ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ТЕХНОЛОГІЙ ЦИФРОВОГО РОЗВИТКУ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Цифровий сервіс для контролю та керування доставками посилок на основі технології ASP.NET Core»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Технології цифрового розвитку»

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

_____ Максим ПАЛАМАРЧУК
(підпис)

Виконав: здобувач вищої освіти групи ТЦР-44

Максим ПАЛАМАРЧУК

Керівник: Микола ГЕРЦЮК
д-р філософії

Рецензент: _____

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Технологій цифрового розвитку

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Технології цифрового розвитку»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Технологій цифрового розвитку
_____ Вікторія ЖЕБКА
« ____ » _____ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Паламарчуку Максиму Олеговичу

1. Тема кваліфікаційної роботи: «Цифровий сервіс для контролю та керування доставками посилок на основі технології ASP.NET Core»
керівник кваліфікаційної роботи д-р філософії Микола ГЕРЦЮК,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «20» лютого 2026 року № 51.

2. Строк подання кваліфікаційної роботи «___» травня 2026 року.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про методи отримання даних про місцезнаходження, методи обміну повідмленнями, технічна документація фреймворку ASP.NET Core.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Аналіз існуючих методів відстеження місцезнаходження та обміну повідомленнями в реальному часі.
2. Проектування застосунку для контролю та керування доставками посилок.
3. Реалізація та опис функціоналу додатку для контролю та керування доставками посилок.
4. Тестування додатку.

5. Перелік графічного матеріалу: *презентація*

6. Дата видачі завдання «20» лютого 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	21.02-28.02.2026	
2	Аналіз та дослідження існуючих аналогів	16.03-19.03.2026	
3	Огляд існуючих алгоритмів отримання поточного місцезнаходження та обміну повідомленнями	20.03-22.03.2026	
4	Проектування застосунку «Цифровий сервіс для контролю та керування доставками посилок»	23.03-29.03.2026	
5	Програмна реалізація застосунку	30.03-15.04.2026	
6	Тестування застосунку	16.04-17.04.2026	
7	Оформлення роботи: вступ, висновки, реферат	17.04-25.04.2026	
8	Розробка демонстраційних матеріалів	26.04-27.04.2026	
9	Попередній захист роботи	09.05-12.05.2026	

Здобувач вищої освіти _____ Максим ПАЛАМАРЧУК
(підпис)

Керівник
кваліфікаційної роботи _____ Микола ГЕРЦЮК
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: _58_ стор., __5__ табл., __44__ рис., __22__ джерел.

Мета роботи – спроектувати веб-додаток для зручного трекінгу посилок та управління доставками посилок та можливістю комунікувати з кур'єром всередині додатку.

Об'єкт дослідження – процеси управління, відстеження поштових відправлень та методи обміну повідомленнями в інформаційних системах.

Предмет дослідження – програмне забезпечення для трекінгу посилок та управління доставками посилок та можливістю комунікувати з кур'єром всередині додатку.

Короткий зміст роботи: В кваліфікаційній роботі проведено дослідження сучасних методів трекінгу геопозиції в інформаційних системах, методи обміну повідомленнями між користувачами в реальному часі.

Проаналізовано існуючі системи управління доставками, висвітливши їх кращі сторони та те чого не вистачає. Спроектовано архітектуру веб-додатку для сервісу управління та контролю доставками, з реалізацією функціоналу обробки замовлень, трекінгу посилки, генерації накладної, чату з кур'єром, сетапу інформаційних сторінок та програми лояльності.

Спроектовано базу даних для зберігання даних про клієнтів, поштових відділень, даних про відправлення, бонусів, повідомлень з чату, реалізовано контроллери та модулі обмеження доступу, використовуючи ASP.NET Core, ASP.NET Identity, EF Core, MySQL та Razor Views.

Розроблена система для контролю та керування доставками з використанням ASP.NET Core може використовуватись у сфері поштово-кур'єрських послуг та логістики.

КЛЮЧОВІ СЛОВА: ВЕБ-ДОДАТОК, ASP.NET CORE, ВІДПРАВЛЕННЯ, ТРЕКІНГ, ДОСТАВКА, ОТРИМУВАЧ, ЧАТ, ВІДДІЛЕННЯ.

ЗМІСТ

ВСТУП

Актуальність теми: в умовах стрімкого розвитку електронної комерції та великої кількості бізнес-процесів, що вимагають цифровізації, роль інформаційних технологій, які забезпечують ефективне управління процесами логістики, зростає. Системи доставки посилок стають невід'ємною частиною функціонування для інтернет-магазинів, торговельних платформ або інших сервісів, які пов'язані з перевезенням товарів. У нашій країні активно розвивається ринок логістичних послуг, представниками якого є Нова пошта, Укрпошта та інші[1, 2]. Але, існуючі системи мають доволі обмежений функціонал або переважно орієнтуються на комерційне використання, без можливості гнучкого налаштування системи. Також кінцеві користувачі очікують від системи не лише базовий функціонал відправлення та отримання посилок, а й додаткових можливостей, відстеження маршруту доставки, інтерактивні мапи, бонусна політика, можливість комунікувати з кур'єром в системі, мульти трекінг. Наразі необхідність є створення систем, що поєднують в собі функціонал управління доставкою, трекінг посилок, елементи гейміфікації, які будуть виступати стимулятором для постійного використання додатку. Таким чином, проектування системи управління доставками посилок є актуальним завданням. Це має зменшити навантаження на працівників, покращення взаємодії з кінцевим користувачем та автоматизацію основних процесів.

Об'єкт дослідження є процеси управління, відстеження поштових відправлень та методи обміну повідомленнями в інформаційних системах.

Предмет дослідження є програмне забезпечення для трекінгу посилок та управління доставками посилок та можливістю комунікувати з кур'єром всередині додатку.

Метою роботи є автоматизація процесів створення, обробки, відстеження та контролю доставки посилок, а також підвищення ефективності взаємодії між користувачами, кур'єрами та адміністраторами системи за допомогою сучасних веб-технологій.

Для досягнення мети потрібно виконати поставленні завдання перелічені нижче:

1. Провести аналіз інформаційних систем доставки посилок.
2. Дослідити функціональні можливості існуючих систем доставки.
3. Визначити основні вимоги до додатку.
4. Розробити базу даних.
5. Спроектувати архітектуру програмного забезпечення.
6. Розробити веб-додаток, використовуючи ASP.NET Core, що містить в собі функціонал створення, редагування, масової зміни статусів, логування зміни статусів, чат між отримувачем та кур’єром, генерацію накладних, трекінг посилок, місій для отримання бонусів, призначення транспорту, зміну ролей користувачів
7. Для оцінки ефективності протестувати розроблену систему.

Методами дослідження є аналіз джерел у сфері електронної комерції, існуючих інформаційних системи доставки, методи виявлення поточного місцезнаходження, способи обміну повідомленнями в реальному часі, методи об’єктно орієнтованого програмування, методи проектування баз даних, методи розробки веб-додатків, методи тестування програмного забезпечення, використовуючи ASP.NET Core[3].

Практичною новизною роботи є поєднання методів контролю логістичних процесів із технологіями відстеження місцезнаходження та інтерактивній взаємодії між користувачами в межах системи. У межах бакалаврської роботи запропоновано підхід, що забезпечує повне управління процесами доставки шляхом інтеграції обробки координат користувачів, обробки статусів посилок та використання технології обміну повідомленнями в реальному часі для комунікації кур’єра з отримувачем. Окрім цього, було удосконалено механізм відстеження відправлення, який надає змогу не лише отримувати інформацію про статус посилки, а й візуалізувати її приблизне переміщення

на інтерактивній карті враховуючи поточне місцезнаходження транспорту, що робить процес доставки прозорішим та більш інформативним для клієнта.

Практична значущість результатів полягає в створенні прототипу веб-додатку, що може використовуватись службами доставки посилок для автоматизації процесів управління та відстеження посилками. Розроблений додаток дозволить підвищити рівень ефективності обробки посилок та зменшити навантаження на операторів відділень через автоматизацію процесів обробки замовлень, що підвищить якість обслуговування клієнтів. Крім цього, додаток забезпечує зручність для користувача за рахунок комунікації з кур'єром напряму та робить трекінг посилки більш інформативним.

Аналіз підтверджує актуальність розробки сучасних інформаційних систем для управління доставкою посилок, які поєднують функції відстеження, автоматизацію логістичних процесів та інтерактивну комунікацію між користувачами. Запропонований підхід дозволяє підвищити ефективність обробки відправлень, поліпшити якість обслуговування клієнтів та забезпечити прозорість процесу доставки. Впровадження таких систем сприятиме подальшому розвитку логістичних послуг та їх адаптації до сучасних вимог цифровізації.

Апробація результатів та публікації. Основні положення і результати бакалаврської роботи доповідались на наступних науково-практичних конференціях, що проходили на базі Державного університету інформаційно-комунікаційних технологій:

1. Паламарчук М.О., Герцюк М.М. Методи відстеження посилок та комунікації в реальному часі у веб-сервісах доставки. // VII Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях».(Подано до друку) 20.04.2026

2. Паламарчук М.О., Герцюк М.М. Цифровий веб-сервіс для контролю та керування доставками посилок на основі ASP.NET Core.// Всеукраїнська науково-технічна конференція «Технології цифрового розвитку: програмні системи та децентралізація».(Подано до друку) 25.04.2026

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Відповідно до об’єкту дослідження, розглянемо процеси управління та відстеження поштових відправлень.

Результатом даних процесів є відстеженння та отримання посилки кінцевим користувачем. Проблема полягає в тому, що існуючі системи в Україні не надають можливості повної інформації про поточне розташування їхнього відправлення. Зазвичай, функціонуючі додатки дають можливість отримати лише умовне місцезнаходження посилки при доставці між відділеннями чи кур’єром, наприклад, “Ваше відправлення виїхало з терміналу №2 міста Київ” та показує ще велику кількість наступних зупинок на шляху до кінцевої точки доставки, що не дає користувачу точного розуміння де саме зараз знаходиться їхня посилка. Така ситуація спостерігається в таких транспортних компаніях гігантах Нова пошта, Укрпошта та Meest Express[1, 2, 6].

Для вирішення цієї проблеми дійшли до висновку, що є необхідність в розробці системи яка допоможе відслідковувати місцезнаходження відправлення за допомогою GPS-координат з мінімальними затримками при транспортуванні посилок між відділеннями та надасть можливість домовлятись з кур’єром про доставку всередині розробленої системи для уникнення непорозумінь. Для досягнення поставленої мети потрібно реалізувати наступні аспекти системи:

– відстеження місцезнаходження посилки, необхідно проаналізувати методи отримання та обробки координат для інтеграції їх в карту маршруту доставки;

– обмін повідомленнями в реальному часі, знайти спосіб для ефективного обміну повідомленнями між клієнтом та кур’єром напряму;

– аналіз аналогічних систем, визначити переваги та недоліки існуючих додатків;

– специфікація вимог до системи, на основі аналізу конкурентів потрібно сформулювати вимоги до продукту, які будуть задовольняти потреби бізнесу та кінцевих користувачів;

– розробка веб-додатку, визначити оптимальний набір технологій для проектування архітектури програмного забезпечення.

Отже, реалізація даних аспектів допоможе комплексно підійти до розробки системи управління та контролю доставками

1.2 Методи відслідковування поточного місцезнаходження

Одним з варіантів відстеження місцезнаходження у веб-застосунках є визначення місцязнаходження по IP-адресі пристрою, що має за основу базу даних, в яких IP-адреси співвідносяться з певними географічними регіонами, містами або провайдерами.

Під час встановлення IP-з'єднання сервер веб-сайту реєструє IP-адреси відвідувачів

Спеціалізоване програмне забезпечення для відстеження IP-адрес порівнює ці IP-адреси з власною базою даних зареєстрованих IP-адрес.

У разі збігу інструменти ідентифікації користувачів та можуть надати його місцезнаходження. Також можна дізнатися, які сторінки вони переглядали та скільки часу провели на кожній із них

Після ідентифікації та реєстрації IP-адреси IP-трекер може доповнити ці дані додатковою інформацією.

Деякі IP-адреси містять більше даних про місцезнаходження, ніж інші. Наприклад, статичні IP-адреси не змінюються, тому ви можете бути впевнені в надійності даних про місцезнаходження. Але для динамічних IP-адрес дані про місцезнаходження не завжди будуть правильними на 100%. Проте це може вказати приблизне місцезнаходження пристрою, пов'язаного з цією IP-адресою, і зазвичай є правильним на рівні штату або округу.

Більш досконалі інструменти відстеження IP можуть використовувати IP-адреси для надання ще більш цінних даних, зіставляючи IP-адреси з базами даних, що містять відповідну інформацію[7].

Основні переваги використання цього методу:

– простота реалізації, так як не потрібен доступ до GPS користувача, дуже просто інтегрувати цей метод та використовувати в системі;

– низьке навантаження на пристрій, даний метод не використовує ресурси самого девайсу, тому продуктивність додатку не знизиться;

– швидке отримання результату визначення IP-адреси девайсу відбувається майже одразу, без необхідності сигналу від супутників.

Не дивлячись на те, що даний підхід є не поганим рішенням є низка недоліків через які не можна використати цей метод у системі:

– низька точність, так як даний спосіб надає лише можливість отримати дані про регіон користувача, без конкретних координат;

– непостійність, в залежності від типу підключення до мережі регіон користувача може змінюватись, що призводить до похибок у визначенні місцезнаходження.

Більш підходящим способом є використання GPS-трекерів. GPS-трекер – це пристрій для дистанційного моніторингу розташування об’єктів в реальному часі. Перевагами цього методу є:

– трекінг локації в реальному часі, допомагає моніторити рух транспорту, його швидкість та зупинки на мапі;

– автономність, трекери можуть працювати від акумулятора або можна підключити до мережі автомобіля;

– історія поїздок, зберігає маршрут переміщення, що дуже корисно для збору аналітики.

Але є й недолік використання GPS-трекера – висока вартість обслуговування, так як потрібно постійно купувати підписки для використання розширеного функціоналу.

Ще одним методом трекінгу місцезнаходження є геомаяк що працює як

маркер місцезнаходження. Це пристрій, що випромінює сигнал BLE, який використовують пристрої відстеження для визначення свого місцезнаходження.

Отже, коли ви встановлюєте геомаяки у відомих місцях, маяк транслює свій ідентифікатор. Пристрій відстеження, що знаходиться поблизу, розпізнає джерело сигналу – геомаяк – і використовує цю інформацію для визначення свого місцезнаходження.

Перевагою геомаяків є те, що для зв'язку не потрібна взаємодія з існуючою IT-інфраструктурою. Сам маяк є мінімальною інфраструктурою, яку необхідно встановити.

Ця технологія геолокації працює точно як на відкритому повітрі, так і в приміщенні. Чим більше маяків встановлено, тим точнішим буде визначення місцезнаходження.

Маяк на батарейках є набагато дешевшим, ніж прокладення кабелів у промислових зонах, для чого, можливо, доведеться зупиняти виробничі процеси та проводити перевірки безпеки.

Недоліком є те, що він дещо менш точний, ніж GPS, і вимагає мінімальної інфраструктури, тоді як GPS її не потребує[8].

У випадку коли не має можливості використати GPS-трекера, IP-адресу, на допомогу приходить отримання геолокації через API браузера.

За допомогою вбудованих методів у JavaScript можна отримати координати користувача. Використовуючи GPS для отримання даних в реальному часі

Для успішного отримання координат необхідно отримати згоду користувача для надання доступу до GPS

Використання браузерного API не потребує складних інтеграцій і надає достатньо точну інформацію про поточне розташування, тому після запису даних в базу даних можна буде спокійно вивести інформацію на карту[9].

Але і такий метод має недоліки. Користувач може спокійно підмінити свої координати, що призведе до неможливості відслідкувати водія, без використання додаткових платних сервісів, що будуть блокувати підміну координат. Також для використання цього методу потрібно аби пристрій водія був активним та

використовував наш веб-додаток, що призводить до наявності можливості підзарядити девайс, аби постійно бути в мережі для оновлення даних на карті

1.3 Методи комунікації в реальному часі

Методи комунікації в режимі реального часу відіграють вирішальну роль у сучасних інформаційних системах, оскільки забезпечують швидку передачу даних між користувачами, серверами та пристроями. На відміну від традиційної моделі обміну даними, за якої клієнт періодично надсилає запити на сервер, технології реального часу дозволяють отримувати нову інформацію миттєво, щойно вона стає доступною. Це особливо важливо для систем, які містять в собі елементи месенджерів.

Одним із найпоширеніших методів є Polling – браузер періодично запитує сервер через встановлені проміжки часу. Його перевагою є простота реалізації, але недоліками є затримки між оновленнями, підвищене навантаження на сервер та неефективне використання мережевих ресурсів, особливо якщо нові дані з’являються нечасто[10].

Удосконаленою версією є Long Polling, у якій сервер тримає з’єднання відкритим, доки не з’явиться нова інформація, після чого надсилає відповідь клієнту. Це зменшує кількість непотрібних запитів і наближає обмін даними до режиму реального часу. Однак цей підхід все ще створює додаткове навантаження на сервер, коли існує велика кількість одночасних з'єднань[10].

Більш сучасним рішенням є WebSocket, який забезпечує двосторонній, постійний канал зв'язку між клієнтом і сервером. Після встановлення з'єднання обидві сторони можуть надсилати дані одна одній без необхідності створювати новий HTTP-запит щоразу. Це значно зменшує затримку і робить WebSocket ефективним для чатів, онлайн-ігор, систем відстеження та сповіщень у реальному часі[10].

Мій додаток для спілкування в режимі реального часу використовує SignalR. Ця технологія застосовується для реалізації функції миттєвого обміну повідомленнями між кур’єром та отримувачем[11].

1.4 Аналіз аналогічних додатків

Виходячи з мети та завдань поставлених вище було проаналізовано аналогічні застосунки, а також існуючі рішення у сфері логістики та відстеження посилок. Основним аналогом подібної системи є Укрпошта[2].

1.4.1 Аналіз платформи Укрпошта

Цей додаток дозволяє користувачам надсилати посилки та відстежувати її переміщення за допомогою номера відстеження(рис.1.1).

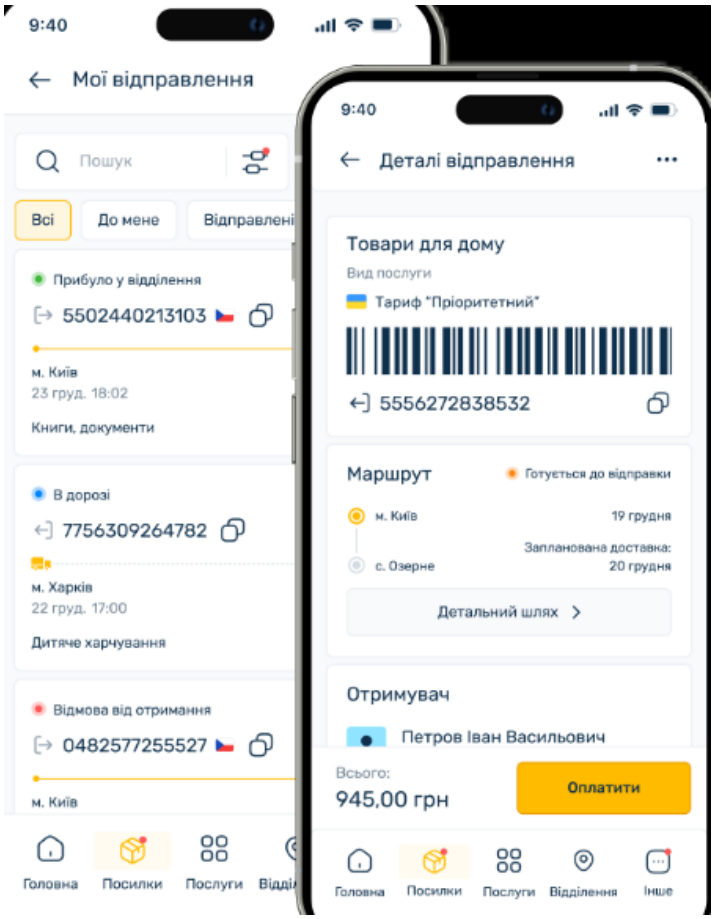


Рис. 1.1 Інтерфейс Укрпошти

На сайті реалізована система авторизації та аутентифікації користувачів. Наявний персональний кабінет у якому можна редагувати свої дані(рис. 1.2) та підключити систему лояльності (рис. 1.3)

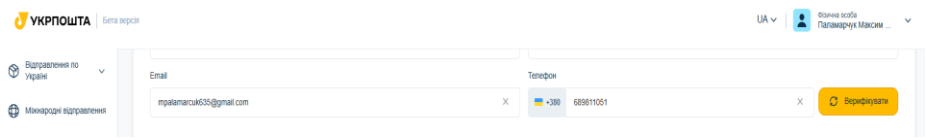


Рис. 1.2 Персональний кабінет

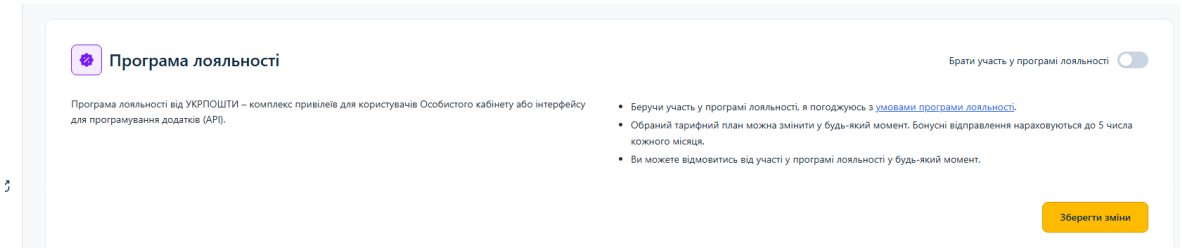


Рис. 1.3 Програма лояльності

На сторінці відправлень відображається загальна інформація про посилки Номер відправлення, Статус, Інформація про отримувача, та дії які можна робити з відправленнями(перегляд, редагування) (рис. 1.4). З цієї сторінки можна Створити посилку абу групу посилок за певною ознакою, дізнатися про тарифи компанії та іншу корисну інформацію

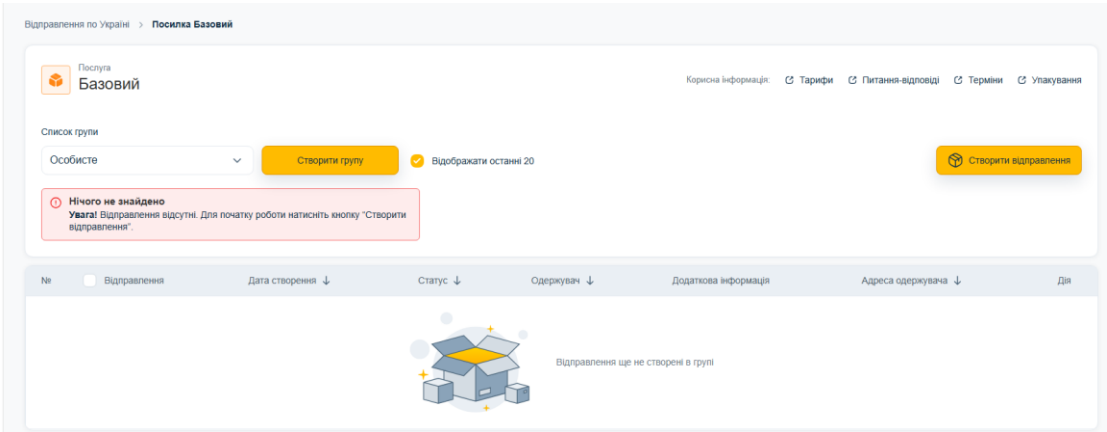


Рис. 1.4 Відправлення по Україні

На веб-сайті доступна система відстеження, яка дозволяє користувачам отримувати інформацію про статус, місцезнаходження та історію переміщення посилки. Користувачі можуть ввести номер для відстеження, щоб переглянути детальну інформацію про проходження посилки через різні сортувальні центри та відділення(рис. 1.5).

