId: role
}),
              );
            }),
            catchError(() =>
of(MapMarkerActions.updateMarkerFailed())),
          ),
      ),
    ),
  );

  updateMapMarkerVisibility$ = createEffect(() =>
    this.actions$.pipe(

ofType(MapMarkerActions.updateMapMarkerVisibility),
      mergeMap(({ id, isHidden }) =>

this.mapMarketService.updateMapMarkerVisibility(id,
isHidden).pipe(
          map(() =>

MapMarkerActions.updateMapMarkerVisibilitySuccess({
id, isHidden }),
          ),
          catchError(() =>

of(MapMarkerActions.updateMapMarkerVisibilityFailed()
),
          ),
        ),
      ),
    ),
  );

  deleteMapMarker$ = createEffect(() =>
    this.actions$.pipe(
      ofType(MapMarkerActions.deleteMapMarker),
      mergeMap(({ id }) =>

this.mapMarketService.deleteMapMarkerById(id).pipe(
          map(() => {
            if (this.modalService?.isOpen) {

this.modalService.changeOpenState(false);
            }

            return
MapMarkerActions.deleteMapMarkerSuccess({ id });
          }),
          catchError(() =>
of(MapMarkerActions.deleteMapMarkerFailed())),
        ),
      ),
    ),
  );
}
```

frontend/src/store/effects/order.effect.ts

```
import { Injectable } from '@angular/core';
import { Actions, createEffect, ofType } from
'@ngrx/effects';
import { catchError, map, mergeMap, tap } from
'rxjs/operators';
import { ModalService, OrderService } from
'../../services';
import { CartActions, OrderActions } from
'../actions';
import { of } from 'rxjs';
import { Router } from '@angular/router';
import { RouterPathEnum } from '../../common/enums';
import { Store } from '@ngrx/store';

@Injectable()
export class OrderEffects {
  constructor(
```

```
    private actions$: Actions,
    private orderService: OrderService,
    private readonly store: Store<any>,
    private readonly router: Router,
    private readonly modalService: ModalService,
  ) {}

  createOrder$ = createEffect(() =>
    this.actions$.pipe(
      ofType(OrderActions.createOrder),
      mergeMap((action) =>

this.orderService.createOrder(action.orderDto).pipe(
          tap(({ id }) => {

this.router.navigate([RouterPathEnum.ThankYou], {
              queryParams: { orderNumber: id },
            });
          }),
          map(() =>
OrderActions.createOrderSuccess()),
          catchError(() =>
of(OrderActions.createOrderFailed())),
        ),
      ),
    ),
  );

  getOrderDetails$ = createEffect(() =>
    this.actions$.pipe(
      ofType(OrderActions.getOrderDetails),
      mergeMap((action) =>

this.orderService.getOrderDetails(action.orderNumber)
.pipe(
          map((order) =>
            OrderActions.getOrderDetailsSuccess({
orderDetails: order }),
          ),
          catchError(() =>
of(OrderActions.getOrderDetailsFailed())),
        ),
      ),
    ),
  );

  getAllOrderDetails = createEffect(() =>
    this.actions$.pipe(
      ofType(OrderActions.getAllOrderDetails),
      mergeMap((action) =>
        this.orderService.getAllOrders().pipe(
          map((orders) =>
            OrderActions.getAllOrderDetailsSuccess({
allOrderDetails: orders }),
          ),
          catchError(() =>
of(OrderActions.getAllOrderDetailsFailed())),
        ),
      ),
    ),
  );

  updateOrderStatus$ = createEffect(() =>
    this.actions$.pipe(
      ofType(OrderActions.updateOrderStatus),
      mergeMap((action) =>
        this.orderService
          .updateOrderStatus(action?.id,
`${action?.status}`)
          .pipe(
            map(() => {
              if (this.modalService?.isOpen) {

this.modalService.changeOpenState(false);
              }

              return
OrderActions.updateOrderStatusSuccess({
                id: action?.id,
                status: action.status,
              });
            }),
            catchError(() =>
of(OrderActions.updateOrderStatusFailed())),
          ),
      ),
    ),
  );

  createOrderItem$ = createEffect(() =>
    this.actions$.pipe(
      ofType(OrderActions.createOrderItem),
      mergeMap((action) =>
        this.orderService
          .createOrderItem({
            productId: action.productId,
            orderId: action.orderId,
            quantity: action.quantity || 1,
          })
          .pipe(
            map(() => {
              if (this.modalService?.isOpen) {

this.modalService.changeOpenState(false);
              }

              return
OrderActions.createOrderItemSuccess();
            }),
            tap(() => {
```

```
this.store.dispatch(OrderActions.getAllOrderDetails()
);
            }),
            catchError(() =>
of(OrderActions.createOrderItemFailed())),
          ),
      ),
    ),
  );

  updateOrderItem$ = createEffect(() =>
    this.actions$.pipe(
      ofType(OrderActions.updateOrderItem),
      mergeMap((action) =>
        this.orderService.updateOrderItem(action.id,
action.quantity).pipe(
          map(() => {
            if (this.modalService?.isOpen) {

this.modalService.changeOpenState(false);
            }

            return
OrderActions.updateOrderItemSuccess({
              id: action.id,
              quantity: action.quantity,
              orderId: action.orderId,
            });
          }),
          catchError(() =>
of(OrderActions.updateOrderItemFailed())),
        ),
      ),
    ),
  );

  deleteOrderItem$ = createEffect(() =>
    this.actions$.pipe(
      ofType(OrderActions.deleteOrderItemById),
      mergeMap((action) =>

this.orderService.deleteOrderItem(action.id).pipe(
          map(() => {
            if (this.modalService?.isOpen) {

this.modalService.changeOpenState(false);
            }

            return
OrderActions.deleteOrderItemByIdSuccess({
              id: action.id,
              orderId: action.orderId,
            });
          }),
          catchError(() =>
of(CartActions.deleteCartItemByIdFailed())),
        ),
      ),
    ),
  );
}
```

frontend/src/store/effects/product.effects.ts

```
import { Injectable } from '@angular/core';
import { Actions, createEffect, ofType } from
'@ngrx/effects';
import { catchError, map, mergeMap, tap } from
'rxjs/operators';
import { MapMarketService, ModalService,
ProductService } from '../../services';
import { ProductActions } from '../actions';
import { of } from 'rxjs';
import { Store } from '@ngrx/store';

@Injectable()
export class ProductEffects {
  constructor(
    private readonly store: Store<any>,
    private actions$: Actions,
    private readonly productService: ProductService,
    private readonly modalService: ModalService,
    private readonly mapMarketService:
MapMarketService,
  ) {}

  getAllProducts$ = createEffect(() =>
    this.actions$.pipe(
      ofType(ProductActions.getAllProduct),
      mergeMap((action) =>
        this.productService.getAllProducts().pipe(
          map((products) =>
ProductActions.getAllProductsSuccess({ products })),
          catchError(() =>
of(ProductActions.getAllProductsFailed())),
        ),
      ),
    ),
  );

  createProduct$ = createEffect(() =>
    this.actions$.pipe(
      ofType(ProductActions.createProduct),
      mergeMap(({ product }) =>

this.productService.createProduct(product).pipe(
          map((newProduct) => {
            if (this.modalService?.isOpen) {

this.modalService.changeOpenState(false);
```

```
            }
            return
ProductActions.createProductSuccess({ product:
newProduct });
          }),
          tap(() => {

this.store.dispatch(ProductActions.getAllProduct());
          }),
          catchError(() =>
of(ProductActions.createProductFailed())),
        ),
      ),
    ),
  );

  updateProductVisibility$ = createEffect(() =>
    this.actions$.pipe(
      ofType(ProductActions.updateProductVisibility),
      mergeMap(({ id, isHidden }) =>

this.productService.updateProductVisibility(id,
isHidden).pipe(
          map(() => {
            if (this.modalService?.isOpen) {

this.modalService.changeOpenState(false);
            }
            return
ProductActions.updateProductVisibilitySuccess({
              id,
              isHidden,
            });
          }),
          catchError(() =>
of(ProductActions.updateProductVisibilityFailed())),
        ),
      ),
    ),
  );

  updateProduct$ = createEffect(() =>
    this.actions$.pipe(
      ofType(ProductActions.updateProduct),
      mergeMap((action) =>

this.productService.updateProductById(action?.id,
action?.product).pipe(
          map((product) => {
            if (this.modalService?.isOpen) {

this.modalService.changeOpenState(false);
            }
            return
ProductActions.updateProductSuccess({ product });
          }),
          tap(() => {

this.store.dispatch(ProductActions.getAllProduct());
          }),
          catchError(() =>
of(ProductActions.updateProductFailed())),
        ),
      ),
    ),
  );

  deleteProductById$ = createEffect(() =>
    this.actions$.pipe(
      ofType(ProductActions.deleteProductById),
      mergeMap(({ id }) =>

this.productService.deleteProductById(id).pipe(
          map(() => {
            if (this.modalService?.isOpen) {

this.modalService.changeOpenState(false);
            }

            return
ProductActions.deleteProductByIdSuccess({ id });
          }),
          catchError(() =>
of(ProductActions.deleteProductByIdFailed())),
        ),
      ),
    ),
  );

  getProductsCatalog$ = createEffect(() =>
    this.actions$.pipe(
      ofType(ProductActions.getCatalog),
      mergeMap(({ id }) =>

this.mapMarketService.getMarkerProducts(id).pipe(
          map((markerWithProductInfo) =>
            ProductActions.getCatalogSuccess({
              productsCatalog:
markerWithProductInfo.products,
            }),
          ),
          catchError(() =>
of(ProductActions.getCatalogFailed())),
        ),
      ),
    ),
  );
}
```

frontend/src/store/effects/user.effects.ts

```
import { Injectable } from '@angular/core';
```

```
import { Actions, createEffect, ofType } from
'@ngrx/effects';
import { catchError, map, mergeMap, tap } from
'rxjs/operators';
import { ModalService, UserService } from
'../../services';
import { UserActions } from '../actions';
import { of } from 'rxjs';
import { Store } from '@ngrx/store';


@Injectable()
export class UserEffects {
  constructor(
    private readonly store: Store<any>,
    private actions$: Actions,
    private userService: UserService,
    private readonly modalService: ModalService,
  ) {}


  getAllUsers$ = createEffect(() =>
    this.actions$.pipe(
      ofType(UserActions.getAllUsers),
      mergeMap((action) =>
        this.userService.getAllUsers().pipe(
          map((users) =>
UserActions.getAllUsersSuccess({ users })),
          catchError(() =>
of(UserActions.getAllUsersFailed())),
        ),
      ),
    ),
  );


  getUserRoles$ = createEffect(() =>
    this.actions$.pipe(
      ofType(UserActions.getUserRoles),
      mergeMap((action) =>
        this.userService.getUserRoles().pipe(
          map((userRoles) =>
UserActions.getUserRolesSuccess({ userRoles })),
          catchError(() =>
of(UserActions.getUserRolesUsersFailed())),
        ),
      ),
    ),
  );


  updateUser$ = createEffect(() =>
    this.actions$.pipe(
      ofType(UserActions.updateUser),
      mergeMap((action) =>
        this.userService.updateUser(action?.id,
action?.user).pipe(
          map((user) => {
            if (this.modalService?.isOpen) {
this.modalService.changeOpenState(false);
            }


            return UserActions.updateUserSuccess({
user });
          }),
          tap(() => {

this.store.dispatch(UserActions.getAllUsers());
          }),
          catchError(() =>
of(UserActions.updateUserFailed())),
        ),
      ),
    ),
  );


  deleteUserById$ = createEffect(() =>
    this.actions$.pipe(
      ofType(UserActions.deleteUserById),
      mergeMap(({ id }) =>
        this.userService.deleteUserById(id).pipe(
          map(() => {
            if (this.modalService?.isOpen) {

this.modalService.changeOpenState(false);
            }


            return
UserActions.deleteUserByIdSuccess({ id });
          }),
          catchError(() =>
of(UserActions.deleteUserByIdFailed())),
        ),
      ),
    ),
  );
}



frontend/src/store/actions/map-marker.action.ts
import { createAction, props } from '@ngrx/store';
import { IMapMarker } from '../../common/interfaces';

export const getMapMarkerByRoleId = createAction(
  '[Map Marker] get Map Markers by role id',
  props<{ roleId: number }>(),
);
export const getMapMarkerByRoleIdSuccess =
createAction(
  '[Map Marker] get Map Markers by role id Success',
  props<{ mapMarkers: Array<IMapMarker> }>(),
);
export const getMapMarkerByRoleIdFailed =
```

```
createAction(
  '[Map Marker] get Map Markers by role id Failed',
);

export const createMapMarker = createAction(
  '[Map Marker] create Map Marker',
  props<{ mapMarker: IMapMarker }>(),
);
export const createMapMarkerSuccess = createAction(
  '[Map Marker] create Map Marker Success',
  props<{ mapMarker: IMapMarker }>(),
);
export const createMapMarkerFailed = createAction(
  '[Map Marker] create Map Marker Failed',
);

export const deleteMapMarker = createAction(
  '[Map Marker] delete-dialog Map Marker',
  props<{ id: number }>(),
);
export const deleteMapMarkerSuccess = createAction(
  '[Map Marker] delete-dialog Map Marker Success',
  props<{ id: number }>(),
);
export const deleteMapMarkerFailed = createAction(
  '[Map Marker] delete-dialog Map Marker Failed',
);

export const updateMapMarkerVisibility =
createAction(
  '[Map Marker] update Map Marker Visibility',
  props<{ id: number; isHidden: boolean }>(),
);
export const updateMapMarkerVisibilitySuccess =
createAction(
  '[Map Marker] update Map Marker Visibility
Success',
  props<{ id: number; isHidden: boolean }>(),
);
export const updateMapMarkerVisibilityFailed =
createAction(
  '[Map Marker] update Map Marker Visibility Failed',
);

export const updateMarker = createAction(
  '[Map Marker] update Map Marker',
  props<{ id: number; mapMarker: IMapMarker }>(),
);
export const updateMarkerSuccess = createAction(
  '[Map Marker] update Map Marker Success',
  props<{ mapMarker: IMapMarker }>(),
);
export const updateMarkerFailed = createAction(
  '[Map Marker] update Map Marker Failed',
);
```

frontend/src/store/actions/order.action.ts