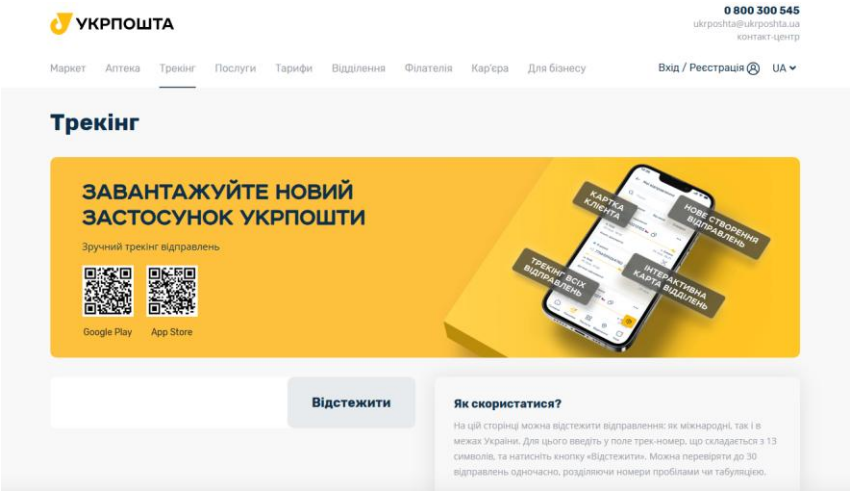


Рис. 1.5 Сторінка трекінгу

Хоч здається сервіс спроектований досить непогано, через те, що кожен окремий сервіс відкривається під новим доменом призводить до поганого UX через постійні оновлення сторінок та купу анімацій, які навантажують пристрій, Окрім цього для створення посилки користувачу потрібно заповнити занадто багато початкової інформації, крім цього без створення групи, не має можливості додати відправлення.

До того ж виникають питання у коректності роботи навігації на веб ресурсі, так як при спробі зареєструватись відбувається редірект на 404 сторінку.

У наступній таблиці представлено опис, та основні переваги та недоліки проаналізованого сервісу.

1.4.2 Аналіз платформи Нова пошта

Нова пошта – одна з найбільших логістичних компаній в Україні, що надає широкий асортимент послуг. Компанія орієнтована як на звичайного користувача

так і на підприємців, надаючи швидку обробку посилок та інтеграцію з інтернет-магазинами[1].

Таблиця 1.1

Аналогічний додаток Укрпошта

Назва	Укрпошта
Архітектура	3-tier web application
Виробник	Державне підприємство
Мова реалізації	Java, JavaScript
Функції	1. відстеження посилки за трек-номером; 2. перегляд статусів доставки; 3. оформлення відправлення; 4. особистий кабінет; 5. історія переміщень.
Переваги	1. доступність по всій території України; 2. простота використання; 3. базовий функціонал відстеження; 4. програма лояльності.
Недоліки	1. відсутність карти з реальним місцезнаходженням; 2. відсутність мульти трекінгу; 3. відсутність оновлення у реальному часі; 3. застарілий інтерфейс; 4. відсутність комунікації з кур’єром; 5. відсутність мапи існуючих відділень.
Веб-сайт	https://www.ukrposhta.ua/

На сайті Нової пошти є можливість відслідкувати посилку за трекінговим номером, що дає користувачу інформацію про поточний статус відправлення. Подається у вигляді хронологічній послідовності з моменту створення

відправлення до поточного стану. Це дозволяє користувачам отримати загальне уявлення про місцезнаходження відправлення.(рис. 1.6)

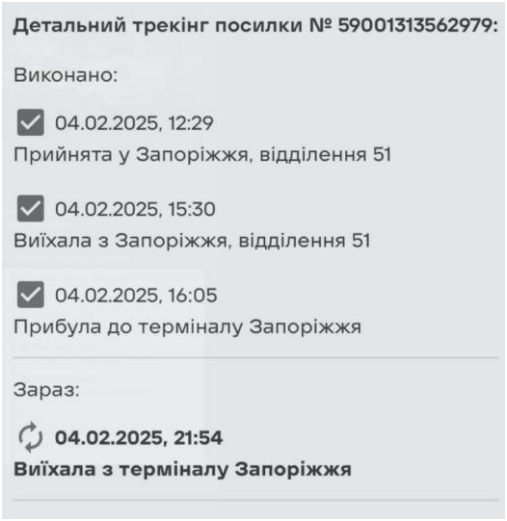


Рис. 1.6 Трекінг посилки в Нова пошта

Однією з переваг сервісу є наявність особистого кабінету, який містить в собі низку функцій: історія відправлень, отримання сповіщень про зміну статусу посилок, управління адресами, накладні, що доволі зручно для користування.(рис. 1.7)

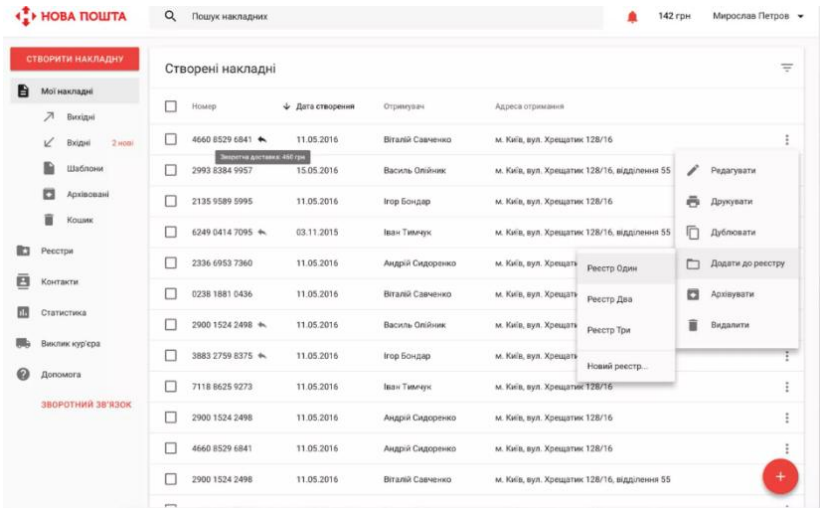


Рис. 1.7 Особистий кабінет в Нова пошта

Також, сервіс тісно співпрацює з інтернет-магазинами через API, що допомагає e-commerce магазинам автоматизувати процеси створення та обробки замовлень. Наразі такий підхід є дуже поширеним в сфері логістики так як це підвищує ефективність обробки замовлень

У наступній таблиці представлено опис, та основні переваги та недоліки проаналізованого сервісу.

Таблиця 1.2

Аналогічний додаток Нова пошта

Назва	Нова пошта
Архітектура	3-tier web application
Виробник	ТОВ «Нова пошта»
Мова реалізації	C#, JavaScript
Функції	1. відстеження посилки за трек-номером; 2. перегляд статусів доставки; 3. оформлення відправлення онлайн; 4. особистий кабінет; 5. push, SMS сповіщення.
Переваги	1. доступність по всій території України; 2. сучасний інтерфейс; 3. базовий функціонал відстеження 4. push, SMS сповіщення.
Недоліки	1. відсутність карти з приблизним місцезнаходженням; 2. відсутність мульти трекінгу 3. відсутність оновлення у реальному часі; 4. відсутність чату з кур’єром.
Веб-сайт	https://novaposhta.ua/

Аналіз існуючих аналогічних систем дозволив виявити їхні сильні та слабкі сторони, що дозволяє розробити додаток з урахуванням переваг та недоліків існуючих систем.

2 АНАЛІЗ ЗАСОБІВ РЕАЛІЗАЦІЇ ТА СТРУКТУРИ ПРОГРАМНОЇ СИСТЕМИ

2.1 Специфікація вимог до ПЗ

Веб-застосунок, що розробляється являє собою систему відстеження, збереження даних про посилки користувачів та комунікацію отримувача з кур’єром. Сервіс призначений для користувачів, що хочуть відправити, відстежити посилку на мапі або зв’язатись з кур’єром без дзвінків суто в чаті.

Основні ролі в додатку:

– Customer – звичайний користувач, який може створювати посилки та переглядати інформацію про них, завантажити накладну, зв’язатись з кур’ром в чаті якщо у посилки відповідний тип доставки, відслідкувати посилку по трек-номеру на мапі, також має доступ до програми лояльності, у вигляді виконання місій за які отримує бонуси і використовує в майбутньому;

– Admin – має більш розширені можливості ніж Customer, а саме, може редагувати посилки, масово змінювати статус посилок, призначати водія для транспорту, надавати ролі користувачам;

– DeliveryAgent – має доступ до блоку для кур’єрів, що включає в себе можливості взяти посилку собі для доставки та комунікації з клієнтом за допомогою чату;

– InterBranchDriver – має доступ до присвоєння собі транспорту та перегляду інформації про посилки, які є в його автомобілі, та шерінгу своїми координатами для інтерактивної мапи.

Функціональні вимоги до програмного забезпечення:

– функціонал авторизації користувачів за допомогою емейлу та паролю;

– функціонал створення, редагування, видалення посилки;

– функціонал логування зміни статусів посилки;

– функціонал пошуку посилки за трекінговим номером;

– функціонал сортування посилки по трекінговому номеру, статусу;

– функціонал перегляду повної інформації про посилку;

– функціонал генерації накладної;

– функціонал трекінгу однієї посилки;

– функціонал мульти трекінгу;

– функціонал обміну повідомленнями з кур’єром в середині додатку;

– функціонал масової зміни статусів;

– функціонал вибору машини для водія;

– функціонал отримання та застосування бонусів з програми лояльності;

– функціонал відображення інтерактивної карти з приблизним маршрутом посилки та розташуванням водія;

– функціонал відображення активних відділень на мапі;

– функціонал отримання та зберігання координат водія;

– система повинна давати можливість кур’єру взяти посилку для доставки;

– функціонал мануальної зміни ролі користувача;

– наявність персонального кабінету користувача.

Вимоги до дизайну інтерфейсу веб-сайту:

– інтерфейс має бути інтуїтивно зрозумілим та зручним у користуванні, а розташування елементів має забезпечувати комфортну взаємодію користувача з функціоналом системи;

– інтерфейс має бути адаптивним і коректно відображатися на сучасних пристроях, у різних браузерах та на екранах різних розмірів.

Вимоги до надійності та безпеки системи:

– необхідно передбачити обробку всіх можливих помилок та виняткових ситуацій, які можуть призвести до некоректної роботи або збою системи;

– доступ до API повинен бути безпечним і реалізованим з використанням диференціації прав доступу на основі ролей користувачів;

Сформовані вимоги до додатку визначають обмеження, характеристики та функціональні можливості системи, що забезпечують ефективне використання додатку.

2.2 Середовище розробки

Для створення web-додатку для системи управління посилками було обрано середовище розробки Visual Studio – професійне середовище від компанії Microsoft, що є одним із кращих рішень для розробки додатків мовою C#.

Дане середовище надає широкий спектр можливостей роботи з бібліотеками та фреймворками для розробника. Також має інструменти для дебагінгу та систему скорочення ключових слів, що пришвидшує проектування додатку.

Visual Studio підтримує роботу з поширеними бібліотеками C#: ASP.NET Core, ASP.NET Identity, EF Core, LINQ, та систему контролю версій Git [12, 13, 14].

Нижче наведені основні переваги Visual Studio для виконання цієї роботи:

– даний інструмент забезпечує повну інтеграцію з платформою .NET, так як даний інструмент підтримується компанією Microsoft;

– проста генерація проекту ASP.NET Core за допомогою готових щаблонів;

– інтелектуальне автодоповнення коду завдяки IntelliSense;

– вбудований SQL-редактор. Можна легко працювати з базами даних MySQL із середовища, переглядати таблиці, виконувати запити та налагоджувати схему БД;

– дебагінг, у реальному часі можна запустити додаток у режимі дебагу проставивши перед цим місця зупинки додатку, що дає змогу побачити які дані потрапляють в змінні.

Середовище Visual Studio є одним з найефективніших середовищ для розробки веб-додатків, використовуючи C#[12].

2.3 Вибір мови програмування та фреймворку

Для реалізації backend-частини додатку було прийнято рішення використовувати мову програмування C#, що містить в собі фреймворк ASP.NET Core. C# є однією з найпопулярніших мов для проектування веб-додатків, так як

проекти написані цією мовою просто масштабувати та вони є стабільними. Мова підтримує об’єктно орієнтовану парадигму, має сувору типізацію та має цілу екосистему бібліотек, що надає можливість створювати додатки з складною бізнес-логікою.

Фреймворк ASP.NET Core значно прискорює розробку веб-додатків завдяки вбудованим механізмам маршрутизації, підтримці шаблонів проектів, інтеграції з ін’єкцією залежностей та вбудованому веб-серверу Kestrel. Ця платформа дозволяє швидко створювати API, організовувати взаємодію з базами даних за допомогою Entity Framework Core, реалізовувати аутентифікацію та авторизацію користувачів через ASP.NET Identity, а також будувати архітектуру «клієнт-сервер».

У рамках цього проекту використання C# та ASP.NET Core дозволяє реалізувати серверну логіку системи відстеження посилок, обробку запитів користувачів, взаємодію з базою даних та контроль доступу до функціоналу системи. Інструменти платформи .NET дозволяють дотримуватися принципів \чистої архітектури, підвищують продуктивність розробки та спрощують подальше обслуговування програмного забезпечення.

Поєднання C# та ASP.NET Core є ефективним рішенням для створення надійного, масштабованого та багатофункціонального веб-додатку, що відповідає сучасним вимогам до інформаційних систем.

2.4 Реалізація безпеки системи

Для захисту чутливих даних про користувачів в застосунку використано бібліотеку ASP.NET Identity, яка містить в собі комплексну реалізацію захисту додатку. Механізмом автентифікації є логін та пароль. Логін прирівнюється до e-mail користувача оскільки він є унікальним для кожного користувача. Паролі зберігаються у захешованому вигляді, що підвищує безпеку системи при хакерській атаці.

Додаток містить в собі систему ролей, що дозволяє при авторизації надати доступ до необхідного функціоналу для водія(InterBranchDriver), звичайного

користувача(Customer), кур’єра(DeliveryAgent) та адміністратора(Admin), обмеживши доступ до забороненого функціоналу.

Крім того, ASP.NET Core Identity разом із вбудованими функціями платформи ASP.NET Core реалізує захист від CSRF-атак шляхом автоматичного додавання токенів захисту від підробки до форм. Система також забезпечує управління сеансами користувачів, включаючи контроль часу очікування сеансу та можливість налаштування політик безпеки входу.

Використання ASP.NET Core Identity відповідає сучасним вимогам безпеки веб-додатків, забезпечуючи надійний захист облікових записів користувачів та контроль доступу до системних ресурсів на всіх рівнях взаємодії.

2.5 Робота з базою даних

У розроблюваному проекті в якості системи управління базами даних використовується Microsoft SQL Server – реляційна система управління базами даних, відома високим рівнем надійності, продуктивності та масштабованості. Вона забезпечує підтримку транзакцій, контроль цілісності даних, механізми індексації та ефективну обробку запитів, що є надзвичайно важливим для систем, призначених для обробки великих обсягів даних.

Взаємодія з базою даних реалізована за допомогою технології Entity Framework Core, яка слугує шаром об'єктно-реляційного відображення. Такий підхід дозволяє працювати з даними як з об'єктами C#, автоматизувати основні операції з базою даних та зменшити необхідність ручного написання SQL-запитів.

Для розробки та запуску програмного забезпечення локально використовується Microsoft Visual Studio; вона надає інструменти для управління проектами, налагодження та інтеграції з базою даних. Зокрема, вона дозволяє здійснювати міграцію баз даних, налаштовувати з'єднання та керувати структурою бази даних.

Адміністрування бази даних здійснюється за допомогою SQL Server Management Studio – спеціалізованого програмного інструменту з графічним

інтерфейсом, що дозволяє створювати та модифікувати таблиці, переглядати та редагувати дані, виконувати SQL-запити та створювати резервні копії.

Таким чином, поєднання SQL Server, Entity Framework Core, Visual Studio та SQL Server Management Studio забезпечує ефективне середовище для зберігання, обробки та управління даними, яке відповідає вимогам сучасних веб-додатків.

2.6 Клієнтська частина та використання шаблонів

При розробці інтерфейсу використовувався HTML, CSS, JavaScript та Bootstrap. Для передачі динамічних даних використовувався вбудований в C# шаблонізатор Razor, який дозволяє:

– рендерити HTML сторінки на сервері;

– перевикористовувати вже створені компоненти;

– зручно використовувати C# та HTML в комбінації для відображення даних, які приходять з серверної частини;

– помітити помилки в коді ще на етапі компіляції.

Це дає змогу реалізувати інтуїтивно зрозумілий інтерфейс з інтерактивними елементами використовуючи простий JavaScript[15].

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ЗАСТОСУНКУ

3.1 Архітектура розробленого додатку

Розроблений веб-додаток для управління доставками було розроблено за стандартною архітектурою “клієнт-сервер”. Слідуючи цьому принципу було чітко розділено обов’зки між інтерфейсом користувача, рівнями доступу до системи та бізнес-логікою продукту

Додаток містить в собі такі логічні рівні:

Серверний рівень(backend) – розроблено з використанням C# та фреймворку ASP.NET Core, за допомогою якого зручно створювати веб-додатки, реалізовуючи бізнес-логіку, обробку запитів та взаємодію з базою даних. У даному рівні було розроблено авторизацію, автентифікацію, модуль бонусної політики, кастомізацію інформаційних сторінок, кур’єрський модуль і т. д.

База даних(data access layer) – за основу якого, використано реляційну СУБД SQL Server Management Studio, що інтегрована з Entity Framework Core – технології об’єктно реляційного відображення(ORM). Даний підхід дозволяє оперувати сутностями використовуючи C#, а не SQL-запити. Користуючись цим фреймворком можна зменшити час розробки, та зменшити кількість помилок при взаємодії з базою даних[16].

Клієнтський рівень(frontend) – розроблений за допомогою HTML, CSS, JavaScript та Bootstrap. Також було використано вбудований шаблонізатор C# Razor для рендерингу сторінок на сервері, що дає змогу передавати C# об’єкти у вигляді HTML. Так як клієнтський рівень відповідає за взаємодію з користувачем Razor значно спрощує побудову цього рівня.

Взаємодія між клієнтським та серверним рівнями відбувається за допомогою HTTP-запитів та Websocket, які обробляються на сервері контроллерами та повертають відповідь у вигляді JSON або View(HTML-сторінок), які потім отримує клієнтський рівень.

Для безпеки використовувалася бібліотека ASP.NET Core Identity для реєстрації, авторизації, автентифікації користувачів, захисту їх сесій та інтерпретації паролів у хешований вигляд.

Додаток побудовано на багаторівневій архітектурі, що забезпечує чіткий розподіл функцій між компонентами системи. Такий підхід спрощує обслуговування програмного забезпечення, підвищує читабельність коду та дає змогу розширювати функціонал системи без істотного впливу на інші його частини.

3.2 Сценарії взаємодії користувача із системою

У рамках розробки веб-додатку цифрової системи управління та трекінгу посилками було створено діаграму сценаріїв взаємодії користувача із застосунком.

З метою підвищення читабельності загальну діаграму та їх опис було розділено на декілька логічно завершених піддіаграм, кожна з яких відображає окрему функціональну підсистему.

3.2.1 Основні операції користувача

На рисунку 3.8 наведено діаграму сценаріїв, що описує основні етапи взаємодії користувача з системою.

Користувач має змогу ознайомитися з ціновою політикою компанії, після цього створити відправлення вказавши вагу, кінцеву точку відправлення у вигляді відділення, отримувача, обрати тип доставки, та вказати домашню адресу отримувача якщо тип доставки Кур’єром. Після успішного створення посилки користувач може переглядати деталі про відправлення.

Окрім цього, користувач може згенерувати накладну після створення посилки аби отримати повну інформацію про відправлення.

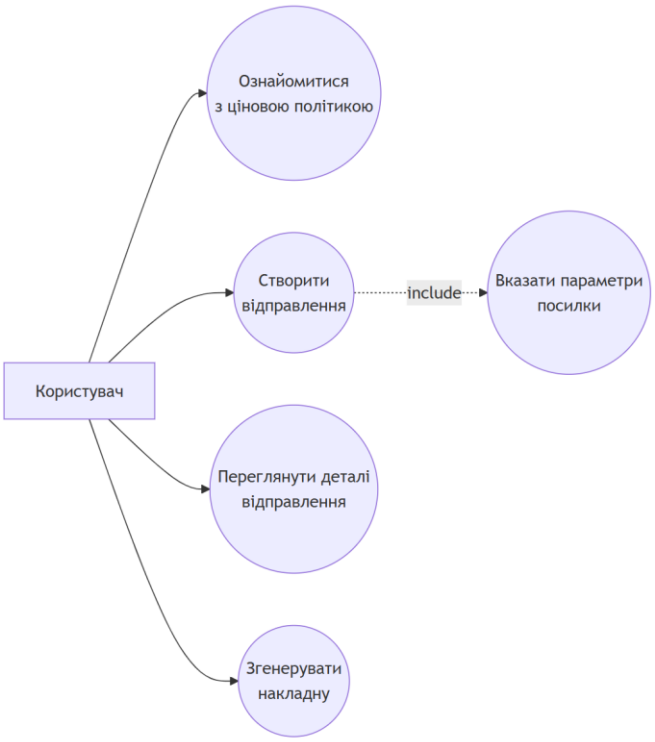


Рис. 3.8 Діаграма основних сценаріїв використання додатку для користувача

3.2.2 Трекінг посилок та карта

На рисунку 3.9 наведено діаграму сценаріїв, що описує процес відстеження посилки.