```
import { createAction, props } from '@ngrx/store';
import { IOrderDetails, IOrderDto } from
'../../common/interfaces';

export const getOrderDetails = createAction(
  '[Order] get order details',
  props<{ orderNumber: number }>(),
);
export const getOrderDetailsSuccess = createAction(
  '[Order] get order detailsSuccess',
  props<{ orderDetails: IOrderDetails }>(),
);
export const getOrderDetailsFailed = createAction(
  '[Order] get order details Failed',
);

export const getAllOrderDetails =
createAction('[Order] get all order details');
export const getAllOrderDetailsSuccess =
createAction(
  '[Order] get all order detailsSuccess',
  props<{ allOrderDetails: Array<IOrderDetails> }>(),
);
export const getAllOrderDetailsFailed = createAction(
  '[Order] get all order details Failed',
);

export const createOrder = createAction(
  '[Order] create Order',
  props<{
    orderDto: IOrderDto;
  }>(),
);
export const createOrderSuccess =
createAction('[Order] create Order Success');
export const createOrderFailed =
createAction('[Order] create Order Failed');

export const updateOrderStatus = createAction(
  '[Order] update Order details',
  props<{ id: number; status: number }>(),
);
export const updateOrderStatusSuccess = createAction(
  '[Order] update Order details Success',
  props<{ id: number; status: number }>(),
);
export const updateOrderStatusFailed = createAction(
  '[Order] update Order details Failed',
);
```

```
export const createOrderItem = createAction(
  '[Order] create Order Item',
  props<{
    quantity?: number | null;
    productId: number;
    orderId: number;
  }>(),
);
export const createOrderItemSuccess = createAction(
  '[Order] create Order Item Success',
);
export const createOrderItemFailed = createAction(
  '[Order] create Order Item Failed',
);

export const updateOrderItem = createAction(
  '[Order] update Order Item',
  props<{ id: number; quantity: number; orderId:
number }>(),
);
export const updateOrderItemSuccess = createAction(
  '[Order] update Order Item Success',
  props<{ id: number; quantity: number; orderId:
number }>(),
);
export const updateOrderItemFailed = createAction(
  '[Order] update Order Item Failed',
);

export const deleteOrderItemById = createAction(
  '[Order] delete-dialog Order Item',
  props<{ id: number; orderId: number }>(),
);
export const deleteOrderItemByIdSuccess =
createAction(
  '[Order] delete-dialog Order Item Success',
  props<{ id: number; orderId: number }>(),
);
export const deleteOrderItemByIdFailed =
createAction(
  '[Order] delete-dialog Order Item Failed',
);
```

frontend/src/store/actions/product.action.ts

```
import { createAction, props } from '@ngrx/store';
import { IProduct } from '../../common/interfaces';

export const getAllProduct = createAction('[Product]
get Products');
export const getAllProductsSuccess = createAction(
  '[Product] get Products Success',
  props<{ products: Array<IProduct> }>(),
);
export const getAllProductsFailed = createAction(
  '[Product] get Products Failed',
);

export const getCatalog = createAction(
  '[Product] get Catalog',
  props<{ id: number }>(),
);
export const getCatalogSuccess = createAction(
  '[Product] get Catalog Success',
  props<{ productsCatalog: Array<IProduct> }>(),
);
export const getCatalogFailed =
createAction('[Product] get Catalog Failed');

export const createProduct = createAction(
  '[Product] create Map Marker',
  props<{ product: any }>(),
);
export const createProductSuccess = createAction(
  '[Product] create Map Marker Success',
  props<{ product: any }>(),
);
export const createProductFailed = createAction(
  '[Product] create Map Marker Failed',
);

export const updateProduct = createAction(
  '[Product] update Product',
  props<{ id: number; product: any }>(),
);
export const updateProductSuccess = createAction(
  '[Product] update Product Success',
  props<{ product: any }>(),
);
export const updateProductFailed = createAction(
  '[Product] update Product Failed',
);

export const updateProductVisibility = createAction(
  '[Product] update Product Visibility',
  props<{ id: number; isHidden: boolean }>(),
);
export const updateProductVisibilitySuccess =
createAction(
  '[Product] update Product Visibility Success',
  props<{ id: number; isHidden: boolean }>(),
);
export const updateProductVisibilityFailed =
createAction(
  '[Product] update Product Visibility Failed',
);

export const deleteProductById = createAction(
```

```
  '[Product] delete-dialog Product',
  props<{ id: number }>(),
);
export const deleteProductByIdSuccess = createAction(
  '[Product] delete-dialog Product Success',
  props<{ id: number }>(),
);
export const deleteProductByIdFailed = createAction(
  '[Product] delete-dialog Product Failed',
);
```

Backend:

backend/package.json

```
{
  "name": "backend",
  "version": "1.0.0",
  "description": "",
  "main": "dist/app.js",
  "scripts": {
    "migrate": "sequelize-cli db:migrate",
    "migrate:undo": "sequelize-cli db:migrate:undo",
    "start:dev": "nodemon --exec ts-node -r
dotenv/config -r tsconfig-paths/register
./src/index.ts",
    "start": "node -r dotenv/config ./src/server.js",
    "build:ts": "tsc",
    "build:copy": "cp -r shared build && cp -r public
build",
    "build:netlify": "netlify deploy --prod",
    "build": "npm run build:ts"
  },
  "author": "Ivan",
  "license": "ISC",
  "dependencies": {
    "@types/cors": "^2.8.13",
    "@types/express": "^4.17.17",
    "@types/fs-extra": "^11.0.1",
    "@types/multer": "^1.4.7",
    "@types/sequelize": "^4.28.14",
    "cloudinary": "^1.35.0",
    "cors": "^2.8.5",
    "d": "^1.0.1",
    "dotenv": "^16.0.3",
    "express": "^4.18.2",
    "fs": "^0.0.1-security",
    "jwt-decode": "^3.1.2",
    "multer": "^1.4.5-lts.1",
    "netlify-lambda": "^2.0.16",
    "path": "^0.12.7",
    "path-to-regexp": "^6.2.1",
    "pg": "^8.10.0",
    "qs": "^6.11.1",
    "sequelize": "^6.30.0",
    "serverless-http": "^3.2.0",
    "ts-node": "^10.9.1",
    "tslib": "^2.5.0",
    "yup": "^1.1.1"
  },
  "devDependencies": {
    "@types/node": "^18.15.11",
    "netlify-cli": "^15.2.0",
    "tsconfig-paths": "^4.2.0",
    "typescript": "^5.0.3"
  }
}
```

backend/DockerFile

```
# Use the official Node.js image as the base image
FROM node:16

# Set the working directory
WORKDIR /app

# Copy package.json and package-lock.json to the
working directory
COPY package*.json ./

# Install dependencies
RUN npm install

# Copy the rest of the application code to the
working directory
COPY . .

# Copy .env file to the working directory
COPY .env .env

# Expose the port the app runs on
EXPOSE 3000

# Command to run the application
CMD ["npm", "start"]
```

backend/src/index.ts

```
import { app } from './server';
import { ENV } from './common/enums';

const server = app.listen(ENV.APP.SERVER_PORT, () =>
{
  console.log(`Listening to connections on port —
${ENV.APP.SERVER_PORT}`);
});

export { server };
```

backend/src/server.ts

```
import express, { json, urlencoded } from 'express';
import { initApi } from './api/api';
import { sequelize } from './data/db/connection';
import { LogRequest } from './middleware/request-logger.middleware';
import cors from 'cors';
import { ConnectionError as DbConnectionError } from 'sequelize';

const app = express();

sequelize
  .authenticate()
  .then(() => {
    return console.log('Database connection was successful');
  })
  .catch(({ message, stack }: DbConnectionError) => {
    return console.error(message, stack);
  });

app.use(cors());
app.use(LogRequest);
// app.use(mapMarkerValidationMiddleware);
app.use(json({ limit: '670kb' }));
app.use(urlencoded({ extended: true }));

app.use(function (req, res, next) {
  res.header('Access-Control-Allow-Origin', '*');
  res.header(
    'Access-Control-Allow-Headers',
    'Origin, X-Requested-With, Content-Type, Accept',
  );
  res.header('Access-Control-Allow-Methods', 'DELETE, PATCH, GET, POST');
  next();
});

initApi(app);

export { app };
```

backend/src/api/api.ts

```
import { Router } from 'express';
import { initUserApi } from './user.api';
import { initRoleApi } from './role.api';
import { initCartApi } from './cart.api';
import { initMapMarkerApi } from './map-marker.api';
import { initProductApi } from './product.api';
import { initOrderApi } from './order.api';

const apis: any[] = [
  initUserApi,
  initRoleApi,
  initCartApi,
  initMapMarkerApi,
  initProductApi,
  initOrderApi,
];

export const initApi = (app: Router): Router => {
  const apiRouter = Router();

  app.use('/api/', apiRouter);
  apis.forEach((api: any) => api(apiRouter));

  return apiRouter;
};
```

backend/src/api/cart.api.ts

```
import { Router } from 'express';
import {
  cartApiPathEnum,
  HttpCode,
  rootApiPathEnum,
  UserRolesEnum,
} from '../common/enums';
import { cartItemService, cartService } from '../services';
import { checkIsFound } from '~/utils';
import { verifyTokenMiddleware } from '~/middleware/verify-token.middleware';
import { authMiddleware } from '~/middleware/auth.middleware';
import { cartValidationMiddleware } from '~/middleware/validation';

export const initCartApi = (apiRouter: Router): Router => {
  const cartRouter = Router();

  apiRouter.use(rootApiPathEnum.Cart, cartRouter);

  cartRouter.get(
    cartApiPathEnum.ROOT,
    [
      verifyTokenMiddleware,
      authMiddleware([UserRolesEnum.Admin, UserRolesEnum.Customer]),
    ],
    async (_req, res) => {
      try {
        const cart = await cartService.getCartByUserId(`${_req.query.userId}`);
```

```
        res.status(HttpCode.OK).json(cart);
      } catch (error) {
        console.log(error);
        res.status(HttpCode.NOT_FOUND).json({
message: error.message });
      }
    },
  );

  cartRouter.post(cartApiPathEnum.ROOT, async (_req,
res) => {
    try {
      const cartItem = await
cartService.createCartByUserId(_req.body?.userId);


res.status(checkIsFound(cartItem)).json(cartItem);
    } catch (error) {
      res.status(HttpCode.BAD_REQUEST).json({
message: error.message });
    }
  });

  cartRouter.post(
    cartApiPathEnum.CreateCartItem,
    verifyTokenMiddleware,
    authMiddleware([UserRolesEnum.Admin]),
    cartValidationMiddleware,
    async (_req, res) => {
      try {
        const cartItem = await
cartItemService.createCartItem({
          ..._req.body,
          cartId: Number(_req.query.cartId || 0),
          productId: Number(_req.query.productId ||
0),
        });


res.status(checkIsFound(cartItem)).json(cartItem);
      } catch (error) {
        res.status(HttpCode.BAD_REQUEST).json({
message: error.message });
      }
    },
  );

  cartRouter.patch(
    cartApiPathEnum.UpdateCartItem,
    verifyTokenMiddleware,
    authMiddleware([UserRolesEnum.Admin]),
    cartValidationMiddleware,
    async (_req, res) => {
      try {
        const cartItem = await
cartItemService.updateCartItem(
          Number(_req.query.id || 0),
          {
            ..._req.body,
          },
        );


res.status(checkIsFound(cartItem)).json(cartItem);
      } catch (error) {
        res.status(HttpCode.NOT_FOUND).json({
message: error.message });
      }
    },
  );

  cartRouter.delete(
    cartApiPathEnum.DeleteCartItem,
    verifyTokenMiddleware,
    authMiddleware([UserRolesEnum.Admin]),
    async (_req, res) => {
      try {
        const idsArray =
Array.isArray(_req.query?.id)
          ? (_req.query?.id as any)?.map((id) =>
Number(id || 0))
          : [Number(_req.query?.id || 0)];

        await cartItemService.deleteById(idsArray);


res.status(HttpCode.NO_CONTENT).json('Success');
      } catch (err) {
        const error = err?.errors?.[0];
        res
          .status(HttpCode.INTERNAL_SERVER_ERROR)
          .json({ message: error?.message });
      }
    },
  );

  cartRouter.delete(
    cartApiPathEnum.DeleteByUserId,
    verifyTokenMiddleware,
    authMiddleware([UserRolesEnum.Admin]),
    async (_req, res) => {
      try {
        await
cartService.deleteCartByUserId(_req.params.id);


res.status(HttpCode.NO_CONTENT).json('Success');
      } catch (err) {
        const error = err?.errors?.[0];
```