Рис. 3.9 Діаграма сценаріїв використання трекінгу для користувача

Також користувач може перейти на сторінку трекінгу, обрати тип трекінгу одна чи кілька посилок та ввести номер посилки, після цього може отримує доступ до інтерактивної карти, на якій відображається приблизний час доставки, приблизний маршрут доставки та поточне місцезнаходження відправлення.

3.2.3 Сценарії використання чату отримувачем

На рисунку 3.10 зображено сценарій використання чату отримувачем.

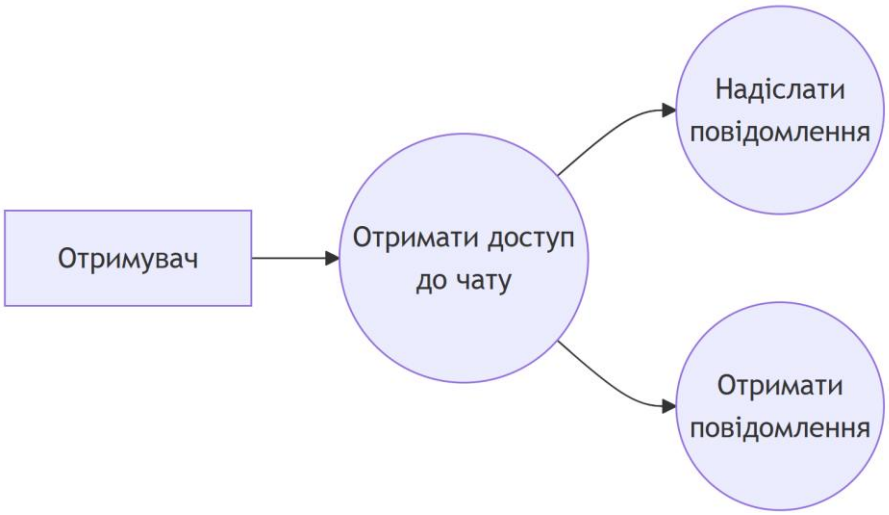


Рис. 3.10 Діаграма сценаріїв використання чату користувачем

Якщо у отримувача є відправлення з типом доставки Кур’єр, він отримує доступ до чату, в якому може спілкуватись з кур’єром, про приблизний час нюанси доставки

3.2.4 Сценарії використання профілю та бонусної системи користувачем

На рисунку 3.11 наведено сценарії, пов’язані з управлінням профілем користувача та бонусною системою.

Додатково користувач отримує доступ до особистого кабінету, в якому може переглядати та редагувати особисті дані ім’я, прізвище, пошту, змінити свій пароль та обов’язково має обрати найближче до нього відділення перед тим як створювати відправлення.

Останнім сценарієм використання системи для користувача є перехід на сторінку бонусів та ознайомитись з бонусами які реалізовані у вигляді місій, які він може отримати виконавши необхідні умови(Отримати знижку на доставку, безкоштовну доставку між відділеннями, безкоштовну кур’єрську доставку).

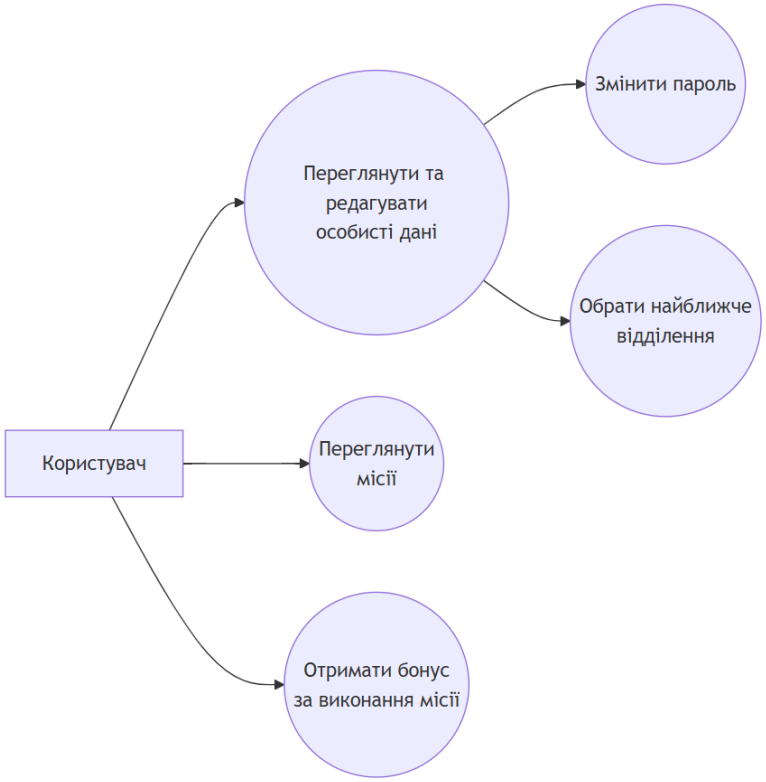


Рис. 3.11 Діаграма сценаріїв взаємодії з профілем та бонусами

3.3 Сценарії взаємодії адміністратора із системою

У цьому розділі розглядаються основні сценарії використання системи з точки зору адміністратора. Відповідна діаграма випадків використання ілюструє взаємодію між адміністратором та підсистемами веб-додатку, що відповідають за управління користувачами, відправками, інформаційними сторінками.

З метою підвищення читабельності загальну діаграму та їх опис було розділено на декілька логічно завершених піддіаграм, кожна з яких відображає окрему функціональну підсистему.

Діаграма дозволяє краще зрозуміти обов’язки, можливості та залученість адміністратора в процеси управління. Що є важливим при реалізації функціоналу доступного для адміністратора та модулю автентифікації.

3.3.1 Сценарії управління користувачами та посилками

На рисунку 3.12 наведено основні сценарії, пов’язані з управлінням користувачами та посилками

Рис. 3.12 Діаграма сценаріїв управління користувачами та посилками

Адміністратор має доступ до перегляду всіх зареєстрованих користувачів та зміни їхніх ролей. Також має доступ до управління посилками: їх створення, редагування(наприклад змінити інформацію про відправника, отримувача чи змінити статус посилки), перегляду інформації, генерації накладної та за потреби може видалити їх.

3.3.2 Сценарії масової зміни статусів і призначення транспорту

На рисунку 3.13 подано сценарій масової зміни статусів посилок.

Адміністратор має доступ до масової зміни статусів посилок, при чому може призначити автомобіль, який буде перевозити відправлення між відділеннями якщо статус на який планують змінити рівний “в дорозі”.

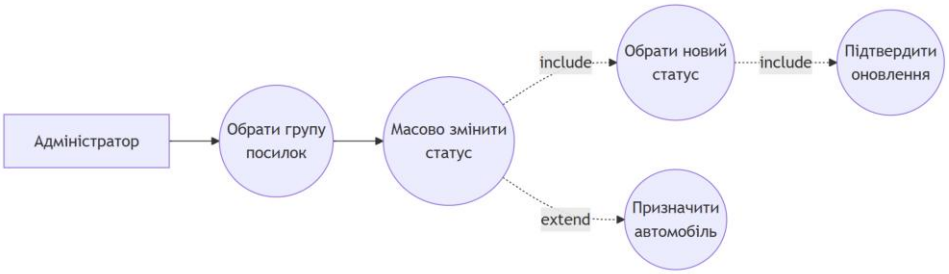


Рис. 3.13 Сценарій масової зміни статусів посилок

3.3.3 Сценарії управління сторінками

На рисунку 3.14 представлено сценарії використання модуля наповнення інформаційних сторінок контентом.

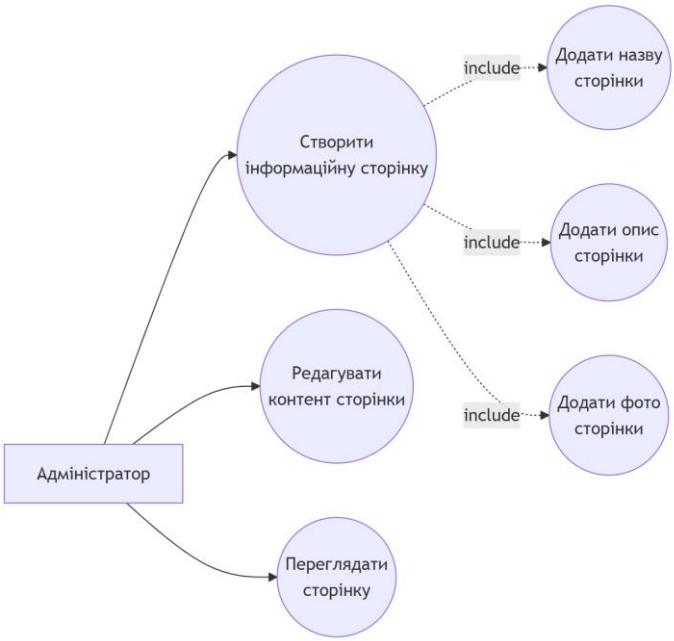


Рис. 3.14 Сценарій наповнення інформаційних сторінок контентом

Ще одним сценарієм використання системи адміністратором є використання модулю наповнення інформаційних сторінок контентом. Він має можливість додати назву сторінки її опис та фото сторінки, для коректного відображення контенту для звичайних користувачів.

3.3.4 Сценарії використання особистого кабінету та додаткових функцій

На рисунку 3.15 подано сценарії, пов’язані з особистим кабінетом адміністратора та додатковими функціональними можливостями системи.

Рис. 3.15 Сценарії використання особистого кабінету та додаткових функцій

Також адміністратор отримує доступ до особистого кабінету, в якому може переглядати та редагувати особисті дані ім’я, прізвище, пошту, змінити свій пароль та обов’язково має обрати найближче до нього відділення перед тим як створювати відправлення. Що дає можливість отримати кастомний профіль навіть адміністратору. Додатково отримує доступ до перегляду всіх існуючих відділення та трекінгу посилок.

3.4 Сценарії взаємодії кур’єра з системою

У цьому розділі розглядаються основні сценарії використання системи з точки зору кур’єра.

З метою підвищення читабельності загальну діаграму та їх опис було розділено на декілька логічно завершених піддіаграм, кожна з яких відображає окрему функціональну підсистему.

3.4.1 Сценарії обробки доставок та комунікації

На рисунку 3.16 наведено сценарії взаємодії кур’єра із системою під час виконання доставки

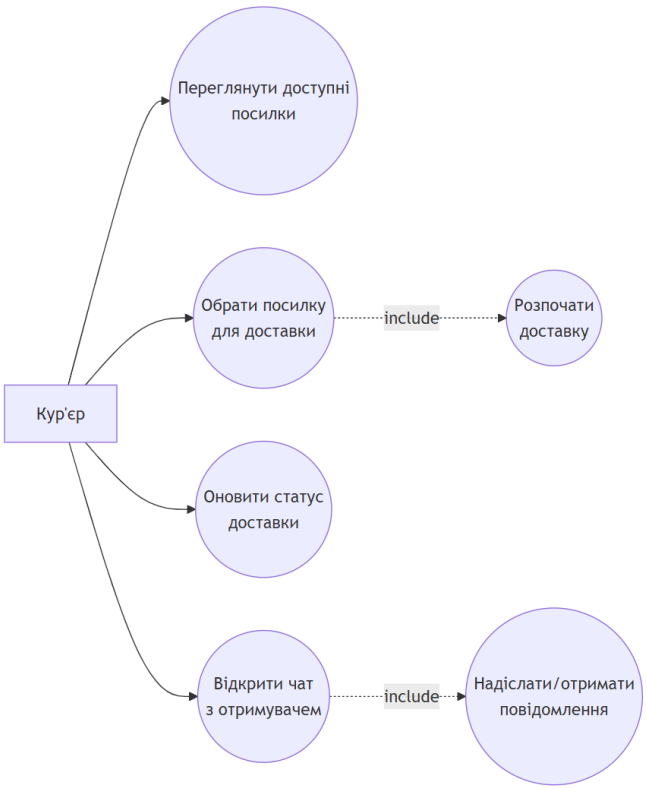


Рис. 3.16 Сценарії обробки доставок та комунікації

Основним сценарієм для використання системи кур’єром є використання сторінки для кур’єрів, на якій він може обрати посилки з типом доставки “кур’єром”, які ще не були забрані іншими кур’єрами та почати доставку до отримувача. Після чого він отримує доступ до переписки з отримувачем для узгодження нюансів та часу доставки.

3.4.2 Сценарії роботи кур’єра з особистим кабінетом

На рисунку 3.17 представлено сценарії роботи кур’єра з особистим кабінетом.

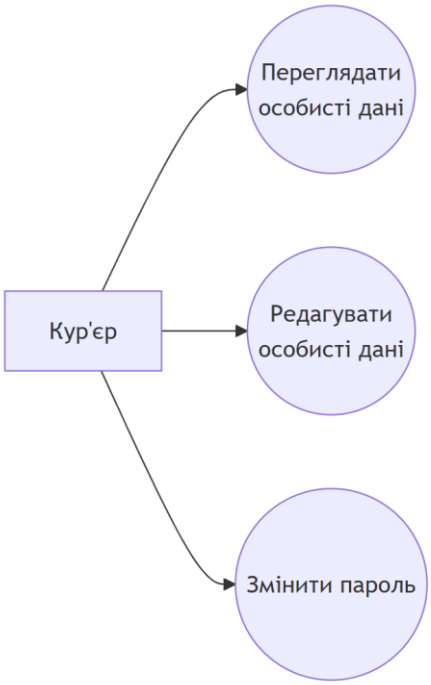


Рис. 3.17 Сценарії роботи кур’єра з особистим кабінетом

Також кур’єр отримує доступ до особистого кабінету, в якому може переглядати та редагувати особисті дані ім’я, прізвище, пошту, змінити свій пароль. Діаграми дозволяють краще зрозуміти обов’язки, можливості та залученість кур’єра в бізнес-процеси.

3.5 Сценарії взаємодії водія з системою

У цьому розділі розглядаються основні сценарії використання системи з точки зору водія.

З метою підвищення читабельності загальну діаграму та їх опис було розділено на декілька логічно завершених піддіаграм, кожна з яких відображає окрему функціональну підсистему.

3.5.1 Сценарій управління транспортом і доставкою

На рисунку 3.18 наведено сценарії взаємодії водія із системою під час організації перевезення посилок між відділеннями.

Рис. 3.18 Сценарії управління транспортом і доставкою

Основним сценарієм для використання системи водієм є використання сторінки для водіїв, на якій він може обрати транспорт доступний транспорт, яким він буде керувати. Після обрання транспорту йому відображається дані по посилкам, які були призначені в цей транспорт. Потім водій повинен надати доступ до своєї геолокації, щоб на інтерактивній карти при трекінгу посилки користувачі бачили її поточне місцезнаходження та приблизний час прибуття у відділення.

3.5.2 Сценарії роботи водія з особистим кабінетом

На рисунку 3.19 представлено сценарії, пов’язані з використанням особистого кабінету водія.

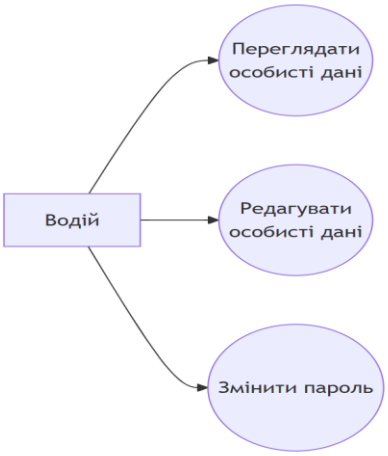


Рис. 3.19 Сценарії роботи водія з особистим кабінетом

Також водій отримує доступ до особистого кабінету, в якому може переглядати та редагувати особисті дані ім’я, прізвище, пошту, змінити свій пароль. Діаграми дозволяють краще зрозуміти обов’язки, можливості та залученість водія в бізнес-процеси.

3.6 Структурна діаграма класів системи управління посилками

Ця діаграма класів ілюструє базову логіку системи управління посилками. В основі структури лежить клас Package, який є центральним об’єктом системи та містить ключові дані про відправлення: номер відстеження, інформацію про відправника та одержувача, адресу доставки, вагу, тип доставки, вартість доставки, дату створення та поточний статус.

Для взаємодії з користувачами система використовує клас ApplicationUser, який представляє зареєстрованого користувача та керує авторизацією та розподілом ролей. Один і той самий користувач може бути одночасно відправником, одержувачем або працівником служби доставки. Ролі дозволяють контролювати доступ у системі; зокрема, існують ролі Адміністратор, Клієнт та Водій або Кур'єр.

Клас Package пов'язаний з класом Branch, який зберігає інформацію про відділення: назву, місто, адресу, години роботи, контактні дані та статус діяльності. Це дозволяє визначити місце початкової або кінцевої обробки відправлення, а також відобразити відділення на карті.

Для організації процесу доставки використовується клас Shipment, який містить інформацію про передачу посилки до доставки: відповідального співробітника, час, коли посилка була прийнята до перевезення, та поточний статус доставки. Таким чином, клас Shipment встановлює зв'язок між посилкою та співробітником, який здійснює перевезення.

Клас StatusHistory служить для зберігання історії змін, що дозволяє фіксувати всі етапи проходження відправлення. Кожен запис в історії містить попередній або

новий статус, дату зміни та, за необхідності, коментар. Це гарантує прозорість відстеження та можливість перегляду всіх змін статусу доставки.

Для полегшення комунікації між учасниками системи можна використовувати клас Chat та пов'язаний з ним клас Message, що спрощує обмін повідомленнями між клієнтом та кур'єром. Це розширює функціонал системи та дозволяє користувачам швидко уточнювати деталі доставки.

Статуси посилок реалізовано у вигляді переліку Status, який містить основні стани життєвого циклу відправлення, такі як: створено, у дорозі, доставлено, отримано, скасовано. Також рекомендується представити тип доставки у вигляді окремого переліку DeliveryType, що містить варіанти, пов’язані зі способом відправлення.

Представлена структура гарантує модульність, масштабованість та підтримку основних бізнес-процесів системи управління посилками. Завдяки чіткому розділенню сутностей та відносин між ними систему можна легко розширити додатковими функціональними можливостями, такими як геолокація транспортних засобів, формування накладних, статистичний аналіз або сповіщення в режимі реального часу.

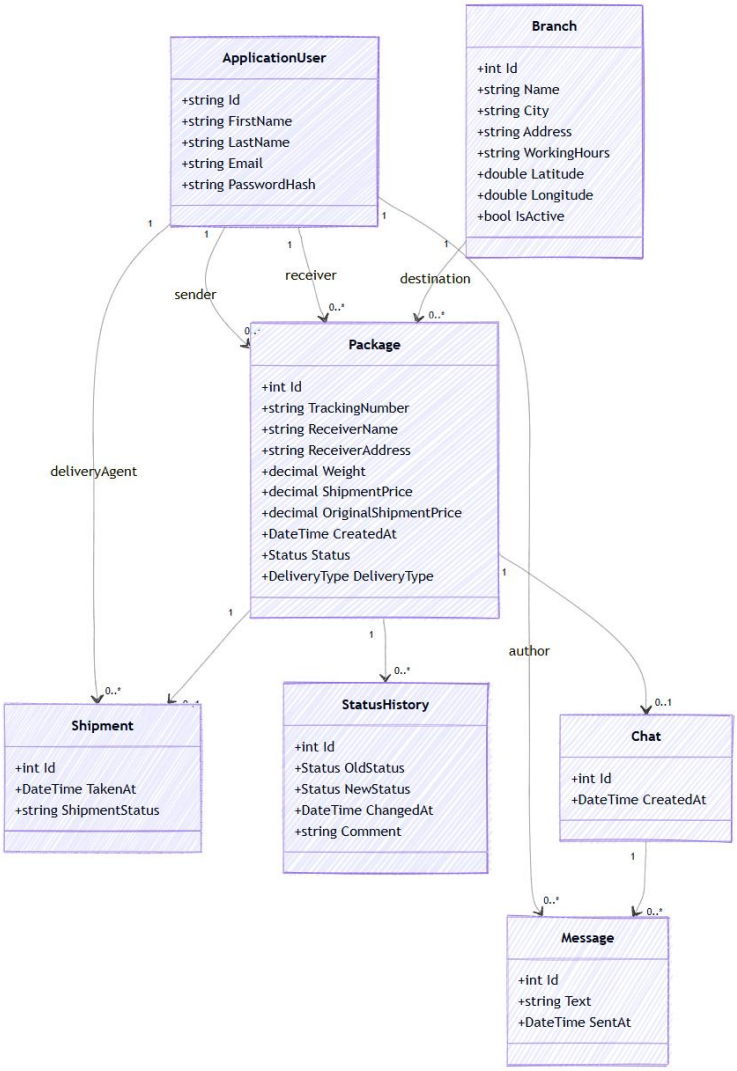


Рис. 3.20 Діаграма класів системи управління посилками

Діаграма класів системи управління посилками наведена на рисунку 3.20.

3.7 Алгоритм відстеження місцезнаходження посилки

У системі управління доставками посилок одним з основним елементів є визначення поточного місцезнаходження посилки. Даний алгоритм реалізовано поєднуючи статус моделі посилки та геолокаційних координат.