```
        res
          .status(HttpCode.INTERNAL_SERVER_ERROR)
          .json({ message: error?.message });
      }
    },
  );

  return cartRouter;
};
```

backend/src/api/map-marker.api.ts

```
import { Router } from 'express';
import {
  HttpCode,
  mapMarkerApiPathEnum,
  rootApiPathEnum,
  UserRolesEnum,
} from '../common/enums';
import { mapMarkerService } from '../services';
import { checkIsFound } from '~/utils';
import { verifyTokenMiddleware } from
'~/middleware/verify-token.middleware';
import { authMiddleware } from
'~/middleware/auth.middleware';
import { mapMarkerValidationMiddleware } from
'~/middleware/validation';

export const initMapMarkerApi = (apiRouter: Router):
Router => {
  const mapMarkerRouter = Router();

  apiRouter.use(rootApiPathEnum.MapMarker,
mapMarkerRouter);

  mapMarkerRouter.get(
    mapMarkerApiPathEnum.ROOT,
    verifyTokenMiddleware,
    authMiddleware([UserRolesEnum.Admin,
UserRolesEnum.Customer]),
    async (_req, res) => {
      try {
        const mapMarker = await
mapMarkerService.getAllMarkersByRole(
          Number(`${_req.query.roleId ||
UserRolesEnum.Customer}`),
        );

res.status(checkIsFound(mapMarker)).json(mapMarker);
      } catch (error) {
        res.status(HttpCode.NOT_FOUND).json({
message: error?.message });
      }
    },
  );

  mapMarkerRouter.get(
    mapMarkerApiPathEnum.GetProducts,
    verifyTokenMiddleware,
    authMiddleware([UserRolesEnum.Admin]),
    async (_req, res) => {
      try {
        const mapMarker = await
mapMarkerService.getByIdWithProductsInfo(
          Number(_req?.query?.id || 0),
        );

res.status(checkIsFound(mapMarker)).json(mapMarker);
      } catch (error) {
        res.status(HttpCode.NOT_FOUND).json({
message: error?.message });
      }
    },
  );

  mapMarkerRouter.post(
    mapMarkerApiPathEnum.ROOT,
    verifyTokenMiddleware,
    authMiddleware([UserRolesEnum.Admin]),
    // mapMarkerValidationMiddleware,
    async (_req, res) => {
      try {
        const createdMapMarker = await
mapMarkerService.createMapMarker(
          _req.body,
        );

res.status(HttpCode.OK).json(createdMapMarker);
      } catch (err) {
        const error = err?.errors?.[0] || 'error';
        res.status(HttpCode.BAD_REQUEST).json({
message: error?.message });
      }
    },
  );

  mapMarkerRouter.patch(
    mapMarkerApiPathEnum.$ID,
    verifyTokenMiddleware,
    authMiddleware([UserRolesEnum.Admin]),
    // mapMarkerValidationMiddleware,
    async (_req, res) => {
      try {
        const mapMarker = await
mapMarkerService.updateMapMarker(
          Number(_req?.params?.id || 0),
          _req.body,
        );
        res.status(HttpCode.OK).json(mapMarker);
```

```
      } catch (err) {
        const error = err.errors[0];
        res
          .status(HttpCode.INTERNAL_SERVER_ERROR)
          .json({ message: error.message });
      }
    },
  );

  mapMarkerRouter.delete(
    mapMarkerApiPathEnum.$ID,
    verifyTokenMiddleware,
    authMiddleware([UserRolesEnum.Admin]),
    async (_req, res) => {
      try {
        await
mapMarkerService.deleteMapMarker(Number(_req?.params?
.id || 0));

res.status(HttpCode.NO_CONTENT).json('Success');
      } catch (err) {
        const error = err?.errors?.[0];
        res
          .status(HttpCode.INTERNAL_SERVER_ERROR)
          .json({ message: error?.message });
      }
    },
  );

  return mapMarkerRouter;
};
```

backend/src/services/map-marker.service.ts

```
import { IMapMarkerDto, IMapMarkerWithProductInfo }
from '~/common/interfaces';
import { mapMarkerRepository } from
'~/data/repositories';

export class MapMarkerService {
  public getAllMarkersByRole(roleId: number):
Promise<Array<IMapMarkerDto>> {
    return mapMarkerRepository.getAllMarkers(roleId);
  }

  public getByIdWithProductsInfo(
    id: number,
  ): Promise<IMapMarkerWithProductInfo> {
    return new Promise(async (resolve, reject) => {
      try {
        const mapMarkers = await
mapMarkerRepository.getMapMarketWithProducts(
          id,
        );
        resolve({
          ...mapMarkers?.[0],
          products: mapMarkers.map(({ products }) =>
products),
        });
      } catch (error) {
        reject(error);
      }
    });
  }

  public createMapMarker(mapMarker: IMapMarkerDto):
Promise<IMapMarkerDto> {
    return
mapMarkerRepository.createMapMarker(mapMarker);
  }

  public async updateMapMarker(
    id: number,
    data: IMapMarkerDto,
  ): Promise<IMapMarkerDto[]> {
    return mapMarkerRepository.updateById(id, data);
  }

  public deleteMapMarker(id: number): Promise<number>
{
    return mapMarkerRepository.deleteById(id);
  }
}
```

backend/src/services/upload-image.service.ts

```
import cloudinary from '../config/cloudinary.config';

export class UploadImageService {
  public async uploadImage(file:
Express.Multer.File): Promise<string> {
    const cloudResponse = await
cloudinary.uploader.upload(file.path);
    return `${cloudResponse.url}`;
  }

  public async deleteUploadedImage(url: string):
Promise<void> {
    try {
      const convertedUrl = url?.split('/');
      if (convertedUrl?.length) {
        const imagePublicId =
          convertedUrl?.[convertedUrl.length -
1]?.split('.')[0];
        await
cloudinary.uploader.destroy(imagePublicId);
      }
    } catch (error) {
      console.log(error);
```

```
    }
  }
}
```

backend/src/services/product.service.ts

```
import { productRepository } from
'../data/repositories';
import { IProductDto } from '~/common/interfaces';

export class ProductService {
  public getAllProducts(): Promise<IProductDto[]> {
    return productRepository.getAll();
  }
  public getById(id: number): Promise<IProductDto> {
    return productRepository.getById(id);
  }
  public createNewProduct(product: IProductDto):
Promise<IProductDto> {
    return productRepository.createProduct(product);
  }

  public async updateProduct(
    id: number,
    data: IProductDto,
  ): Promise<Array<IProductDto>> {
    return productRepository.updateById(id, data);
  }

  public deleteProduct(id: number): Promise<number> {
    return productRepository.deleteById(id);
  }
}
```

backend/src/middleware/auth.middleware.ts

```
import jwt_decode from 'jwt-decode';
import { userService } from '~/services';

export const authMiddleware = (roles) => {
  return async (req, _res, next) => {
    const token = req?.headers?.token as string;

    try {
      if (token) {
        const decodedToken = jwt_decode(token);
        let user: any = await
userService.getUserBySubId(decodedToken?.['sub']);
        user = user?.[0]?.dataValues;

        if (user?.role) {
          roles.forEach((role) => {
            if ([user?.role].includes(role)) return
next();
          });
        } else {
          throw new Error('no role');
        }
      } else {
        return _res.status(403).json({
          message: 'Forbidden',
        }) as any;
      }
    } catch (error) {
      console.log(error);
      return _res.status(403).json({
        message: 'Forbidden',
      }) as any;
    }
  };
};
```

backend/src/middleware/upload-fille.middleware.ts

```
import multer from 'multer';

export const multerUploadFile = (): multer.Multer =>
{
  const diskStorage = multer.diskStorage({
    filename: function (req, file, cb) {
      const uniqueSuffix = Date.now() + '-' +
Math.round(Math.random() * 1e9);
      const fileFormat =
file.originalname.split('.');
      cb(
        null,
        `${file.fieldname}-${uniqueSuffix}.${
          fileFormat[fileFormat.length - 1]
        }`,
      );
    },
  });

  return multer({ storage: diskStorage });
};
```

backend/src/middleware/verify-token.middleware.ts