На етапі доставки між відділеннями використовується механіхм при якому координати посилки визначаються спираючись на місцезнаходження водія. Координати отримуються з браузера водія через Geolocation API, після чого на сервер йде HTTP-запит update-location та сервер обробляє його та зберігає дані для

подальшого використання. Нижче на рисунку 3.21 відображено алгоритм збереження координат водія, для відображення позиції на карті.[17]

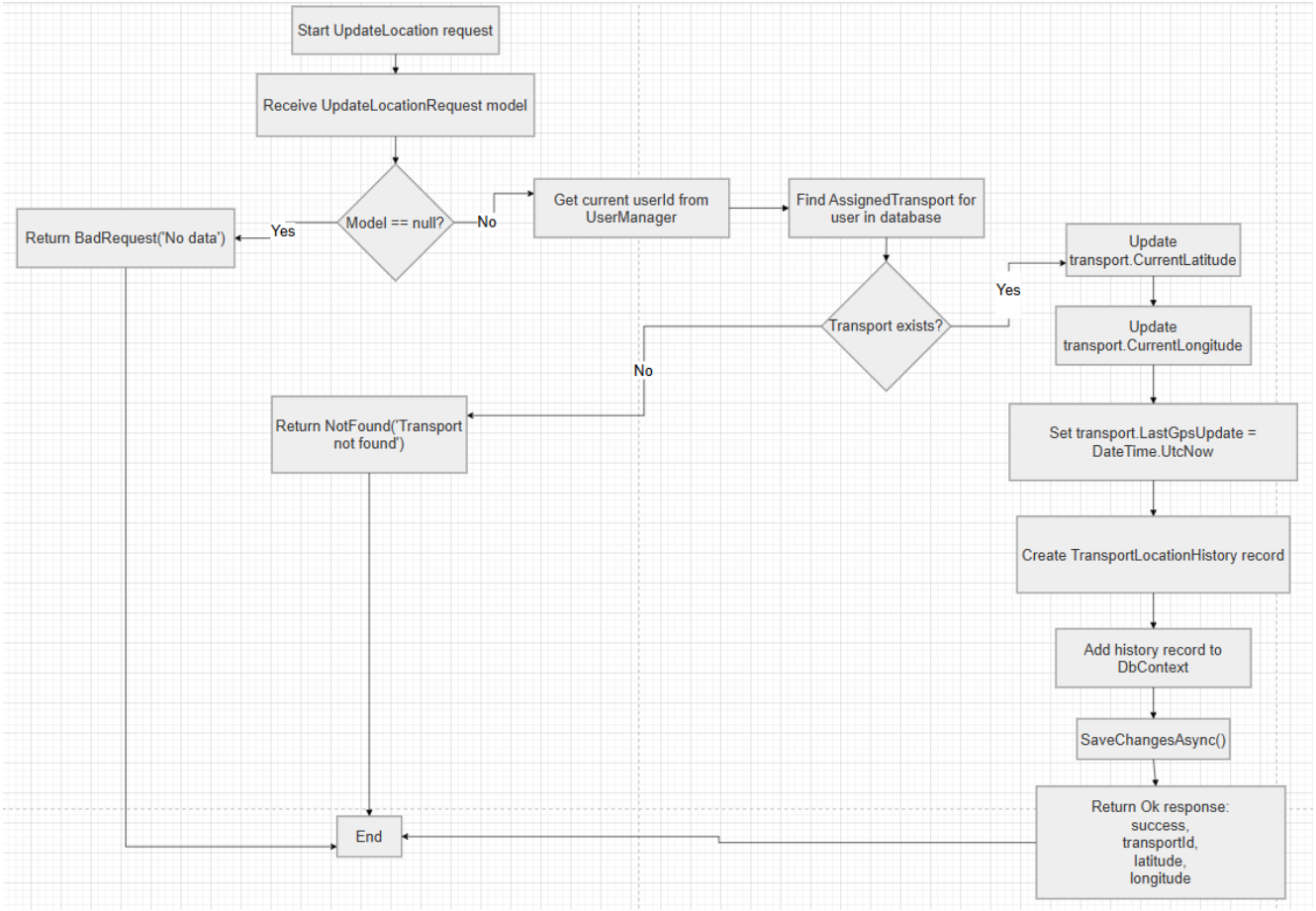


Рис. 3.21 Алгоритм збереження координат водія, для відображення позиції на карті

Однією з особливостей реалізації є періодичне оновлення координат за допомогою використання інтервалів, що допомагає майже в реальному часі отримувати місцезнаходження посилки. На рисунку 3.22 наведено алгоритм повторного запиту координат водія.

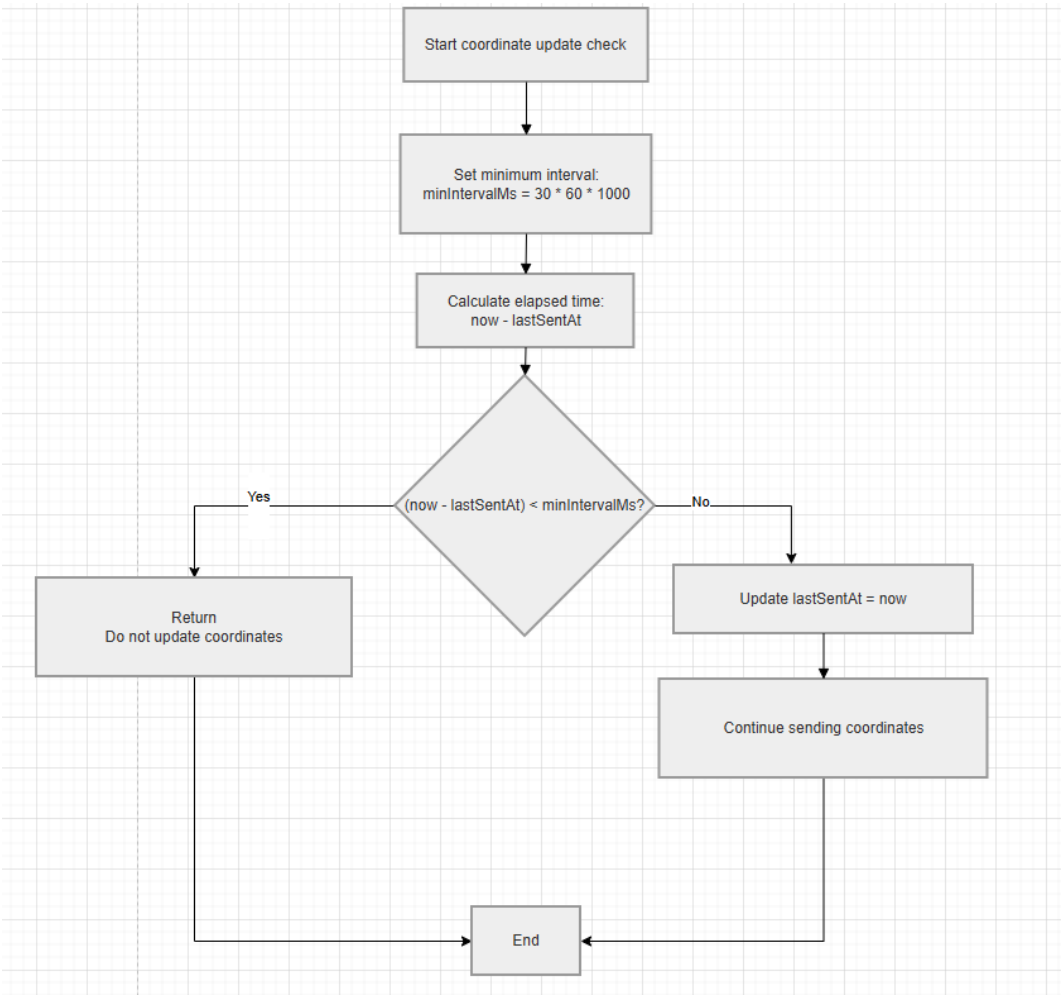


Рис. 3.22 Алгоритм повторного запиту координат водія

3.8 Алгоритм обміну повідомленнями в реальному часі

У системі управління доставками посилок ще одним з основним елементів є обмін повідомленнями кур'єра з отримувачем. У додатку даний функціонал реалізовано за допомогою бібліотеки SignalR.

SignalR – бібліотека яка допомагає встановити постійне з'єднання між сервером і клієнтом. Це значить, сервер може надсилати повідомлення клієнтам у будь-який час, а клієнт може відповідати на ці повідомлення, що забезпечує спілкування та оновлення в онлайн режимі без оновлення сторінки або постійного опитування[11].

Реалізація чату включає в себе реалізацію контроллера та хабу. Контроллер містить в собі метод Index() та GetHistory(). Index() перевіряє чи доступний

користувачу чат взагалі і якщо так то користувач отримує до нього доступ. На рисунку 3.23 відображено алгоритм методу Index().

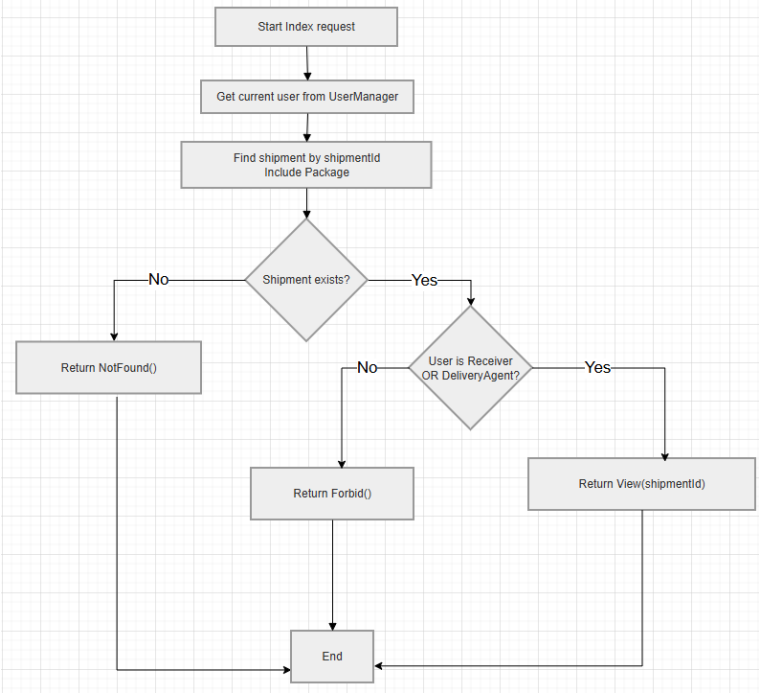


Рис. 3.23 Алгоритм методу Index

Метод GetHistory() в свою чергу відображає уже відправлені повідомлення в чаті. На рисунку 3.24 відображено алгоритм методу GetHistory().

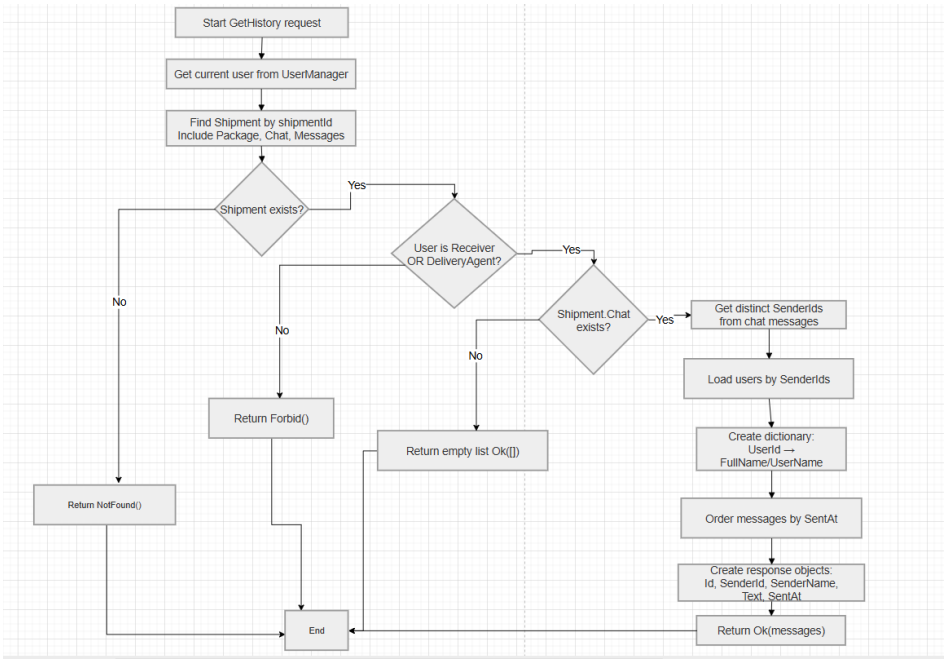


Рис. 3.24 Алгоритм методу GetHistory()

Основна логіка алгоритму прописана в класі ChatHub, який приймає в себе текст повідомлення та айді доставки, спочатку відбувається перевірка чи чат взагалі доступний користувачу, якщо так то в методі SendMessage() обробляється текст повідомлення при успішній відправці іншому користувачу відправляється Websocket з повідомленням та часом відправки. На рисунку 3.25 відображено алгоритм роботи методу SendMessage().

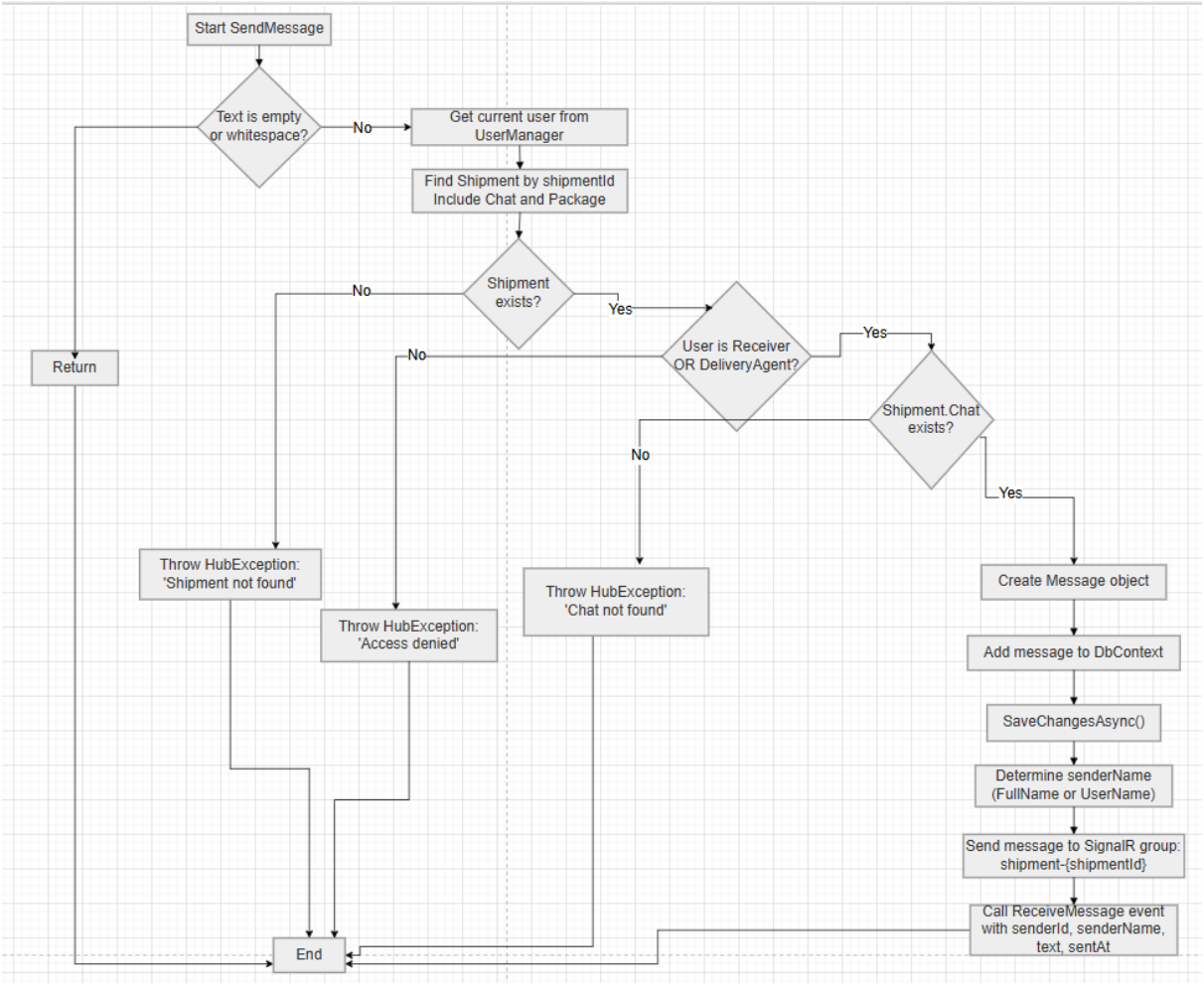


Рис. 3.25 Алгоритм методу SendMessage()

3.9 Алгоритми в контроллері DeliveryController

Функціонал для кур’єрів є одним з найважливіших у даній системі тому окрім основних статусів посилок було додано модель ShipmentStatus, що містить в собі перелік статусів доставки.

Основні статуси доставки в додатку:

– Taken - кур’єр призначив посилку на себе;

– InTransit - кур’єр на шляху до отримувача;

– Delivered - кур’єр віддав посилку отримувачу;

– Cancelled - отримувач скасував доставку кур’єром.

Для реалізації цього було створено контроллер DeliveryController в якому створено метод AvailablePackages(), що показує кур’єру всі посилки в статусі Delivered і ніхто з кур’єрів ще її не забрав. На рисунку 3.26 відображено алгоритм роботи методу AvailablePackages().

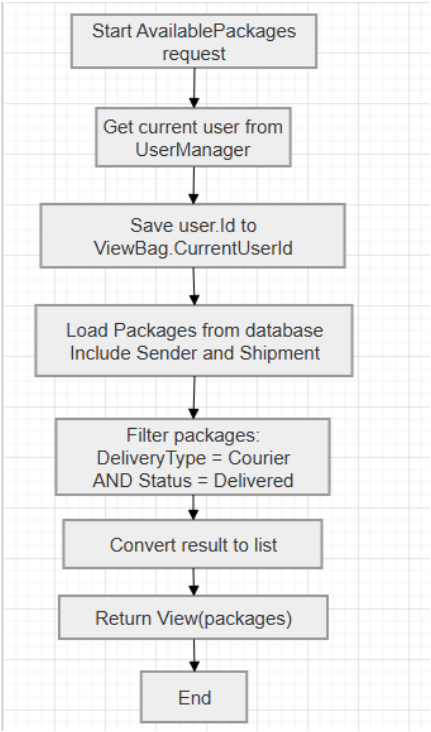


Рис. 3.26 Алгоритм методу AvailablePackages()

Також реалізовано метод TakePackage(), який допомагає кур’єру після перегляду доступних посилок до доставки призначити їх собі, якщо ще ніхто не забрав після чого статус доставки змінюється на Taken. На рисунку 3.27 відображено алгоритм роботи методу TakePackage().

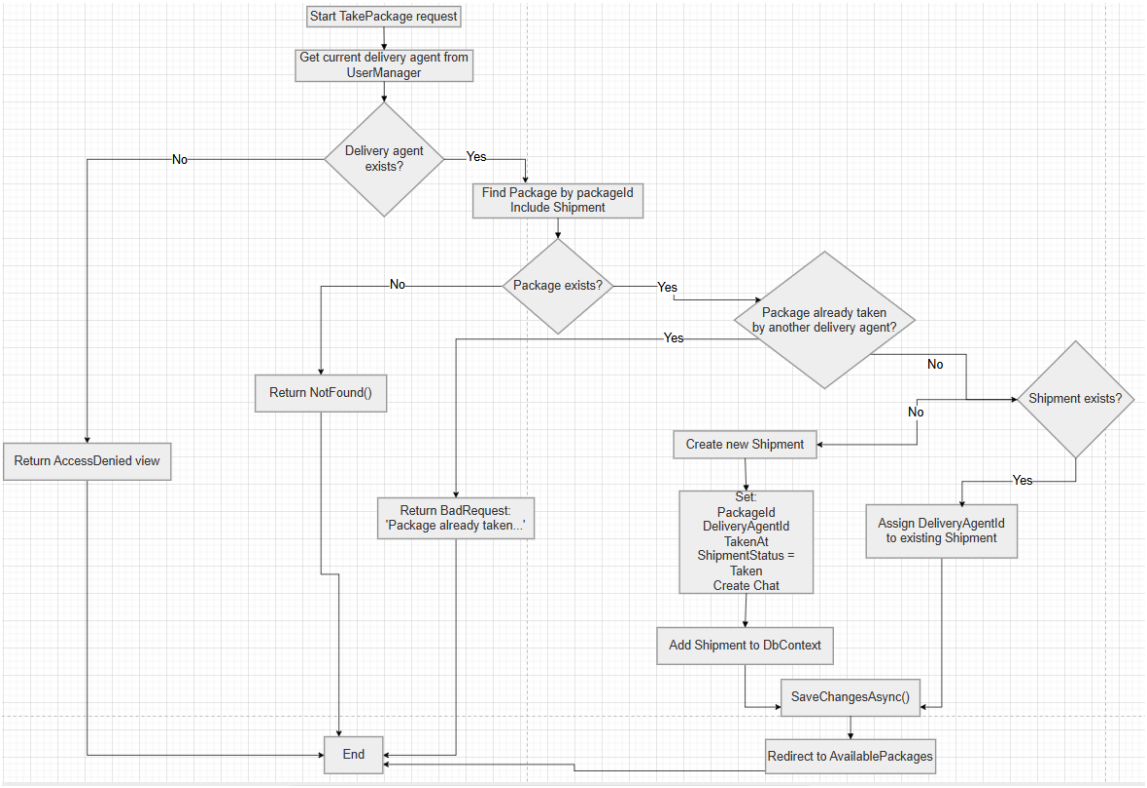


Рис. 3.27 Алгоритм методу TakePackage()

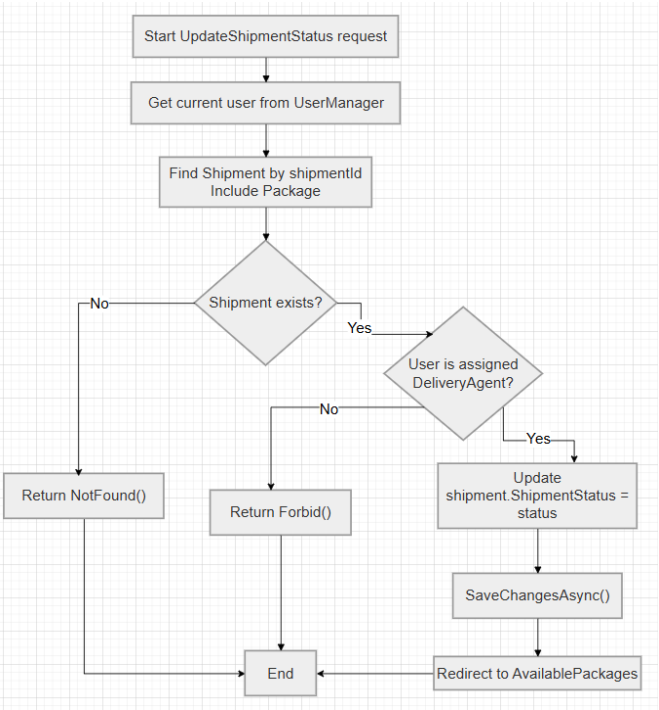


Рис. 3.28 Алгоритм методу UpdateShipmentStatus()

Та фінальний метод контроллера це – UpdateShipmentStatus(), що допомагає кур’єру оновлювати статус доставки посилки. На рисунку 3.28 відображено алгоритм роботи методу UpdateShipmentStatus().

3.9 Алгоритми які використовуються в особистому кабінеті користувача PackageController

У сфері логістики та поштових відправлень дуже важлива наявність особистого кабінету, тому було прийняти рішення розробити його.

Основною сторінкою в кабінеті є сторінка що містить всі твої відправлення, для цього було реалізовано метод Index(), було додано можливість сортувати посилки за трекінговим номером, відправником та статусом посилки. На рисунку 3.29 відображено алгоритм роботи методу Index().

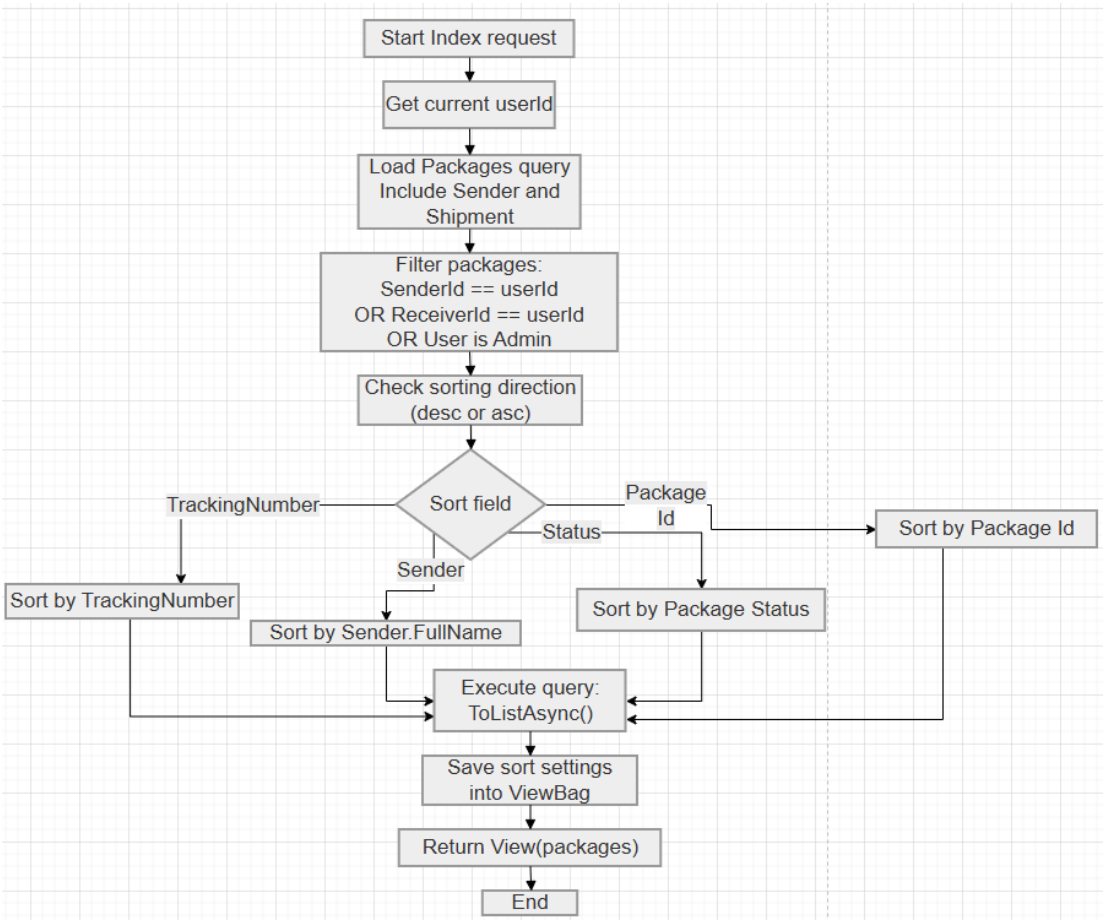


Рис. 3.29 Алгоритм методу Index()

Аби створити відправлення користувачу спочатку потрібно вказати найближче до себе відділення. Після чого необхідно вказати Weight, Sender, ReciverEmail, обрати тип доставки Courier або Pickup, обрати відділення отримувача та якщо тип доставки Courier вказати адресу отримувача. Ціна доставки визначається автоматично та реалізована в сервісі DeliveryPricingService(). Ціна доставки відправлення залежить від двох факторів: ваги(Weight) та типу доставки(Courier або Pickup). Алгоритм визначення ціни доставки відображений на рисунку 3.30.

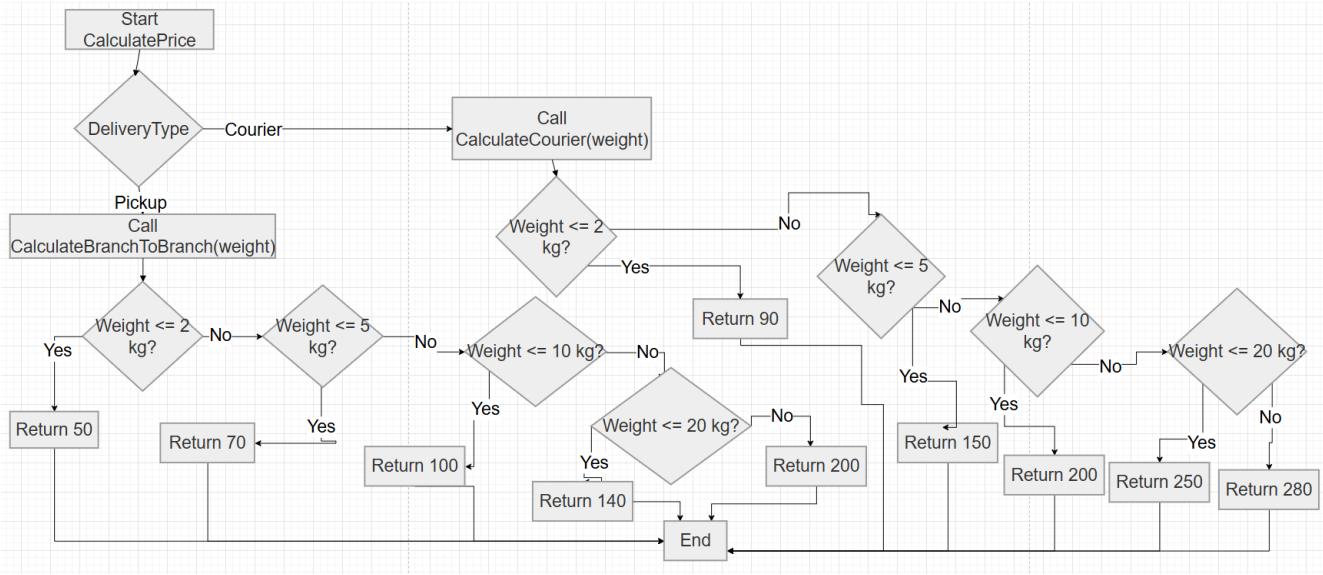


Рис. 3.30 Алгоритм методу DeliveryPricingService()

Однією з найважливіших складових систем у сфері логістики є генерація накладної у форматі pdf, якщо користувач потребує її. Генерацію накладної було реалізовано окремим сервісом InvoiceDocument() аби не засмічувати контроллери бізнес логікою. Він містить два методи. Перший Compose() генерує структуру документа, в якому міститься інформація про трекінговий номер посилки(TrackingNumber), відправника(Sender), отримувача(Receiver), адресу отримувача(ReceiverAddress), вагу(Weight) та дату створення посилки(CreatedAt), реалізація відображена на рисунку 3.31.

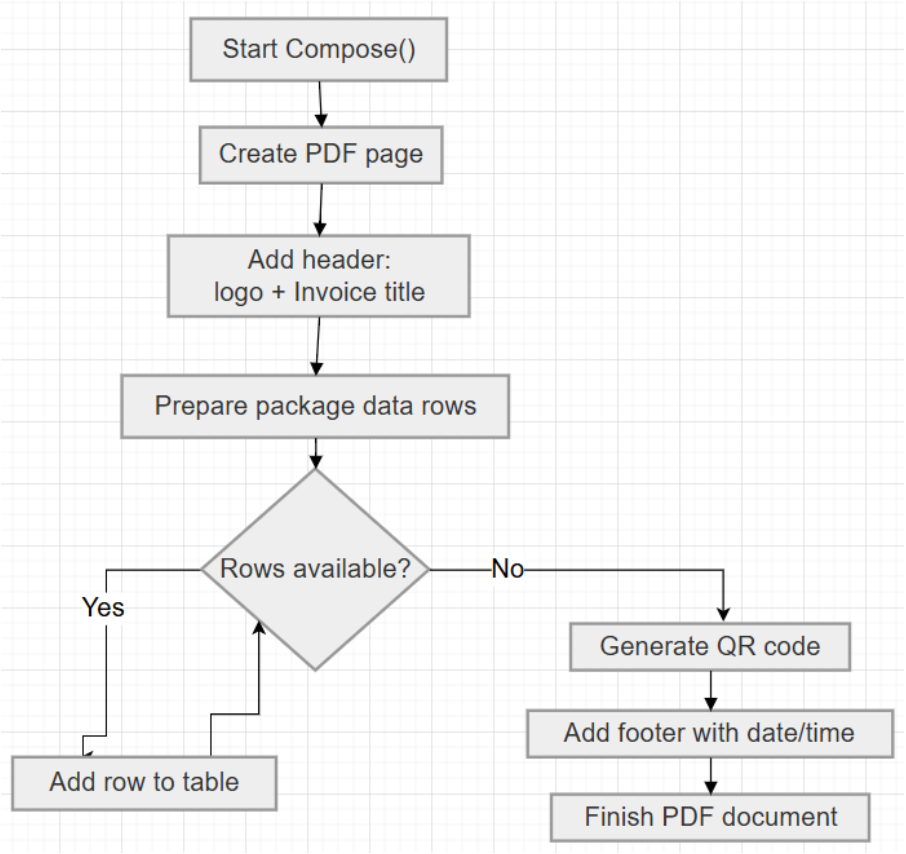


Рис. 3.31 Алгоритм методу Compose()

Метод GenerateQrCode() відповідає за формування QR-коду для накладної посилки. Створюється об’єкт QRCodeGenerator, після чого на основі трекінгового номера посилки _package.TrackingNumber генеруються дані для QR-коду. Потім ці дані перетворюються у PNG-зображення за допомогою класу PngByteQRCode.

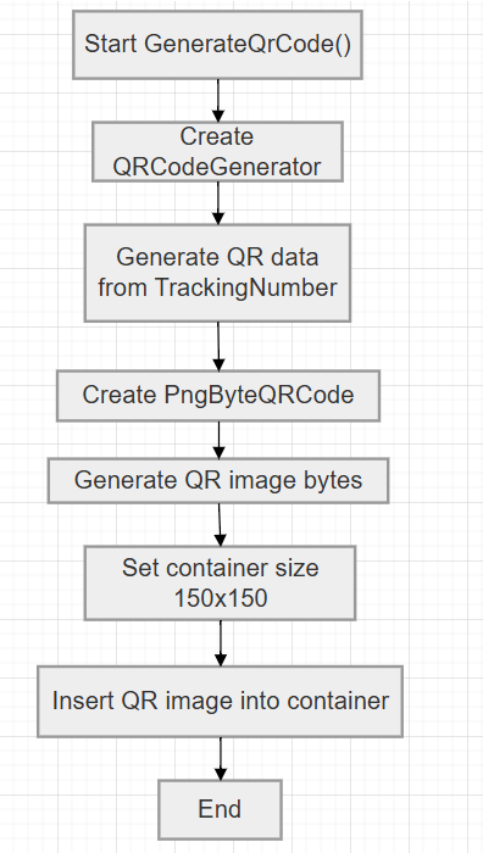


Рис. 3.32 Алгоритм методу GenerateQrCode()

Для QR-коду задається фіксований розмір 150x150px, а параметр ImageScaling.FitArea забезпечує коректне масштабування зображення всередині контейнера. Алгоритм роботи методу GenerateQrCode() відображено на рисунку 3.32.

3.10 Структура бази даних

Для проектування веб- додатку було обрано реляційну модель бази даних. База даних складається з моделей, які є найважливішими для мінімально життєздатного продукту в сфері логістики: посилки, відділення, доставки, транспорт, користувачі, чати, місії для бонусної політики, ролі та логування статусів посилок.

Однією з основних таблиць є AspNetUsers, яка зберігає в собі персональні дані користувача, включно з відділенням користувача, транспорт, яким

відбувається перевезення між відділеннями, логін та пароль, повне ім’я. Таблиця AspNetUserRoles містить в собі містить в собі ролі користувачів.

Вся інформація про посилки зберігається в таблиці Packages, що містить в собі інформацію про відправника, отримувача, відділення, трекінговий номер, статус, вагу посилки, тип, ціну доставки, транспорт, який перевозить відправлення. А для логування зміни статусів посилки додано таблицю StatusHistory.

Інформація про доставку кур’єром міститься в таблиці Shipment, яка містить в собі дані про кур’єра, статус доставки та чат для комунікації з отримувачем.

Для реалізації функціоналу бонусної політики у вигляді місій було додано наступні таблиці: Mission, UserReward, UserMissionProgress.

У свою чергу таблиця Mission містить в собі загальні дані про назву, тип місії, тип нагороди та активність функціоналу.

UserReward містить інформація про користувача, тип отриманої нагороди, час її отримання та використання. UserMissionProgress зберігає в собі прогрес користувача в даній активності.

Таблиця Transport містить всю інформацію про транспорт: номер, тип транспорту, координати та посилки, що перевозяться в поточній поїздці.

Таблиця Branch зберігає всі дані про відділення: адресу, робочі години та місцезнаходження.

Для комунікації за допомогою чату створено дві таблиці Chat та Message.

Перша зберігає дані для прив’язки чату до доставки та видачі всіх повідомлень по конкретній доставці. Друга містить дані про конкретні повідомлення: текст, час відправки, отримання повідомлення та ChatId.

Структура бази даних спроектована та адаптована для роботи з ORM(Entity Framework Core), простішого масштабування додатку, та ефективного управління даними.

Структура бази даних проілюстрована в додатку В на рисунку В.1.

3.11 UI системи

За кожною своєчасною доставкою стоїть не лише потужність двигуна та вдалий маршрут. Існує прихована сила, яка тихо забезпечує ефективність операцій: розумний дизайн UX/UI. Від інформаційних панелей, що скорочують час прийняття рішень удвічі, до мобільних додатків, які працюють на складі о 2-й ночі, – дизайнерські рішення безпосередньо визначають, наскільки швидким, точним і масштабованим може бути логістичний бізнес[18].

Користувацький інтерфейс для цифрової системи доставки та контролю посилками було розроблено адаптивним для різних розширень екрану. Критеріями прийнятності для інтерфейсу стали: логічність інтерфейсу, зручність, та простота. Для верстки HTML-сторінок використано шаблонізатор Razor, а для стилів сторінки – CSS та Bootstrap, для створення додатків із використанням адаптивного дизайну[16].

На рисунках(3.32-3.39) поданих нижче відображено основні елементи інтерфейсу додатку.

На рисунку 3.32 відображено дашборд, який надає доступ до основних функцій додатку. Інтерфейс надає доступ до створення посилок, перегляду інформації про посилку, редагування, видалення, генерації накладної, масової зміни статусів посилок, чату з кур'єром, перегляду карти всіх відділень, перегляду та редагування даних профілю, перегляду умов отримання бонусів, а також перегляду інформаційних сторінок що містять в собі інформацію про додаток та цінову політику.

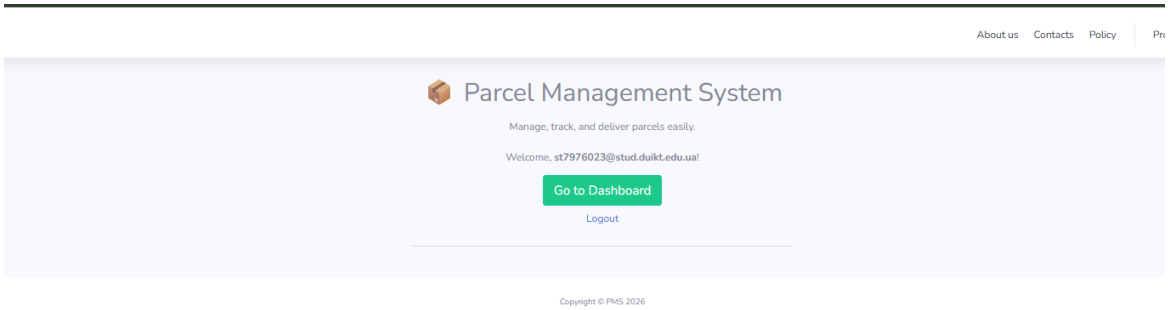


Рис. 3.32 Головна сторінка застосунку

На рисунку 3.33 відображено сторінку «Register», основною функцією якої є реєстрація користувача. Для реєстрації необхідно вказати повне ім’я користувача, його пошту та пароль. Пароль повинен містити велику літеру, малу літеру, цифру та спецсимвол. Пароль повинен складатися щонайменше з шести символів[13].

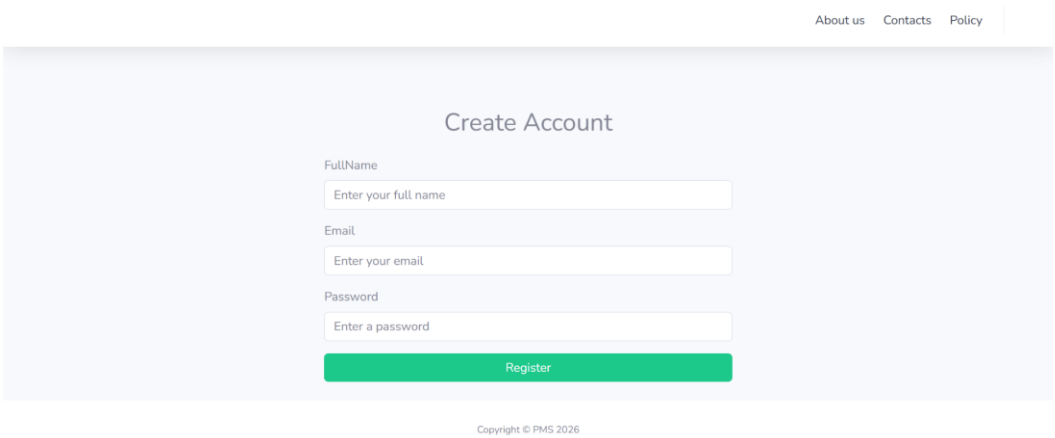


Рис. 3.33 Сторінка «Register»

На рисунку 3.34 відображено сторінку «Login», що призначена для авторизації користувачів. В формі є перевірка на валідність пошти та відповідність паролю введеного при реєстрації.

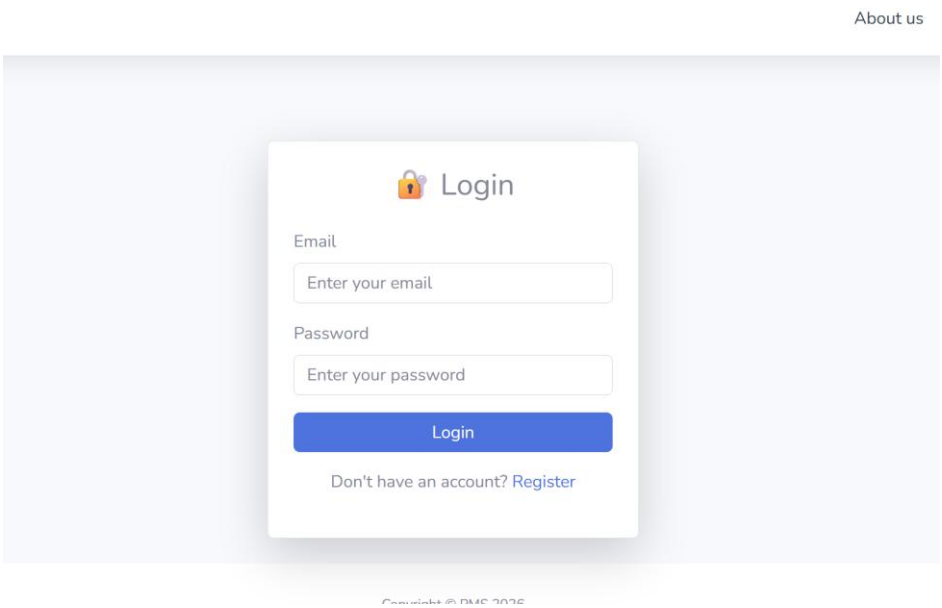


Рис. 3.34 Сторінка «Login»

На рисунку 3.35 відображено сторінку з усіма створеними посилками, що має базове сортування по трекінговому номеру, відправнику, статусу. Також можна

згенерувати накладну лише в один клік та якщо кур’єр взяв в роботу замовлення доступ до чату.

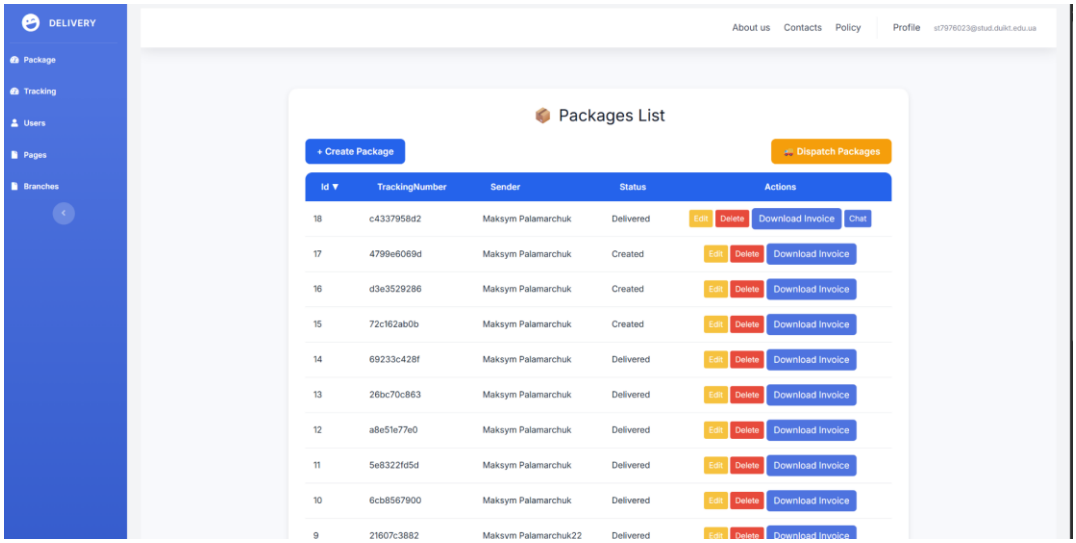


Рис. 3.35 Сторінка відправлень

На рисунку 3.36 відображено профіль користувача, який містить в собі інформацію про користувача, а саме: повне ім’я користувача, пошта та найближче відділення до нього. Крім, цього користувач може змінити свій поточний пароль на новий.