```
import { RequestHandler } from 'express';
import jwt_decode from 'jwt-decode';
import { userService } from '~/services';

export const verifyTokenMiddleware: RequestHandler =
async (
  req,
  _res,
  next,
): Promise<void> => {
```

```
  const token = req?.headers?.token as string;

  try {
    if (token) {
      const decodedToken = jwt_decode(token);
      const user = await
userService.getUserBySubId(decodedToken?.['sub']);

      if (user?.length) {
        return next();
      }
    } else {
      return _res.status(403).json({
        message: 'Forbidden',
      }) as any;
    }
  } catch (error) {
    console.log(error);
    return _res.status(403).json({
      message: 'Forbidden',
    }) as any;
  }
};
```

backend/src/config/cloudinary.config.ts

```
import dotenv from 'dotenv';
import cloudinary from 'cloudinary';
dotenv.config();

cloudinary.v2.config({
  cloud_name: process.env.CLOUDINARY_NAME,
  api_key: process.env.CLOUDINARY_API_KEY,
  api_secret: process.env.CLOUDINARY_API_SECRET,
});

export default cloudinary.v2;
```

backend/src/config/database.config.js

```
const {
  DB_NAME,
  DB_USERNAME,
  DB_PASSWORD,
  DB_HOST,
  DB_PORT,
  DB_DIALECT,
  DB_TEST_NAME,
  DB_TEST_USERNAME,
  DB_TEST_PASSWORD,
  DB_TEST_HOST,
  DB_TEST_PORT,
  USE_SSL,
} = process.env;

const dialectOptions =
  USE_SSL === 'true'
    ? {
        ssl: {
          require: true,
          rejectUnauthorized: false,
        },
      }
    : {};

module.exports = {
  development: {
    database: DB_NAME,
    username: DB_USERNAME,
    password: DB_PASSWORD,
    host: DB_HOST,
    port: DB_PORT,
    dialect: DB_DIALECT,
    logging: false,
    dialectOptions,
  },
  test: {
    database: DB_TEST_NAME,
    username: DB_TEST_USERNAME,
    password: DB_TEST_PASSWORD,
    host: DB_TEST_HOST,
    port: DB_TEST_PORT,
    dialect: DB_DIALECT,
    logging: false,
    dialectOptions,
  },
  production: {
    database: DB_NAME,
    username: DB_USERNAME,
    password: DB_PASSWORD,
    host: DB_HOST,
    port: DB_PORT,
    dialect: DB_DIALECT,
    logging: false,
    dialectOptions,
  },
};
```

backend/src/data/db/connection.ts

```
import { Dialect, Sequelize } from 'sequelize';
import databaseConfig from
'../../config/database.config';
import { EnvironmentEnum } from '../../common/enums';

const db =
```

```
  process.env.NODE_ENV ===
EnvironmentEnum.DEVELOPMENT
    ? databaseConfig.development
    : databaseConfig.production;

const sequelize = new Sequelize({
  ...db,
  port: Number(db.port),
  dialect: db.dialect as Dialect,
});

export { sequelize };
```

backend/src/data/models/map-marker.module.ts

```
import { DataTypes, Model, ModelCtor, Sequelize }
from 'sequelize';
import { ModelName } from '../../common/enums';
import { IMapMarkerDto } from '~/common/interfaces';

interface mapMarkerInstance extends IMapMarkerDto,
Model {}

const createMapMarkerModule = (
  orm: Sequelize,
): ModelCtor<mapMarkerInstance> => {
  return orm.define<mapMarkerInstance>(
    ModelName.MapMarker,
    {
      id: {
        type: DataTypes.INTEGER,
        autoIncrement: true,
        allowNull: false,
        primaryKey: true,
      },
      name: {
        type: DataTypes.STRING,
      },
      description: {
        type: DataTypes.STRING,
      },
      hidden: {
        type: DataTypes.BOOLEAN,
      },
      productIds: {
        type: DataTypes.ARRAY(DataTypes.INTEGER),
        defaultValue: [],
      },
      lat: {
        type: DataTypes.REAL,
      },
      lng: {
        type: DataTypes.REAL,
      },
      createdAt: DataTypes.DATE,
      updatedAt: DataTypes.DATE,
    },
    {
      tableName: 'map_marker',
      underscored: true,
    },
  );
};
export default createMapMarkerModule;
```

backend/src/data/models/product.module.ts

```
import { DataTypes, Model, ModelCtor, Sequelize }
from 'sequelize';
import { ModelName } from '../../common/enums';
import { IProductDto } from '~/common/interfaces';

interface productInstance extends IProductDto, Model
{}

const createProductModule = (orm: Sequelize):
ModelCtor<productInstance> => {
  return orm.define<productInstance>(
    ModelName.Product,
    {
      id: {
        type: DataTypes.INTEGER,
        autoIncrement: true,
        allowNull: false,
        primaryKey: true,
      },
      name: {
        allowNull: false,
        type: DataTypes.STRING,
      },
      description: {
        type: DataTypes.STRING,
      },
      image: {
        allowNull: false,
        type: DataTypes.STRING,
      },
      hidden: {
        type: DataTypes.BOOLEAN,
        allowNull: false,
      },
      price: {
        type: DataTypes.REAL,
        allowNull: false,
      },
      createdAt: DataTypes.DATE,
      updatedAt: DataTypes.DATE,
    },
```

```
    {
      tableName: 'product',
      underscored: true,
    },
  );
};
export default createProductModule;
```

backend/src/data/models/user.model.ts

```
import { DataTypes, Model, ModelCtor, Sequelize } from 'sequelize';
import { IUserDto } from '../../common/interfaces';
import { ModelName } from '../../common/enums';

interface userInstance extends IUserDto, Model {}

const createUserModule = (orm: Sequelize): ModelCtor<userInstance> => {
  return orm.define<userInstance>(
    ModelName.User,
    {
      id: {
        type: DataTypes.UUID,
        defaultValue: DataTypes.UUIDV4,
        allowNull: false,
        primaryKey: true,
      },
      name: {
        allowNull: false,
        type: DataTypes.STRING,
      },
      email: {
        allowNull: false,
        type: DataTypes.STRING,
        unique: true,
      },
      subId: {
        allowNull: false,
        type: DataTypes.STRING,
        unique: true,
      },
      roleId: {
        allowNull: false,
        type: DataTypes.INTEGER,
        defaultValue: 1,
      },
      createdAt: DataTypes.DATE,
      updatedAt: DataTypes.DATE,
    },
    {
      tableName: 'user',
      underscored: true,
    },
  );
};
export default createUserModule;
```

backend/src/data/models/product.module.ts