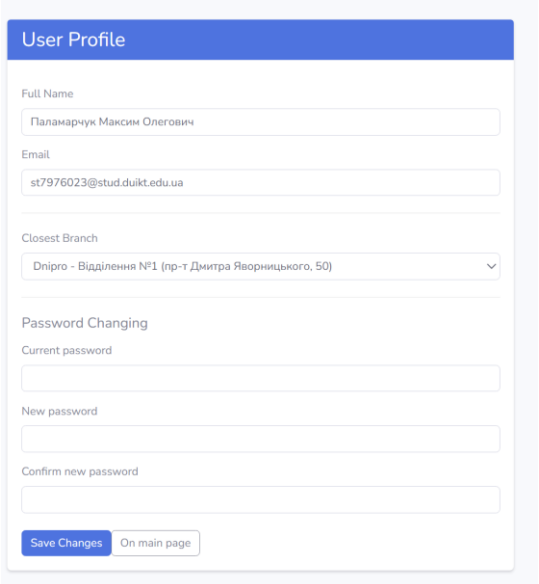


Рис. 3.36 Сторінка профілю

На рисунку 3.37 відображено сторінку бонусів, що містить в собі доступні місії, їх опис та прогрес користувача, після їх виконання користувач отримує

бонуси які може застосувати при створенні наступних відправлень, доступні бонуси відображаються в окремому блоці

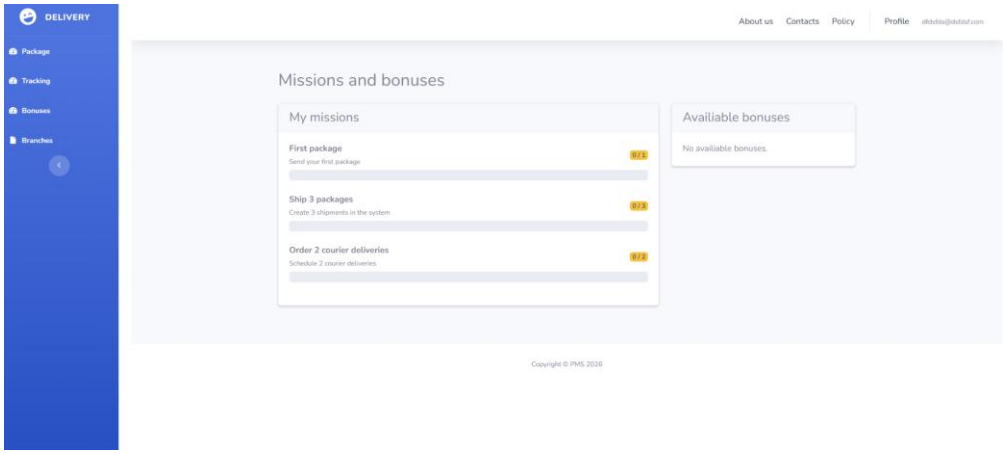


Рис. 3.37 Сторінка бонусів

На рисунку 3.38 відображено сторінку відділень що містить в собі мапу з усіма наявними відділеннями. На цій сторінці є можливість відфільтрувати відділення по місту та побачити їх на карті, клікнувши на відділення, отримуємо дані про нього: адресу розташування робочі години та його номер.

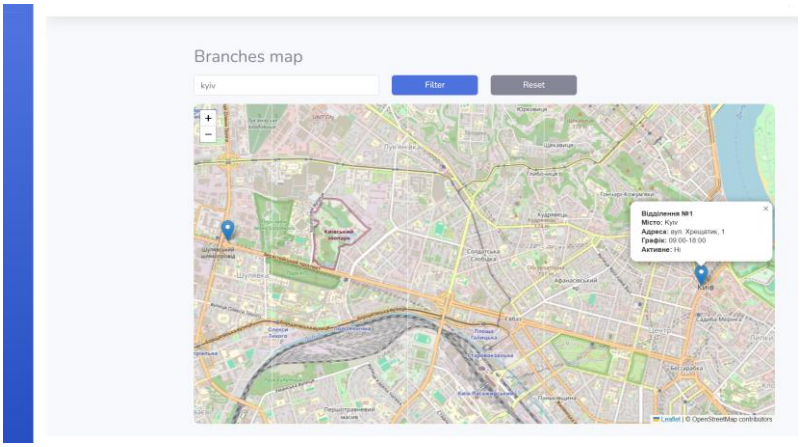


Рис. 3.38 Сторінка відділень

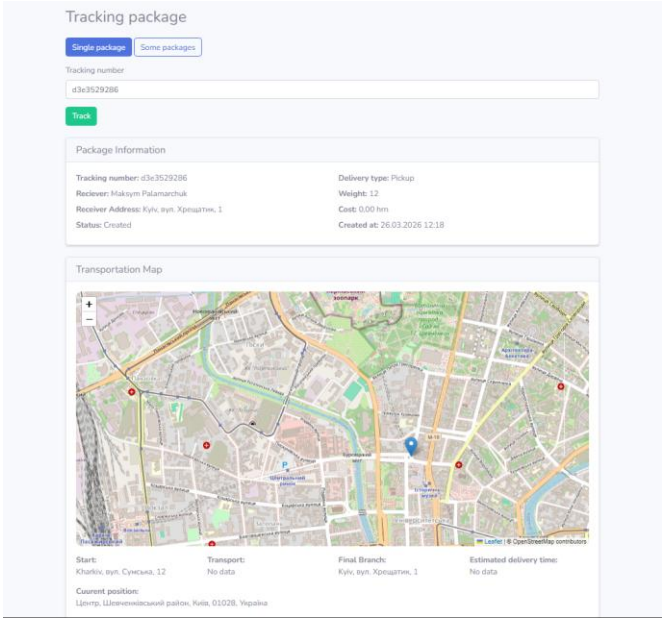


Рис. 3.39 Сторінка трекінгу

На рисунку 3.39 відображено сторінку трекінгу посилок. Містить в собі поле для вводу номеру відстеження посилки і потім вся отримана інформація про неї виводиться. Якщо у відправлення статус Created, то на карті поточне місцезнаходження дорівнює координатам відділення відправника. Після переходу в статус InTransit на карті будується маршрут між відділенням відправника та отримувача та відображається поточне місцезнаходження транспорту. Також після переходу в статус InTransit користувачу відображається приблизний час доставки до відділення отримувача.

4 ТЕСТУВАННЯ ДОДАТКУ

4.1 Тестування бізнес логіки, використовуючи xUnit

Для того щоб основні контроллери надійно працювали, було проведено модульне тестування за допомогою фреймворку xUnit та Moq. Основною метою тестування стало підтвердження роботи ключових контролерів та сервісів, а саме: реєстрація, авторизація, відновлення паролю; створення, редагування, перегляд інформації, генерація накладної, видалення та масова зміна статусів посилок; відображення доступних посилок для кур’єрів, взяття посилки кур’єром, зміна статусу доставки кур’єром[19][20].

4.1.1 Тестування AccountController

У контролері перевірено метод реєстрації, авторизації, відновлення паролю, підтвердження пошти, виходу з акаунту, перевірку на існування пошти в базі даних. Сценарії, які були використані для тестування відображені таблиці 4.3.

Таблиця 4.3

Чеклист для AccountController

Назва перевірки	Вхідні дані
Реєстрація з валідними даними	Коректні email, пароль
Реєстрація з невалідними даними	Некоректні дані
Авторизація з валідними даними	Вірний email і пароль
Авторизація з невалідними даними	Невірні дані
Відновлення паролю (невалідний email)	Невалідний email
Відновлення паролю (валідний email)	Існуючий email

Продовження таблиці 4.3

Назва перевірки	Вхідні дані
Відновлення паролю (неіснуючий email)	Email відсутній
Скидання паролю без токена	Відсутній токен
Успішне скидання паролю	Валідний токен
Підтвердження пошти без токена	Немає токена/id
Підтвердження пошти (неіснуючий user)	Невірний userId
Успішне підтвердження пошти	Валідні дані
Вихід з акаунту	Авторизований користувач

Тести перевіряють реакцію інтерфейсу на відповідні набори даних та валідацію даних. Результати тестування контролера відображені на рисунку 4.40.

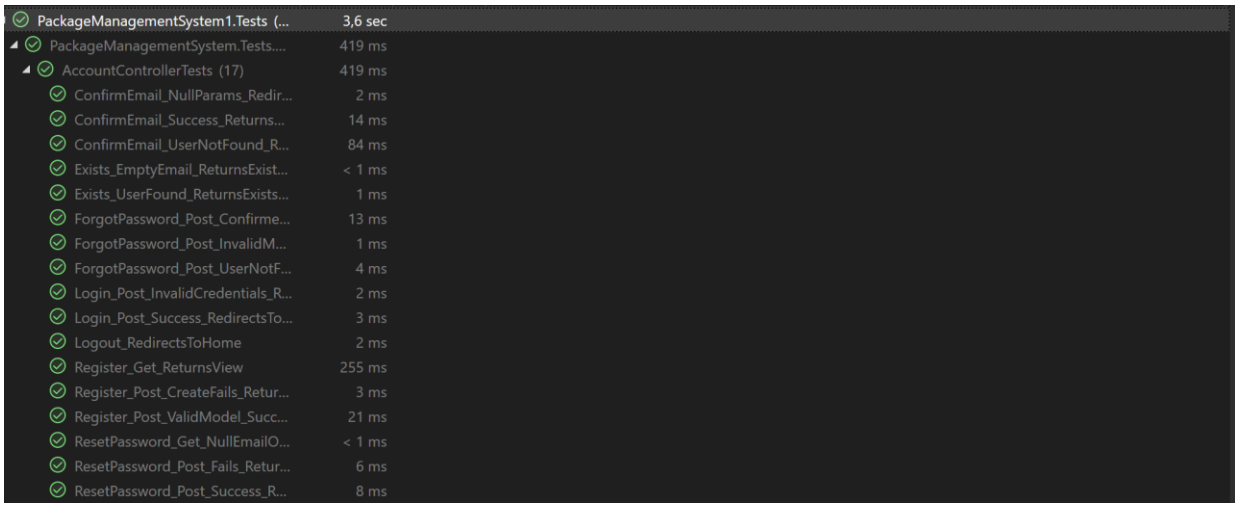


Рис. 4.40 Результати тестування AccountControllerTests

Під час тестування було перевірено функціонування аутентифікації та управління обліковими записами користувачів, зокрема обробку реєстраційних запитів із валідними та невалідними даними, перевірку авторизації користувачів, правильність процесів відновлення та скидання пароля, а також підтвердження електронної адреси та процес виходу з системи.

4.1.2 Тестування PackageController

У контролері перевірено метод виводу, створення, редагування, перегляду інформації, видалення посилки, генерацію накладної, масової зміни статусів. Сценарії, які були використані для тестування відображені в таблиці 4.4.

Таблиця 4.4

Чеклист для PackageController

Назва тесту	Вхідні дані
Відображення лише своїх посилок	Авторизований користувач
Створення посилки з неіснуючим відправником	Некоректний UserId
Успішне створення посилки	Валідні дані
Перегляд даних про посилку	Існуючий Id
Видалення посилки	Існуючий Id
Генерація накладної	Валідний Id
Масова зміна статусів без вибраних посилок	Порожній список
Створення посилки з неіснуючим відправником	Некоректний UserId
Успішне створення посилки	Валідні дані
Перегляд даних про посилку	Існуючий Id
Видалення посилки	Існуючий Id
Генерація накладної	Валідний Id
Масова зміна статусів без вибраних посилок	Порожній список
Успішна масова зміна статусів	Валідні дані

Продовження таблиці 4.4

Назва тесту	Вхідні дані
Створення посилки з невалідними даними	Порожні або некоректні поля
Створення посилки з неіснуючим відділенням	Некоректний BranchId

Тести перевіряють реакцію інтерфейсу на відповідні набори даних та валідацію даних. Результати тестування контролера відображені на рисунку 4.41.

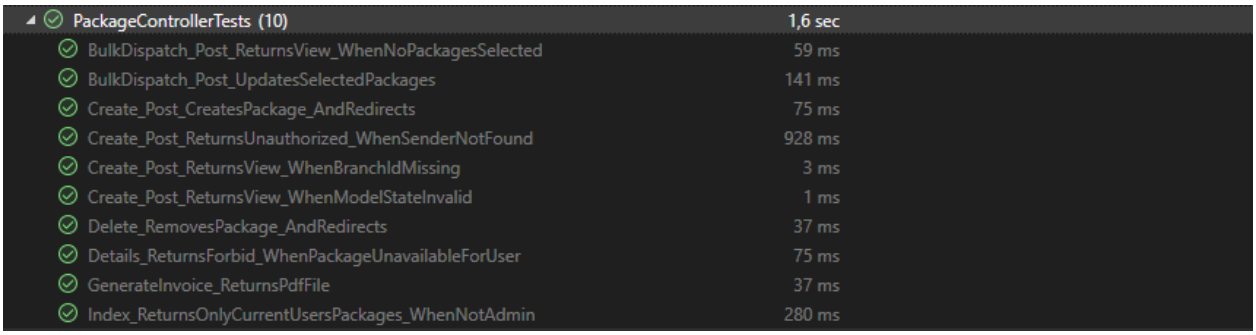


Рис. 4.41 Результати тестування PackageControllerTests

Тестування підтвердило правильність реалізації бізнес-логіки управління відправленнями, зокрема обмеження доступу до відправлень користувача, перевірку вхідних даних під час створення відправлення, цілісність взаємозв’язків між об’єктами системи, а також правильність виконання таких операцій, як перегляд, видалення, формування накладних та масове оновлення статусів.

4.1.3 Тестування DeliveryController

У контролері перевірено метод виводу доступних до доставки кур’єром посилок, взяття відправлення в роботу, зміни статусу доставки. Сценарії, які були використані для тестування відображені в таблиці 4.5.

Таблиця 4.5

Чеклист для DeliveryController

Назва тесту	Вхідні дані
Відображення доступних посилок	Авторизований кур’єр
Взяття посилки неіснуючим користувачем	Некоректний UserId
Взяття неіснуючої посилки	Некоректний Id
Взяття зайнятої посилки	Посилка вже призначена
Успішне взяття посилки в роботу	Валідні дані
Оновлення неіснуючої доставки	Некоректний Id
Оновлення доставки іншим кур’єром	Інший користувач
Успішне оновлення статусу	Валідні дані

Тести перевіряють реакцію інтерфейсу на відповідні набори даних та валідацію даних. Результати тестування контролера відображені на рисунку 4.42.

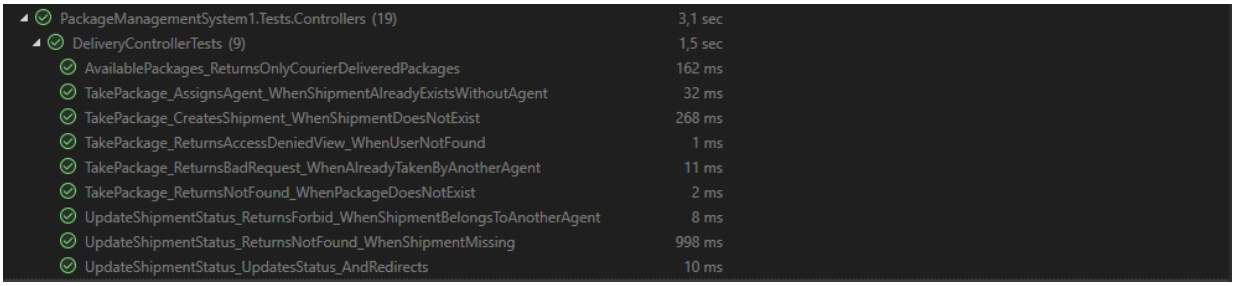


Рис. 4.42 Результати тестування DeliveryControllerTests

Тестування дозволило переконатися, що сценарії взаємодії кур'єра з системою обробляються правильно, а також переконатися у належному функціонуванні механізму доставки.

4.2 Кросбраузерне тестування

Веб-додаток часто є першим пунктом контакту з користувачами, а перше враження має велике значення. Однак забезпечити безперебійну роботу у всіх браузерах та на всіх пристроях стає дедалі складніше. Сучасний веб є фрагментованим. Користувачі отримують доступ до цифрових продуктів на настільних комп'ютерах, планшетах і смартфонах за допомогою різних браузерів, зокрема Chrome, Safari, Firefox, Edge та інших, кожен із яких має свої особливості та механізми візуалізації[21].

Для перевірки стабільності інтерфейсу, коректності верстки та взаємодії з бізнес логікою сервісу для контролю та керування доставками посилок проведено кросбраузерне тестування. Процес відбувався вручну за допомогою Edge, Chrome DevTools та інших інструментів вбудованих в браузери[22].

Тестування проводилось в таких браузерах: Google Chrome(рис. 4.43), Microsoft Edge(рис. 4.44).

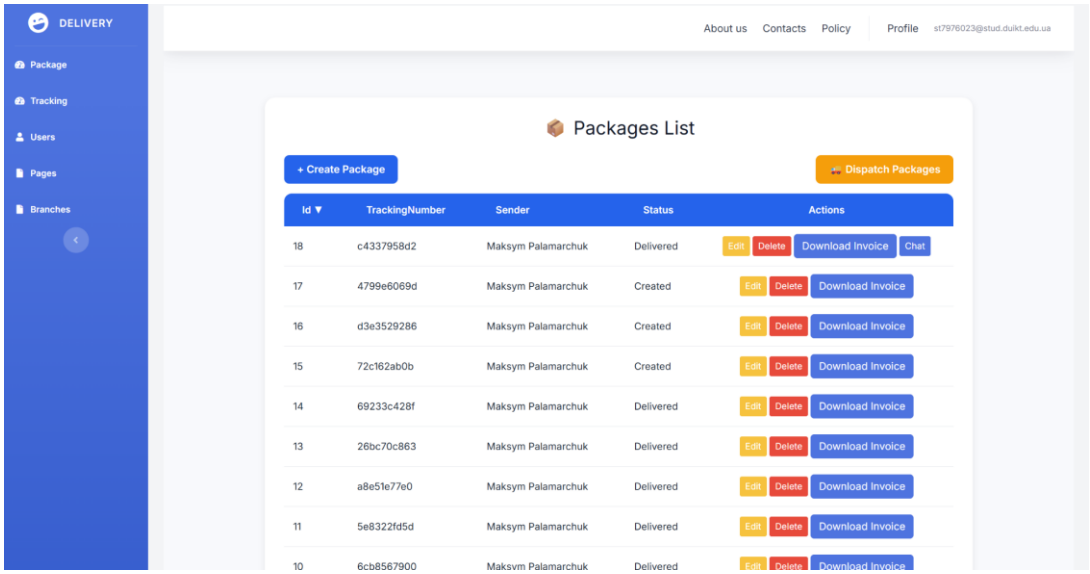


Рис. 4.43 Google Chrome

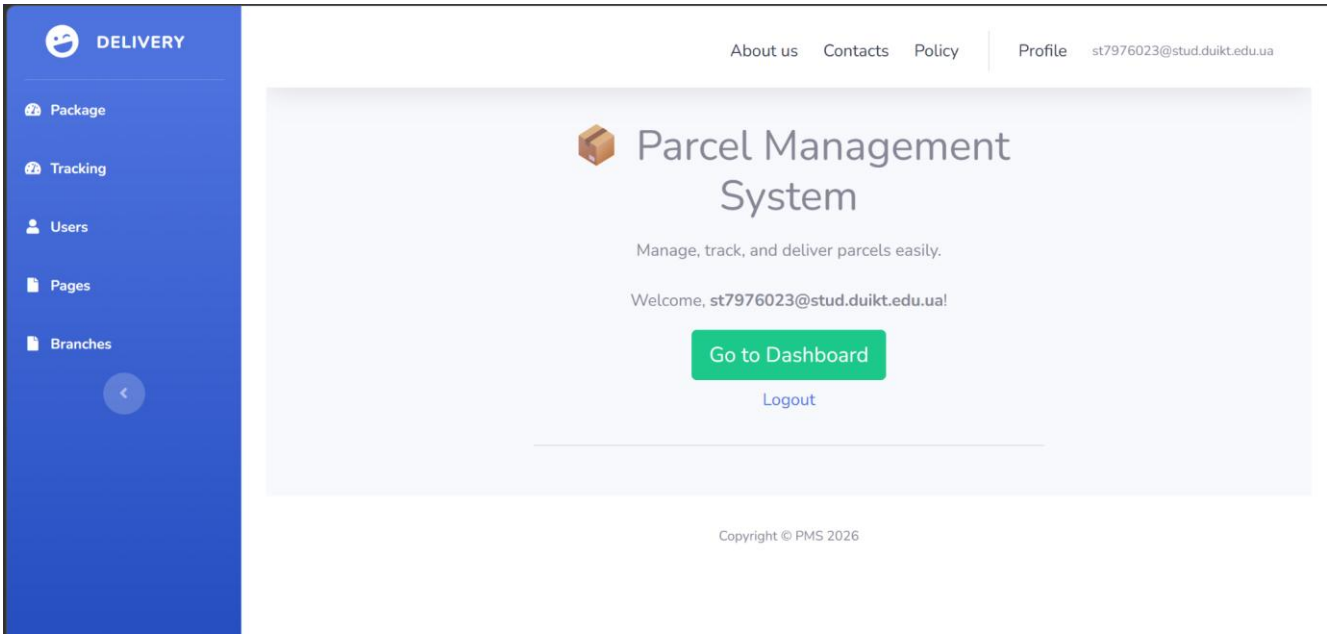


Рис. 4.44 Microsoft Edge

Під час тестування було перевірено основні елементи інтерфейсу:

– головну сторінку;

– сторінку відділень;

– сторінку профілю;

– сторінку бонусів;

– сторінку трекінгу.

За підсумками тестування можна сказати, що додаток відповідає стандартам сумісності з різними браузерами, не ламаючи функціонал, на різних платформах.

ВИСНОВКИ

У результаті виконаної дипломної роботи було розроблено веб-додаток для контролю та управління доставками посилок. Цей додаток підвищує прозорість трекінгу посилок та спрощує комунікацію між кур’єром та отримувачем.

Під час роботи було виконано наступні завдання:

1. Проаналізовано існуючі системи доставки посилок. В результаті було виявлено, що в них відсутні можливості трекінгу посилки в реальному часі з відображенням приблизного маршруту руху транспорту та відсутня можливість комунікації отримувача з кур’єром в додатку. В рамках аналізу конкурентів було проаналізовано функціональні можливості таких сервісів як: Укрпошта, Нова пошта.

2. Проаналізовано існуючі методи відстеження поточного місцезнаходження та підходи для обміну повідомленнями між користувачами. В результаті чого, було визначено підходящий метод відстеження локації та спосіб обміну повідомленнями, що дозволяє підвищити прозорість трекінгу посилок та спростити комунікацію з кур’єром.

3. Розроблено алгоритм, для визначення місцезнаходження посилки та обміну повідомлень між отримувачем та кур’єром. Окрім того, розроблено систему бонусів для користувачів у вигляді місій, за проходження яких, користувач отримує знижку на доставку, безкоштовну доставку між відділеннями або кур’єром.

4. Спроектовано базу даних, що містить в собі таблиці, посилок, доставки, чатів, користувачів, місій, нагород, ролей користувачів, логування зміни статусів, транспорту, відділень, що зберігають дані для забезпечення функціонування додатку.

5. Розроблено веб-додаток, використовуючи ASP.NET Core, що містить в собі функціонал створення, редагування, масової зміни статусів, логування зміни статусів, чат між отримувачем та кур’єром, генерацію накладних, трекінг посилок, місій для отримання бонусів, призначення транспорту, зміну ролей користувачів.

6. Проведено тестування системи – підтверджено коректне функціонування модулю авторизації, реєстрації посилок та доставки завдяки модульному тестуванню. Крім цього, було проведено функціональне та кросбраузерне тестування. Підтверджено стабільність роботи системи та адаптивність інтерфейсу.

Результат підтверджує ефективність наданого рішення. Розроблений додаток розширює можливості трекінгу посилок та обміну повідомлень між користувачами в системі.

ПЕРЕЛІК ПОСИЛАНЬ

1. Нова пошта – офіційний сайт [Електронний ресурс]. – Режим доступу: https://novaposhta.ua/ (дата звернення: 16.03.2026).
2. Укрпошта – офіційний сайт [Електронний ресурс]. – Режим доступу: https://www.ukrposhta.ua/ (дата звернення: 16.03.2026).
3. ASP.NET Core MVC [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-10.0
4. Паламарчук М.О., Герцюк М.М. Методи відстеження посилок та комунікації в реальному часі у веб-сервісах доставки. // VII Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях».(Подано до друку) 20.04.2026
5. Паламарчук М.О., Герцюк М.М. Цифровий веб-сервіс для контролю та керування доставками посилок на основі ASP.NET Core.// Всеукраїнська науково-технічна конференція «Технології цифрового розвитку: програмні системи та децентралізація».(Подано до друку) 25.04.2026
6. Meest Express – офіційний сайт [Електронний ресурс]. – Режим доступу: https://ua.meest.com/ (дата звернення: 16.03.2026).
7. IP Tracking: What Is It, and How Does It Work [Електронний ресурс]. – Режим доступу: https://www.leadforensics.com/blog/ip-tracking-how-does-ip-tracking-work/ (дата звернення: 21.03.2026).
8. 4 geolocation technologies [Електронний ресурс]. – Режим доступу: https://www.sensolus.com/knowledge-hub/behind-the-tech/four-geolocation-technologies-compared-how-can-they-improve-your-operational-efficiency/ (дата звернення: 21.03.2026).
9. HTML Geolocation API [Електронний ресурс]. – Режим доступу: https://www.w3schools.com/html/html5_geolocation.asp (дата звернення: 21.03.2026).
10. Polling and Long-Polling: Essential Techniques for Real-Time Web Applications [Електронний ресурс]. – Режим доступу: https://kennethnnalue.medium.com/polling-and-long-polling-essential-techniques-for-real-time-web-applications-12868b13f21b (дата звернення: 21.03.2026).
11. What is SignalR? A Real-Time Communication Framework for .NET [Електронний ресурс]. – Режим доступу: https://dev.to/dazevedo/what-is-signalr-a-real-time-communication-framework-for-net-29h0 (дата звернення: 21.03.2026).
12. Visual Studio [Електронний ресурс]. – Режим доступу: https://visualstudio.microsoft.com/ (дата звернення: 26.03.2026).
13. Configure ASP.NET Core Identity [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/aspnet/core/security/authentication/identity-configuration?view=aspnetcore-10.0 (дата звернення: 01.04.2026).

14. Entity Framework Core [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/ef/core/ (дата звернення: 27.03.2026).
15. Razor View [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/aspnet/core/mvc/views/razor?view=aspnetcore-10.0 (дата звернення: 01.04.2026).
16. SQL Server Management Studio [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/ssms/
17. Logistics UX/UI Design: 5 Principles to Move Your Business Forward [Електронний ресурс]. – Режим доступу: https://www.lazarev.agency/articles/logistics-ux-ui-design (дата звернення: 10.04.2026).
18. What Is Bootstrap and Who Uses It? [Електронний ресурс]. – Режим доступу: https://www.coursera.org/articles/what-is-bootstrap (дата звернення: 10.04.2026).
19. xUnit [Електронний ресурс]. – Режим доступу: https://xunit.net/ (дата звернення: 16.04.2026).
20. Moq [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/shows/visual-studio-toolbox/unit-testing-moq-framework (дата звернення: 16.04.2026).
21. 6 Major Reasons You Need to Implement Cross-Browser Testing [Електронний ресурс]. – Режим доступу: https://www.testdevlab.com/blog/why-you-need-cross-browser-testing (дата звернення: 17.04.2026).
22. Browserstack [Електронний ресурс]. – Режим доступу: https://www.browserstack.com/ (дата звернення: 17.04.2026).

ДОДАТОК А. ПРЕЗЕНТАЦІЯ ДОДАТКА

Державний університет
інформаційно-комунікаційних технологій
Навчально-науковий інститут інформаційних технологій
кафедра Технологій цифрового розвитку

Цифровий сервіс для контролю та керування доставками посилок на основі технології ASP.NET Core

Виконав:
студент 4 курсу
групи ТЦР-44
Паламарчук М. О.
Керівник роботи:
д-р філософії, доцент кафедри ТЦР Герцюк Микола Модестович

Київ 2026

Актуальність дослідження

Існуючі системи мають доволі обмежений функціонал, але кінцеві користувачі очікують від системи не лише базовий функціонал відправлення та отримання посилок, а й додаткових можливостей, відстеження маршруту доставки, інтерактивні мапи, бонусної політики, можливість комунікувати з кур'єром в системі, мульти трекінг. Тому є необхідність створення систем, що поєднують в собі функціонал управління доставкою, трекінг посилок, елементи гейміфікації, які будуть виступати стимулятором для постійного використання додатку. Таким чином, проектування системи управління доставками посилок є актуальним завданням, в результаті виконання якого має зменшитись навантаження на працівників, покращитись взаємодія з кінцевим користувачем.

Мета, об’єкт та предмет дослідження

Метою роботи є автоматизація процесів створення, обробки, відстеження та контролю доставки посилок, а також підвищення ефективності взаємодії між користувачами, кур’єрами та адміністраторами системи за допомогою сучасних веб-технологій.

Об’єктом дослідження є процеси управління, відстеження поштових відправлень та методи обміну повідомленнями в інформаційних системах.

Предметом дослідження є методи для трекінгу та управління доставками посилок та методи обміну повідомленнями.

Завдання

Для досягнення мети потрібно виконати поставленні завдання перелічені нижче:

1. Провести аналіз інформаційних систем доставки посилок.
2. Дослідити функціональні можливості існуючих систем доставки.
3. Визначити основні вимоги до додатку.
4. Розробити базу даних.
5. Спроектувати архітектуру програмного забезпечення.
6. Розробити веб-додаток, використовуючи ASP.NET Core, що містить в собі функціонал створення, редагування, масової зміни статусів, логування зміни статусів, чат між отримувачем та кур’єром, генерацію накладних, трекінг посилок, місій для отримання бонусів, призначення транспорту, зміну ролей користувачів
7. Для оцінки ефективності протестувати розроблену систему.

Методи дослідження

Методами дослідження є аналіз джерел у сфері електронної комерції, існуючих інформаційних системи доставки, методи виявлення поточного місцезнаходження, способи обміну повідомленнями в реальному часі, методи об’єктно орієнтованого програмування, методи проектування баз даних, методи розробки веб-додатків, методи тестування програмного забезпечення, використовуючи ASP.NET Core.

5

Аналіз аналогів

Аналіз аналогів показав:

– Існуючі додатки (Укрпошта, Нова пошта) не мають системи обміну повідомленнями між учасниками доставки.

– Відсутність функціоналу мульти трекінгу посилок.

– Система лояльності відсутня для більшості користувачів.

– Відсутній функціонал перегляду місцезнаходження посилки в реальному часі на карті.

– Обмежений функціонал масової зміни статусів посилок.

6

Порівняння аналогів

Функція	Укрпошта	Нова пошта	PackageManagementSystem
Відстеження посилки за трек-номером	+	+	+
Мульти трекінг	-	-	+
Генерація PDF-накладної	+	+	+
Онлайн-чат з кур'єром	-	-	+
Масова зміна статусів посилок	Частково	+	+
Система лояльності	-	Частково	+
Перегляд місцезнаходження посилки в реальному часі на карті	-	-	+

7

Вимоги до програмного забезпечення

Функціональні:

– Авторизація та реєстрація користувачів

– CRUD-операції для посилки.

– Логування зміни статусів посилки.

– Генерація накладної в форматі PDF.

– Мульти трекінг посилок.

– Система обміну повідомленнями між отримувачем та кур'єром.

– Масова зміна статусів посилок.

– Отримання та застосування бонусів з програми лояльності.

– Відображення маршруту посилки та розташуванням водія на карті.

– Відображення активних відділень на мапі.

Нефункціональні:

– Інтерфейс має бути інтуїтивно зрозумілим та зручним у користуванні.

– Інтерфейс має бути адаптивним і коректно відображатися на сучасних пристроях.

– Необхідно передбачити обробку помилок та виняткових ситуацій, які можуть призвести до збою системи.

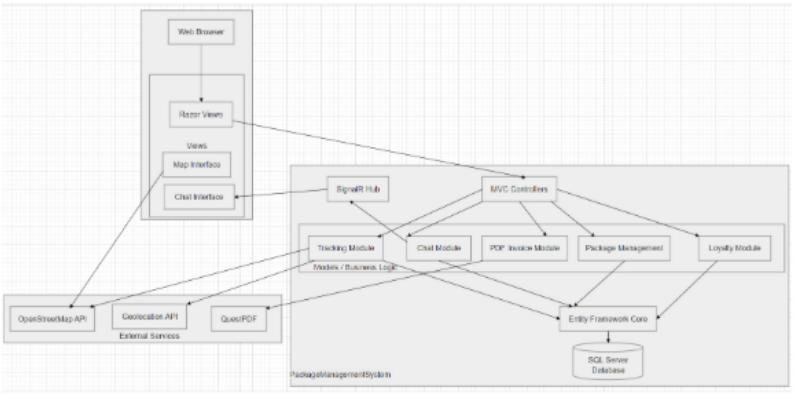
– Доступ до API повинен бути реалізованим з використанням диференціації прав доступу на основі ролей користувачів.

8

Архітектура системи

Система PackageManagementSystem побудована за архітектурою (Three-Tier Architecture). Основними компонентами якої є

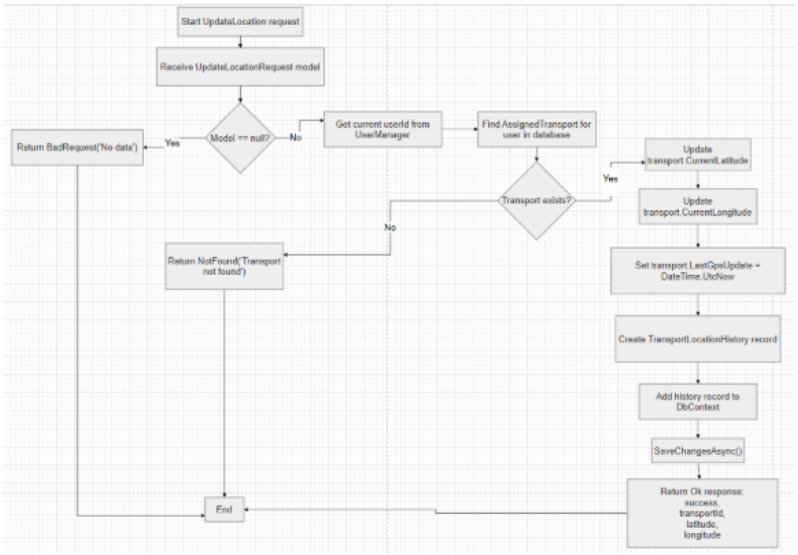
– Controllers(Controller Layer)
– Services(Business Logic Layer)
– База даних(Data Access Layer)
– Views(Presentation Layer)
– External Services Layer



9

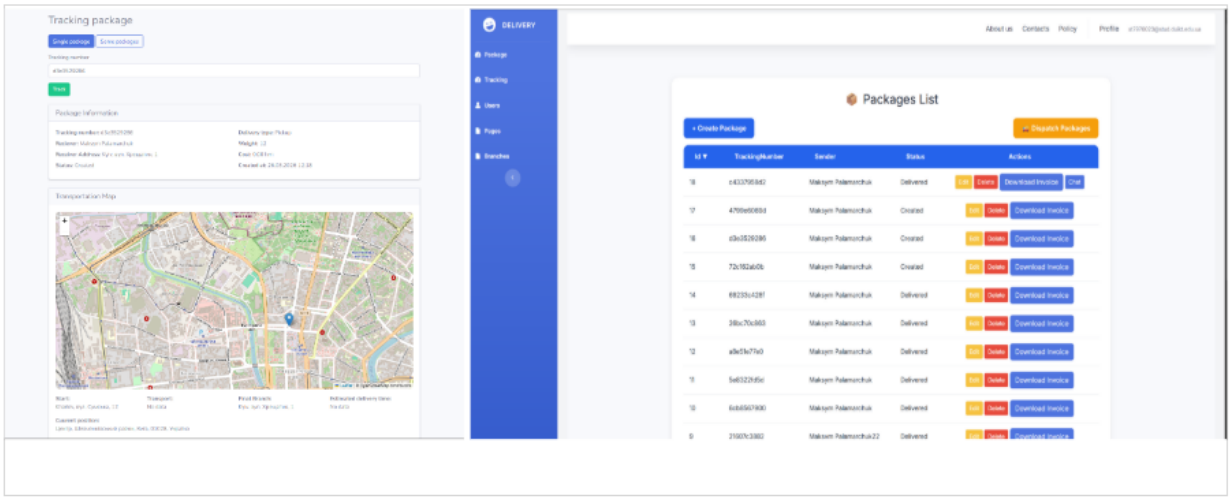
Алгоритм оновлення геолокації водія

Реалізований за допомогою Geolocation API, що за допомогою браузера запитує у користувача дозвіл на доступ до даних про його місцезнаходження. Якщо користувач надасть дозвіл, браузер визначить координати користувача та запише їх в базу даних для відображення на карті при відстеженні посилки користувачем.



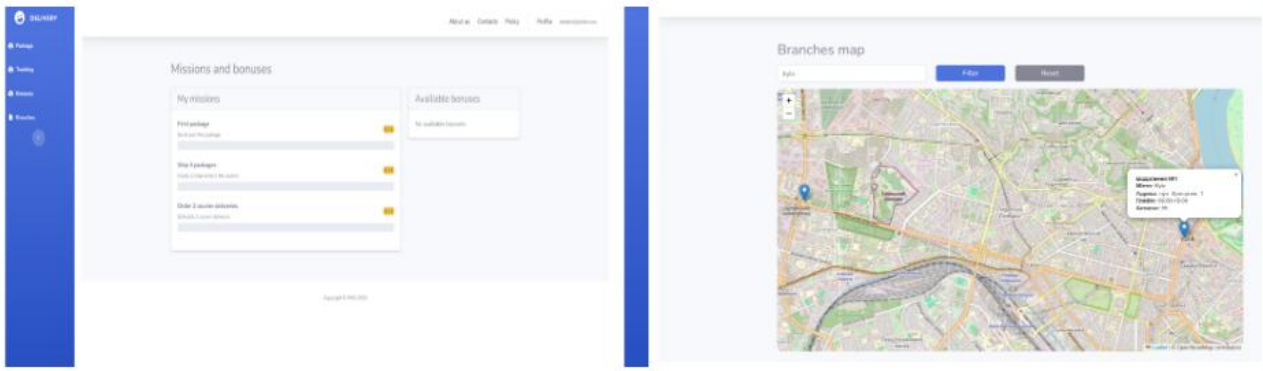
10

Інтерфейс користувача



11

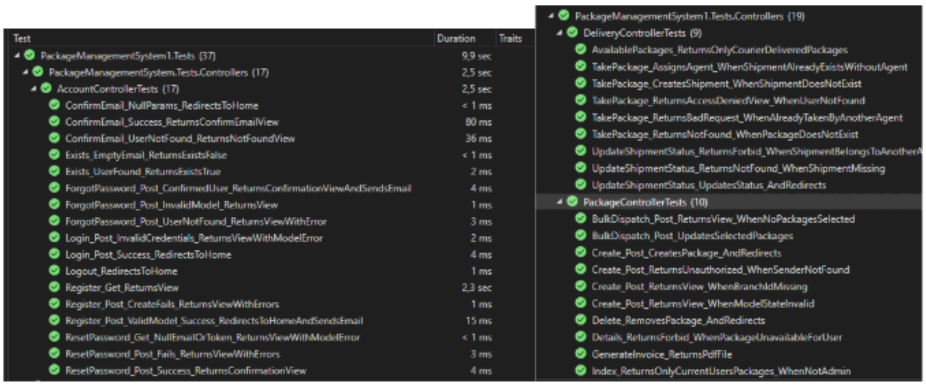
Інтерфейс користувача



12

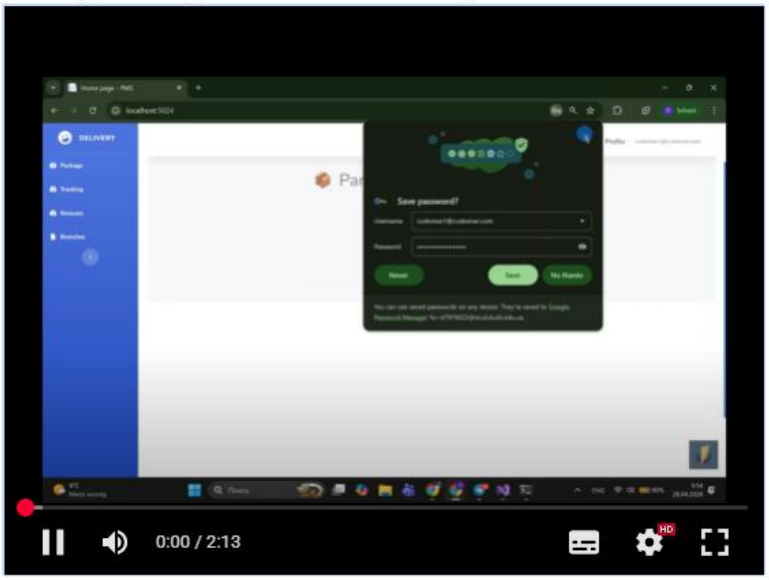
Тестування системи

Для перевірки працездатності системи було проведено модульне тестуванню за допомогою xUnit. Перевірено основні сценарії використання: для модулю авторизації, реєстрації посилок та доставки. Крім цього, було проведено Monkey та кросбраузерне тестування. У результаті проведеного тестування було виправлено 10 дефектів модульному та Monkey тестуванню. Пасрейт тестів наразі складає 100%.



13

Демонстрація роботи додатку



14

Апробація результатів дослідження

1. Паламарчук М.О., Герцюк М.М. Методи відстеження посилок та комунікації в реальному часі у веб-сервісах доставки. // VII Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях».
2. Паламарчук М.О., Герцюк М.М. Цифровий веб-сервіс для контролю та керування доставками посилок на основі ASP.NET Core.// Всеукраїнської науково-технічної конференції «Технології цифрового розвитку: програмні системи та децентралізація»

15

Висновки

1. Проаналізовано існуючі системи доставки посилок. В результаті було виявлено, що в них відсутні можливості трекінгу посилки в реальному часі з відображенням приблизного маршруту руху транспорту та відсутня можливість комунікації отримувача з кур'єром в додатку. В рамках аналізу конкурентів було проаналізовано функціональні можливості таких сервісів як: Укрпошта, Нова пошта.
2. Проаналізовано існуючі методи відстеження поточного місцезнаходження та підходи для обміну повідомленнями між користувачами. В результаті чого, було визначено підходящий метод відстеження локації та спосіб обміну повідомленнями, що дозволяє підвищити прозорість трекінгу посилок та спростити комунікацію з кур'єром.
3. Розроблено алгоритм, для визначення місцезнаходження посилки та обміну повідомлень між отримувачем та кур'єром. Окрім того, розроблено систему бонусів для користувачів у вигляді місій, за проходження яких, користувач отримує знижку на доставку, безкоштовну доставку між відділеннями або кур'єром.
4. Спроектовано базу даних, що містить в собі таблиці, посилок, доставки, чатів, користувачів, місій, нагород, ролей користувачів, логування зміни статусів, транспорту, відділень, що зберігають дані для забезпечення функціонування додатку.
5. Розроблено веб-додаток, використовуючи ASP.NET Core, що містить в собі функціонал створення, редагування, масової зміни статусів, логування зміни статусів, чат між отримувачем та кур'єром, генерацію накладних, трекінг посилок, місій для отримання бонусів, призначення транспорту, зміну ролей користувачів.
6. Проведено тестування системи - підтверджено коректне функціонування модулю авторизації, реєстрації посилок та доставки завдяки модульному тестуванню. Крім цього, було проведено функціональне та кросбраузерне тестування