```
import { DataTypes, Model, ModelCtor, Sequelize } from 'sequelize';
import { ModelName } from '../../common/enums';
import { IProductDto } from '~/common/interfaces';

interface productInstance extends IProductDto, Model {}

const createProductModule = (orm: Sequelize): ModelCtor<productInstance> => {
  return orm.define<productInstance>(
    ModelName.Product,
    {
      id: {
        type: DataTypes.INTEGER,
        autoIncrement: true,
        allowNull: false,
        primaryKey: true,
      },
      name: {
        allowNull: false,
        type: DataTypes.STRING,
      },
      description: {
        type: DataTypes.STRING,
      },
      image: {
        allowNull: false,
        type: DataTypes.STRING,
      },
      hidden: {
        type: DataTypes.BOOLEAN,
        allowNull: false,
      },
      price: {
        type: DataTypes.REAL,
        allowNull: false,
      },
      createdAt: DataTypes.DATE,
      updatedAt: DataTypes.DATE,
    },
    {
      tableName: 'product',
      underscored: true,
    },
  );
};
```

```
export default createProductModule;
```

backend/.env

```
#
# APPLICATION
#
NODE_ENV=development
#
# DATABASE
#
DB_NAME=Floresta2.0
DB_USERNAME=postgres
DB_PASSWORD=1
DB_HOST=localhost
DB_PORT=5432
DB_DIALECT=postgres
USE_SSL=false
#
#Cloudinary
#
CLOUDINARY_NAME=---------
CLOUDINARY_API_KEY=-----------
CLOUDINARY_API_SECRET=-----------
```

backend/src/services/cart-item.service.ts

```
import { ICartItemDto } from '~/common/interfaces';
import { cartItemRepository } from
'~/data/repositories';

export class CartItemService {
  public getById(id: number): Promise<ICartItemDto> {
    return cartItemRepository.getById(id);
  }
  public createCartItem(cartItemDto: ICartItemDto):
Promise<ICartItemDto> {
    return
cartItemRepository.createCartItem(cartItemDto);
  }

  public updateCartItem(
    id: number,
    cartItemDto: ICartItemDto,
  ): Promise<ICartItemDto[]> {
    return cartItemRepository.updateById(id,
cartItemDto);
  }

  public deleteCartById(id: number): Promise<number>
{
    return cartItemRepository.deleteCartById(id);
  }
  public deleteById(ids: Array<number>):
Promise<number> {
    return cartItemRepository.delete(ids);
  }
}
```

backend/src/services/cart.service.ts

```
import { ICart, ICartFullInfo } from
'~/common/interfaces';
import { cartRepository } from '~/data/repositories';
import { cartMapper } from '~/utils/cart.mapper';
import { cartItemService, cartService } from
'~/services/index';

export class CartService {
  public getById(id: number): Promise<ICart> {
    return cartRepository.getById(id);
  }
  public createCartByUserId(userId: string):
Promise<ICart> {
    return cartRepository.createCart({
      userId,
      productIds: [],
    });
  }

  public getCartByUserId(userId: string):
Promise<ICartFullInfo> {
    return new Promise(async (resolve, reject) => {
      try {
        const cart = cartMapper(
          (await cartRepository.getByUserId(userId))
as any,
        );

        resolve(cart);
      } catch (error) {
        reject(error);
      }
    });
  }

  public deleteCartByUserId(userId: string):
Promise<number> {
    return new Promise(async (resolve, reject) => {
      try {
        const cart = await
cartService.getCartByUserId(userId);

        if (cart.id) {
          await
cartService.createCartByUserId(userId);
          await cartRepository.deleteById(cart.id);
          await
cartItemService.deleteCartById(cart.id);
```

```
        }

        resolve(0);
      } catch (error) {
        reject(error);
      }
    });
  }
}
```

backend/src/data/atributes/map-marker.atributes.ts

```
import { ParamsTypeEnum, UserRolesEnum } from
'~/common/enums';
import { productModule } from '~/data/models';
import { productAttribute } from
'~/data/atributes/product.atribute';

export const mapMarkerAttribute = [
  'id',
  'lat',
  'lng',
  'hidden',
  'productIds',
  'name',
  'description',
];

const getParamsByRole = (roleId =
UserRolesEnum.Customer) => {
  const paramsByRole: any = {
    attributes: mapMarkerAttribute,
  };

  if (roleId === UserRolesEnum.Customer) {
    return {
      ...paramsByRole,
      where: {
        hidden: false,
      },
    };
  }
  return paramsByRole;
};

export const getMapMarketParams = (
  type = '',
  additionalParams: { [key: string]: any } = {},
) => {
  const defaultParams = {
    attributes: mapMarkerAttribute,
  };

  switch (type) {
    case 'filter':
      return { ...defaultParams, where: {
...additionalParams } };
    case ParamsTypeEnum.GetByRoleId:
      return {
...getParamsByRole(additionalParams?.roleId) };
    case ParamsTypeEnum.ProductInfo:
      return {
        ...defaultParams,
        include: {
          model: productModule,
          attributes: productAttribute,
          as: 'products',
        },
      };
    default:
      return defaultParams;
  }
};
```

backend/src/data/atributes/order-details.atribute.ts

```
import {
  orderItemModule,
  paymentDetailModule,
  productModule,
  userModule,
} from '~/data/models';
import { productAttribute } from
'~/data/atributes/product.atribute';

export const orderDetailsAttribute = [
  'id',
  'type',
  'isAnonymous',
  'total',
  'status',
  'createdAt',
  'mapMarkerId',
];

export const getOrderDetailsParams = (type = '',
additionalParams = {}) => {
  const defaultParams = {
    attributes: orderDetailsAttribute,
    include: [
      {
        model: orderItemModule,
        attributes: ['id', 'quantity'],
        as: 'items',
        include: [
          {
            model: productModule,
            attributes: productAttribute,
```

```
            as: 'product',
          },
        ],
      },
      {
        model: paymentDetailModule,
        attributes: ['id', 'provider'],
        as: 'paymentDetails',
      },

      {
        model: userModule,
        attributes: ['id', 'name'],
        as: 'user',
      },
    ],
  };

  switch (type) {
    case 'filter':
      return { ...defaultParams, where: {
...additionalParams } };
    default:
      return defaultParams;
  }
};
```

backend/src/data/atributes/order-item.atribute.ts

```
export const orderItemAttribute = ['id', 'productId',
'orderId'];

export const getOrderItemParams = (type = '',
additionalParams = {}) => {
  const defaultParams = {
    attributes: orderItemAttribute,
  };

  switch (type) {
    case 'filter':
      return { ...defaultParams, where: {
...additionalParams } };
    default:
      return defaultParams;
  }
};
```

backend/src/data/atributes/user.atribute.ts

```
import { roleModule } from '~/data/models';

export const userAttribute = ['id', 'name', 'email',
['sub_id', 'subId']];

export const roleIdAsRole = [['role_id', 'role']];
export const getUserParams = (type = '',
additionalParams = {}): any => {
  const defaultParams = {
    attributes: [...userAttribute, ...roleIdAsRole],
  };

  switch (type) {
    case 'withRoleName':
      return {
        attributes: [...userAttribute],
        include: {
          model: roleModule,
          as: 'role',
          attributes: ['name'],
        },
      };
    case 'filter':
      return { ...defaultParams, where: {
...additionalParams } };
    default:
      return defaultParams;
  }
};
```

backend/src/data/atributes/role.atribute.ts

```
export const roleAttribute = ['id', 'name'];

export const getRoleParams = (type = '',
additionalParams = {}) => {
  const defaultParams = {
    attributes: roleAttribute,
  };

  switch (type) {
    case 'filter':
      return { ...defaultParams, where: {
...additionalParams } };
    default:
      return defaultParams;
  }
};
```

backend/src/data/atributes/order-item.atribute.ts

```
export const orderItemAttribute = ['id', 'productId',
'orderId'];

export const getOrderItemParams = (type = '',
additionalParams = {}) => {
  const defaultParams = {
    attributes: orderItemAttribute,
  };
```

```
  switch (type) {
    case 'filter':
      return { ...defaultParams, where: {
...additionalParams } };
    default:
      return defaultParams;
  }
};
```

backend/src/data/atributes/index.ts

```
export * from './user.atribute';
export * from './role.atribute';
export * from './cart.atribute';
export * from './cart-item.atribute';
export * from './order-details.atribute';
export * from './order-item.atribute';
export * from './payment-details.atribute';
export * from './product.atribute';
export * from './map-marker.atributes';
```

backend/src/data/repositories/index.ts

```
import { UserRepository } from './user.repository';
import { RoleRepository } from './role.repository';
import { CartRepository } from './cart.repository';
import { CartItemRepository } from './cart-
item.repository';
import { OrderDetailsRepository } from './order-
details.repository';
import { OrderItemRepository } from './order-
item.repository';
import { PaymentDetailsRepository } from './payment-
details.repository';
import { ProductRepository } from
'./product.repository';
import { MapMarkerRepository } from './map-
marker.repository';

export const userRepository = new UserRepository();
export const roleRepository = new RoleRepository();
export const cartRepository = new CartRepository();
export const cartItemRepository = new
CartItemRepository();
export const orderDetailsRepository = new
OrderDetailsRepository();
export const orderItemRepository = new
OrderItemRepository();
export const paymentDetailsRepository = new
PaymentDetailsRepository();
export const productRepository = new
ProductRepository();
export const mapMarkerRepository = new
MapMarkerRepository();
```

backend/src/data/repositories/map-marker.repository.ts

```
import { mapMarkerModule } from '../models';
import { getMapMarketParams } from '../atributes';
import { IMapMarkerDto } from '~/common/interfaces';
import { ParamsTypeEnum } from '~/common/enums';
import { QueryTypes } from 'sequelize';

export class MapMarkerRepository {
  public getById(id: number): Promise<IMapMarkerDto |
null> {
    return mapMarkerModule.findByPk(id,
getMapMarketParams());
  }

  public getMapMarketWithProducts(id: number):
Promise<any> {
    return mapMarkerModule.sequelize.query(
      `SELECT "map_marker"."id", "map_marker"."lat",
"map_marker"."lng", "map_marker"."hidden",
          "products"."id" AS "products.id",
"products"."name" AS "products.name",
"products"."image"
          AS "products.image", "products"."price" AS
"products.price",
          "products"."hidden" AS "products.hidden"
FROM "map_marker" AS "map_marker"
          LEFT OUTER JOIN "product" AS "products" ON
"products"."id" = ANY("map_marker"."product_ids")
          WHERE "map_marker"."id" = ${id};`,
      { type: QueryTypes.SELECT, raw: false, nest:
true },
    );
  }

  public getAllMarkers(roleId: number):
Promise<Array<IMapMarkerDto> | null> {
    return mapMarkerModule.findAll(
      getMapMarketParams(ParamsTypeEnum.GetByRoleId,
{ roleId: roleId }),
    );
  }

  public createMapMarker(newMapMarker:
IMapMarkerDto): Promise<IMapMarkerDto> {
    return mapMarkerModule.create(newMapMarker as
any);
  }

  public async updateById(
    id: number,
    data: IMapMarkerDto,
  ): Promise<IMapMarkerDto[]> {
    const result = await mapMarkerModule.update(data,
{
      where: { id },
      returning: true,
    });
```

```
    return result[1];
  }

  public deleteById(id: number): Promise<number> {
    return mapMarkerModule.destroy({
      where: { id },
    });
  }
}
```

backend/src/data/repositories/order-item.repository.ts

```
import { orderItemModule } from '../models';
import { getOrderItemParams } from '../atributes';
import { IOrderItemDto } from '~/common/interfaces';

export class OrderItemRepository {
  public getById(id: number): Promise<IOrderItemDto | null> {
    return orderItemModule.findByPk(id, getOrderItemParams());
  }

  public createOrderItem(orderItem: IOrderItemDto): Promise<IOrderItemDto> {
    return orderItemModule.create(orderItem as any);
  }

  public async updateById(
    id: number,
    data: IOrderItemDto,
  ): Promise<IOrderItemDto[]> {
    const result = await orderItemModule.update(data, {
      where: { id },
      returning: true,
    });
    return result[1];
  }

  public deleteByOrderId(orderId: number): Promise<number> {
    return orderItemModule.destroy({
      where: { orderId: orderId },
    });
  }

  public deleteById(id: number): Promise<number> {
    return orderItemModule.destroy({
      where: { id: id },
    });
  }
}
```

backend/src/data/repositories/product.repository.ts

```
import { productModule } from '../models';
import { getProductParams } from '../atributes';
import { IProductDto } from '~/common/interfaces';

export class ProductRepository {
  public getById(id: number): Promise<IProductDto | null> {
    return productModule.findByPk(id, getProductParams());
  }

  public getAll(): Promise<Array<IProductDto>> {
    return productModule.findAll(getProductParams());
  }

  public createProduct(product: any): Promise<IProductDto> {
    return productModule.create(product);
  }

  public async updateById(
    id: number,
    data: IProductDto,
  ): Promise<Array<IProductDto>> {
    const result = await productModule.update(data, {
      where: { id },
      returning: true,
    });
    return result[1];
  }

  public deleteById(id: number): Promise<number> {
    return productModule.destroy({
      where: { id },
    });
  }
}
```

backend/src/data/repositories/user.repository.ts

```
import { userModule } from '../models';
import { IUserDto } from '../../common/interfaces';
import { getUserParams } from '../atributes';

export class UserRepository {
  public getAll(): Promise<Array<IUserDto>> {
    return userModule.findAll(getUserParams('withRoleName'));
  }

  public getById(subId: string): Promise<IUserDto[] | null> {
    return userModule.findAll(getUserParams('filter', { subId: subId }));
```

```
  }

  public createUser(user: any): Promise<IUserDto> {
    return userModule.create(user);
  }

  public async updateById(id: string, data:
IUserDto): Promise<IUserDto[]> {
    const result = await userModule.update(data, {
      where: { id },
      returning: true,
    });
    return result[1];
  }

  public deleteById(id: string): Promise<number> {
    return userModule.destroy({
      where: { id },
    });
  }
}
```