16

Дякую за увагу

ДОДАТОК Б. КОД ОСНОВНИХ МОДУЛІВ

```
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using PackageManagementSystem.Models;
using PackageManagementSystem.Services;
using PackageManagementSystem.ViewModels;
namespace PackageManagementSystem.Controllers
{
   [Route("account")]
   public class AccountController : Controller
   {
      private readonly UserManager<ApplicationUser>
_userManager;
      private readonly SignInManager<ApplicationUser>
_signInManager;
      private readonly IEmailSender _emailSender;
    public
AccountController(UserManager<ApplicationUser>
userManager, SignInManager<ApplicationUser>
signInManager, IEmailSender emailSender)
      {
         _userManager = userManager;
         _signInManager = signInManager;
         _emailSender = emailSender;
      }
      [Route("register")]
      [HttpGet]
      public IActionResult Register() => View();
      [Route("register")]
      [HttpPost]
      [ValidateAntiForgeryToken]
      public async Task<IActionResult>
Register(RegisterVM registerVM)
      {
         if (ModelState.IsValid)
         {
            var user = new ApplicationUser { UserName =
registerVM.Email, Email = registerVM.Email, FullName =
registerVM.FullName };
            var resultOfCreation = await
_userManager.CreateAsync(user, registerVM.Password!);
            if (resultOfCreation.Succeeded)
            {
               await _userManager.AddToRoleAsync(user,
"Customer");
               await _signInManager.SignInAsync(user,
false);
               var token = await
_userManager.GenerateEmailConfirmationTokenAsync(use
r);
               var confirmEmailLink =
Url.Action("ConfirmEmail", "Account",
                  new {userId = user.Id, token = token},
Request.Scheme );
               await _emailSender.SendEmailAsync(
                   user.Email!,
                  "Confirm Your Email",
                  $"<p>Click the link to confirm your
email:</p><a href='{confirmEmailLink}'>Comfirm</a>"
                  );
               return RedirectToAction("Index", "Home");
            }
            foreach (var error in resultOfCreation.Errors)
            {
               ModelState.AddModelError(string.Empty,
error.Description);
            }
         }
         return View(registerVM);
      }
      public IActionResult Index()
      {
         return View();
      }
      [Route("login")]
      [HttpGet]
      public IActionResult Login()
      {
         return View();
      }
      [Route("login")]
      [HttpPost]
      [ValidateAntiForgeryToken]
      public async Task<IActionResult> Login(LoginVM
loginVM)
      {
         if (ModelState.IsValid) {
            var loginResult = await
_signInManager.PasswordSignInAsync(loginVM.Email!,
loginVM.Password!, false, false);
            if (loginResult.Succeeded)
            {
               return RedirectToAction("Index", "Home");
            }
            ModelState.AddModelError("", "Invalid
credentials");
         }
         return View(loginVM);
      }
      [Route("forgot-password")]
      [HttpGet]
      public IActionResult ForgotPassword() => View();
      [Route("forgot-password")]
      [HttpPost]
      public async Task<IActionResult>
ForgotPassword(ForgotPasswordVM forgotPasswordVM)
      {
         if (!ModelState.IsValid)
         {
            ModelState.AddModelError("", "Invalid form
data");
            return View(forgotPasswordVM);
         }
         var user = await
_userManager.FindByEmailAsync(forgotPasswordVM.Emai
l!);
         if (user != null && await
_userManager.IsEmailConfirmedAsync(user))
         {
            var token = await
_userManager.GeneratePasswordResetTokenAsync(user);
            var resetLink = Url.Action("ResetPassword",
"Account", new { email = forgotPasswordVM.Email, token
}, Request.Scheme);
            await _emailSender.SendEmailAsync(
                  user.Email!,
                  "Reset your password",
                  $"<p>Click the link to reset your
password:</p><a href='{resetLink}'>Reset Password</a>");
            return View("ForgotPasswordConfirmation");
         }
         ModelState.AddModelError("", "User not found or
email not confirmed");
         return View(forgotPasswordVM);
      }
      [Route("reset-password")]
      [HttpGet]
      public IActionResult ResetPassword(string email,
string token)
      {
         if( email == null || token == null)
         {
```

```
            ModelState.AddModelError("", "Invalid password
reset token");
        }
        return View();
    }
    [Route("reset-password")]
    [HttpPost]
    public async Task<IActionResult>
ResetPassword(ResetPasswordVM resetPasswordVM)
    {
        if(ModelState.IsValid)
        {
            var user = await
_userManager.FindByEmailAsync(resetPasswordVM.Email
);
            if(user != null)
            {
                var result = await
_userManager.ResetPasswordAsync(user,
resetPasswordVM.Token, resetPasswordVM.Password);
                if(result.Succeeded)
                {
                    return View("ResetPasswordConfirmation");
                }
                foreach (var error in result.Errors)
                {
                    ModelState.AddModelError("",
error.Description);
                }
                return View(resetPasswordVM);
            }
            return View("ResetPasswordConfirmation");
        }

        return View(resetPasswordVM);
    }
    [HttpGet]
    public async Task<IActionResult>
ConfirmEmail(string userId, string token)
    {
        if (userId == null || token == null)
        {
            return RedirectToAction("Index", "Home");
        }
        var user = await
_userManager.FindByIdAsync(userId);
        if (user == null)
        {
            ViewBag.ErrorMassage = $"User ID {userId} is
invalid";
            return View("Not found");
        }
        var result = await
_userManager.ConfirmEmailAsync(user, token);
        if (result.Succeeded)
        {
            return View("ConfirmEmail");
        }
        ViewBag.ErorTitle = "Email cannot be confirmed";
        return View();
    }
    [Route("logout")]
    [HttpPost]
    public async Task<IActionResult> Logout()
    {
        await _signInManager.SignOutAsync();
        return RedirectToAction("Index", "Home");
    }
    [HttpGet("exists")]
    public async Task<IActionResult> Exists([FromQuery]
string email)
    {
        if (string.IsNullOrWhiteSpace(email))
            return Ok(new { exists = false });
        var user = await
_userManager.FindByEmailAsync(email.Trim());
        return Ok(new { exists = user != null });
    }
}
```

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
using PackageManagementSystem.Data;
using PackageManagementSystem.Models;
using PackageManagementSystem.ViewModels;
namespace PackageManagementSystem.Controllers
{
    [Authorize(Roles = "Admin")]
    [Route("admin")]
    public class AdminController : Controller
    {
        private readonly UserManager<ApplicationUser>
_userManager;
        private readonly RoleManager<IdentityRole>
_roleManager;
        private readonly
PackageManagementSystemDbContext _context;


        public AdminController(
            UserManager<ApplicationUser> userManager,
            RoleManager<IdentityRole> roleManager,
            PackageManagementSystemDbContext context)
        {
            _userManager = userManager;
            _roleManager = roleManager;
            _context = context;
        }
        [HttpGet("users")]
        public async Task<IActionResult> Users()
        {
            var users = await
_userManager.Users.ToListAsync();
            var result = new
List<UserRoleListItemViewModel>();
            foreach (var user in users)
            {
                var roles = await
_userManager.GetRolesAsync(user);
                result.Add(new UserRoleListItemViewModel
                {
                    UserId = user.Id,
                    Email = user.Email ?? "",
                    FullName = user.FullName ?? "",
                    Role = roles.FirstOrDefault()
                });
            }
            return View(result);
        }
        [HttpGet("change-role/{userId}")]
        public async Task<IActionResult> ChangeRole(string
userId)
        {
            var user = await
_userManager.FindByIdAsync(userId);
            if (user == null)
                return NotFound();
            var currentRoles = await
_userManager.GetRolesAsync(user);
            var allRoles = await _roleManager.Roles
                .Select(r => r.Name!)
                .ToListAsync();
            var vm = new ChangeUserRoleViewModel
            {
                UserId = user.Id,
                Email = user.Email ?? "",
                FullName = user.FullName ?? "",
                CurrentRole = currentRoles.FirstOrDefault(),
                NewRole = currentRoles.FirstOrDefault() ?? "",
                AvailableRoles = allRoles.Select(role => new
SelectListItem
                {
                    Value = role,
                    Text = role
                }).ToList()
```

```
        };
        return View(vm);
    }
    [HttpPost("change-role/{userId}")]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> ChangeRole(string
userId, ChangeUserRoleViewModel vm)
    {
        if (userId != vm.UserId)
            return BadRequest();
        var user = await
_userManager.FindByIdAsync(vm.UserId);
        if (user == null)
            return NotFound();


        if (string.IsNullOrWhiteSpace(vm.NewRole))
        {
            ModelState.AddModelError(nameof(vm.NewRol
e), "Choose new role.");
        }
        if (!await
_roleManager.RoleExistsAsync(vm.NewRole))
        {
            ModelState.AddModelError(nameof(vm.NewRol
e), "Choosed role doesn't exist.");
        }
        if (!ModelState.IsValid)
        {
            var allRoles = await _roleManager.Roles
                .Select(r => r.Name!)
                .ToListAsync();
            vm.AvailableRoles = allRoles.Select(role => new
SelectListItem
            {
                Value = role,
                Text = role
            }).ToList();
            return View(vm);
        }
        var currentRoles = await
_userManager.GetRolesAsync(user);
        if (currentRoles.Any())
        {
            var removeResult = await
_userManager.RemoveFromRolesAsync(user,
currentRoles);
            if (!removeResult.Succeeded)
            {
                ModelState.AddModelError(string.Empty,
"Failed to delete current role.");
            }
        }
        if (!ModelState.IsValid)
        {
            var allRoles = await _roleManager.Roles
                .Select(r => r.Name!)
                .ToListAsync();
            vm.AvailableRoles = allRoles.Select(role => new
SelectListItem
            {
                Value = role,
                Text = role
            }).ToList();
            return View(vm);
        }
        var addResult = await
_userManager.AddToRoleAsync(user, vm.NewRole);
        if (!addResult.Succeeded)
        {
            ModelState.AddModelError(string.Empty, "Failed
to assign new role.");
            var allRoles = await _roleManager.Roles
                .Select(r => r.Name!)
                .ToListAsync();
            vm.AvailableRoles = allRoles.Select(role => new
SelectListItem
            {
                Value = role,
                Text = role
            }).ToList();
            return View(vm);
        }


        TempData["SuccessMessage"] = $"User role
{user.Email} changed on {vm.NewRole}.";
        return RedirectToAction(nameof(Users));
    }
    [HttpGet("pages")]
    public async Task<IActionResult> Pages()
    {
        var pages = await _context.Pages!
            .AsNoTracking()
            .OrderBy(x => x.Title)
            .ToListAsync();
        return View(pages);
    }
    [HttpGet("pages/edit/{id}")]
    public async Task<IActionResult> EditPage(int id)
    {
        var page = await _context.Pages!.FindAsync(id);
        if (page == null)
            return NotFound();
        var vm = new PageVM
        {
            Id = page.Id,
            Title = page.Title,
            Slug = page.Slug,
            ShortDescription = page.ShortDescription,
            Description = page.Description,
            PhotoUrl = page.PhotoUrl,
        };
        return View(vm);
    }
    [HttpPost("pages/edit/{id}")]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> EditPage(int id,
PageVM vm)
    {
        if (id != vm.Id)
            return BadRequest();
        var page = await _context.Pages!.FindAsync(id);
        if (page == null)
            return NotFound();
        if (!ModelState.IsValid)
            return View(vm);
        var slugExists = await _context.Pages
            .AnyAsync(x => x.Slug == vm.Slug && x.Id !=
vm.Id);
        if (slugExists)
        {
            ModelState.AddModelError(nameof(vm.Slug),
"Page with this slug already exists.");
            return View(vm);
        }
        page.Title = vm.Title;
        page.Slug = vm.Slug.Trim().ToLower();
        page.ShortDescription = vm.ShortDescription;
        page.Description = vm.Description;
        page.PhotoUrl = vm.PhotoUrl;
        await _context.SaveChangesAsync();
        TempData["SuccessMessage"] = "Page updated
successfully.";
        return RedirectToAction(nameof(Pages));
    }
  }
}
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using PackageManagementSystem.Data;
using PackageManagementSystem.ViewModels;
using System.Security.Claims;
namespace PackageManagementSystem.Controllers
{
  [Authorize(Roles = "Customer")]
  public class BonusesController : Controller
```

```
    {
        private readonly PackageManagementSystemDbContext _context;
        public BonusesController(PackageManagementSystemDbContext context)
        {
            _context = context;
        }
        public async Task<IActionResult> Index()
        {
            var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);
            if (string.IsNullOrEmpty(userId))
                return Unauthorized();
            var missions = await _context.Missions
                .Where(m => m.IsActive)
                .OrderBy(m => m.Id)
                .ToListAsync();
            var progresses = await _context.UserMissionProgresses
                .Where(p => p.UserId == userId)
                .ToListAsync();
            var rewards = await _context.UserRewards
                .Where(r => r.UserId == userId && !r.IsUsed)
                .OrderBy(r => r.CreatedAt)
                .ToListAsync();


            var model = new BonusIndexVM();
            foreach (var mission in missions)
            {
                var progress = progresses.FirstOrDefault(p => p.MissionId == mission.Id);
                model.Missions.Add(new MissionProgressItemVM
                {
                    Title = mission.Title,
                    Description = mission.Description,
                    CurrentValue = progress?.CurrentValue ?? 0,
                    TargetValue = mission.TargetValue,
                    IsCompleted = progress?.IsCompleted ?? false
                });
            }
            model.AvailableRewards = rewards
                .Select(r => new RewardItemVM
                {
                    RewardType = r.RewardType,
                    Value = r.Value
                })
                .ToList();
            return View(model);
        }
    }
}
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using PackageManagementSystem.Data;
using PackageManagementSystem.Models;
namespace PackageManagementSystem.Controllers
{
    [Authorize]
    [Route("{shipmentId}/chat")]
    public class ChatController : Controller
    {
        private readonly PackageManagementSystemDbContext _context;
        private readonly UserManager<ApplicationUser> _userManager;
        public ChatController(PackageManagementSystemDbContext context, UserManager<ApplicationUser> userManager)
        {
            _context = context;
            _userManager = userManager;
        }
        [HttpGet]
        public async Task<IActionResult> Index(int shipmentId)
        {
            var user = await _userManager.GetUserAsync(User);
            var shipment = await _context.Shipments
                .Include(s => s.Package)
                .FirstOrDefaultAsync(s => s.Id == shipmentId);
            if (shipment == null)
                return NotFound();
            if (user!.Id != shipment.Package.ReceiverId &&
                user.Id != shipment.DeliveryAgentId)
                return Forbid();
            return View(shipmentId);
        }
        [HttpGet("history")]
        public async Task<IActionResult> GetHistory(int shipmentId)
        {
            var user = await _userManager.GetUserAsync(User);
            var shipment = await _context.Shipments
                .Include(s => s.Package)
                .Include(s => s.Chat)
                    .ThenInclude(c => c.Messages)
                .FirstOrDefaultAsync(s => s.Id == shipmentId);


            if (shipment == null)
                return NotFound();
            if (user!.Id != shipment.Package.ReceiverId &&
                user.Id != shipment.DeliveryAgentId)
                return Forbid();
            if (shipment.Chat == null)
                return Ok(new List<object>());
            var senderIds = shipment.Chat.Messages
                .Select(m => m.SenderId)
                .Distinct()
                .ToList();
            var users = await _userManager.Users
                .Where(u => senderIds.Contains(u.Id))
                .ToDictionaryAsync(
                    u => u.Id,
                    u => !string.IsNullOrWhiteSpace(u.FullName) ? u.FullName : u.UserName!);
            var messages = shipment.Chat.Messages
                .OrderBy(m => m.SentAt)
                .Select(m => new
                {
                    m.Id,
                    m.SenderId,
                    SenderName = users.ContainsKey(m.SenderId) ? users[m.SenderId] : "User",
                    m.Text,
                    m.SentAt
                });
            return Ok(messages);
        }
    }
}
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using PackageManagementSystem.Data;
using PackageManagementSystem.Models;
namespace PackageManagementSystem.Controllers
{
    [Authorize(Roles = "DeliveryAgent,Admin")]
    public class DeliveryController : Controller
    {
        private readonly PackageManagementSystemDbContext _context;
        private readonly UserManager<ApplicationUser> _userManager;
        public DeliveryController(PackageManagementSystemDbContext context, UserManager<ApplicationUser> userManager)
        {
            _context = context;
```

```
        _userManager = userManager;
    }
    [HttpGet]
    [Route("pack")]
    public async Task<IActionResult> AvailablePackages()
    {
        var user = await _userManager.GetUserAsync(User);
        ViewBag.CurrentUserId = user!.Id;
        var packages = await _context.Packages
            .Include(p => p.Sender)
            .Include(p => p.Shipment)
            .Where(p => p.DeliveryType == DeliveryType.Courier && p.Status == Status.Delivered)
            .ToListAsync();
        return View(packages);
    }



    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> TakePackage(int packageId)
    {
        var deliveryAgent = await _userManager.GetUserAsync(User);
        if (deliveryAgent == null) return View("AccessDenied", "Home");
        var package = await _context.Packages
            .Include(p => p.Shipment)
            .FirstOrDefaultAsync(p => p.Id == packageId);
        if (package == null) return NotFound();
        if (package.Shipment != null && package.Shipment.DeliveryAgentId != null &&
            package.Shipment.DeliveryAgentId != deliveryAgent.Id)
        {
            return BadRequest("Package already taken by another delivery agent.");
        }
        if (package.Shipment == null)
        {
            var shipment = new Shipment
            {
                PackageId = package.Id,
                DeliveryAgentId = deliveryAgent.Id,
                TakenAt = DateTime.UtcNow,
                ShipmentStatus = ShipmentStatus.Taken,
                Chat = new Chat()
            };
            _context.Shipments.Add(shipment);
        }
        else
        {
            package.Shipment.DeliveryAgentId = deliveryAgent.Id;
        }
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(AvailablePackages));
    }
    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> UpdateShipmentStatus(int shipmentId, ShipmentStatus status)
    {
        var user = await _userManager.GetUserAsync(User);
        var shipment = await _context.Shipments
            .Include(s => s.Package)
            .FirstOrDefaultAsync(s => s.Id == shipmentId);
        if (shipment == null) return NotFound();
        if (shipment.DeliveryAgentId != user!.Id) return Forbid();
        shipment.ShipmentStatus = status;
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(AvailablePackages));
    }
  }
}
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
using PackageManagementSystem.Data;
using PackageManagementSystem.Models;
using PackageManagementSystem.Services;
using PackageManagementSystem.Services.Pdf;
using PackageManagementSystem.ViewModels;
using System.IO.Pipelines;
using System.Security.Claims;
namespace PackageManagementSystem.Controllers
{
  [Authorize]
  [Route("packages")]
  public class PackageController : Controller
  {
    private readonly PackageManagementSystemDbContext? _context;
    private readonly UserManager<ApplicationUser> _userManager;
    private readonly ILogger<PackageController> _logger;
    private readonly IPdfService _pdfService;
    private readonly IDeliveryPricingService _pricingService;
    private readonly IRewardService _rewardService;
    private readonly IMissionService _missionService;
    public PackageController(PackageManagementSystemDbContext context,
                             UserManager<ApplicationUser> userManager, ILogger<PackageController> logger,
                             IPdfService pdfService, IDeliveryPricingService deliveryPricingService,
                             IRewardService rewardService, IMissionService missionService)
    {
      _context = context;
      _userManager = userManager;
      _logger = logger;
      _pdfService = pdfService;
      _pricingService = deliveryPricingService;
      _rewardService = rewardService;
      _missionService = missionService;
    }
    [Authorize]
    [Route("index")]
    [HttpGet]
    public async Task<IActionResult> Index(string sort = "Id", string direction = "desc")
    {
      var userId = _userManager.GetUserId(User);
      var q = _context!.Packages
        .Include(p => p.Sender)
        .Include(p => p.Shipment)
        .Where(p => (p.SenderId == userId || p.ReceiverId == userId) || User.IsInRole("Admin"))
        .AsQueryable();
      bool desc = string.Equals(direction, "desc", StringComparison.OrdinalIgnoreCase);
      q = sort switch
      {
        "TrackingNumber" => desc ? q.OrderByDescending(p => p.TrackingNumber) : q.OrderBy(p => p.TrackingNumber),
        "Sender" => desc ? q.OrderByDescending(p => p.Sender!.FullName) : q.OrderBy(p => p.Sender!.FullName),
        "Status" => desc ? q.OrderByDescending(p => p.Status) : q.OrderBy(p => p.Status),
        _ => desc ? q.OrderByDescending(p => p.Id) : q.OrderBy(p => p.Id),
      };
      var packages = await q.ToListAsync();
      ViewBag.Sort = sort;
```

```
        ViewBag.Dir = desc ? "desc" : "asc";
        return View(packages);
    }
    [Route("create")]
    [HttpGet]
    public IActionResult Create() {
        ViewBag.Branches = _context!.Branches.Where(b
=> b.IsActive)
            .OrderBy(b => b.City).ThenBy(b => b.Name)
            .ToList();
        return View();
    }
    [Route("create")]
    [HttpPost]
    public async Task<IActionResult>
Create(CreatePackageVM createPackageVM)
    {
        if (!ModelState.IsValid)
            return View(createPackageVM);
        var getSender = await _userManager.Users
            .FirstOrDefaultAsync(x => x.UserName ==
User.Identity!.Name);
        if (getSender == null)
            return Unauthorized();
        if (!createPackageVM.BranchId.HasValue)
        {
            ModelState.AddModelError(nameof(createPackag
eVM.BranchId), "Choose branch");
            return View(createPackageVM);
        }
        var branch = await _context.Branches
            .FirstOrDefaultAsync(b => b.Id ==
createPackageVM.BranchId.Value && b.IsActive);
        if (branch == null)
        {
            ModelState.AddModelError(nameof(createPackag
eVM.BranchId), "Branch not found");
            return View(createPackageVM);
        }
        var package = new Package();
        if (createPackageVM.DeliveryType ==
DeliveryType.Courier)
        {
            var receiver = await
_userManager.FindByEmailAsync(createPackageVM.Recei
verEmail!);
            if (receiver == null)
            {
                ModelState.AddModelError(nameof(createPac
kageVM.ReceiverEmail),
                    "Courier delivery availiable only for
registered users");
                return View(createPackageVM);
            }
            package.ReceiverId = receiver.Id;
            if
(string.IsNullOrWhiteSpace(createPackageVM.CourierAdd
ress))
            {
                ModelState.AddModelError(nameof(createPac
kageVM.CourierAddress),
                    "Enter delivery address");
                return View(createPackageVM);
            }
            package.CourierAddress =
createPackageVM.CourierAddress.Trim();
        }
        else
        {
            package.CourierAddress = null;
        }
        package.TrackingNumber =
Guid.NewGuid().ToString("N").Substring(0, 10);
        package.SenderId = getSender.Id;
        package.Sender = getSender;
        package.Weight = createPackageVM.Weight;
        package.ReceiverName =
createPackageVM.ReceiverName;
        package.DeliveryType =
createPackageVM.DeliveryType;
        var basePrice =
_pricingService.CalculatePrice(package.Weight,
package.DeliveryType);
        var rewardResult = await
_rewardService.ApplyRewardAsync(
                                User.FindFirstValue(Claim
Types.NameIdentifier)!,
                                package.DeliveryType,
                                basePrice
                            );
        package.OriginalShipmentPrice = basePrice;
        package.ShipmentPrice = rewardResult.FinalPrice;
        package.AppliedRewardType =
rewardResult.AppliedRewardType;
        package.Status = Status.Created;
        package.CreatedAt = DateTime.UtcNow;
        package.BranchId = branch.Id;
        package.SenderBranchId = getSender.BranchId;
        package.ReceiverAddress = $"{branch.City},
{branch.Address}";
        await _context.Packages.AddAsync(package);
        await _context.SaveChangesAsync();
        await
_missionService.ProcessShipmentCreatedAsync(
            User.FindFirstValue(ClaimTypes.NameIdentifier)
!,
            package.DeliveryType == DeliveryType.Courier);
        return RedirectToAction("Index");
    }
    [Authorize]
    [Route("edit")]
    [HttpGet]
    public async Task<IActionResult> Edit(int id)
    {
        var userId = _userManager.GetUserId(User);
        var package = await _context!.Packages
            .FirstOrDefaultAsync(p => (p.Id == id &&
p.SenderId == userId) || (User.IsInRole("Admin") && p.Id
== id));
        if (package == null)
            return Forbid();
        var editPackageVM = new CreatePackageVM()
        {
            Id = package.Id,
            ReceiverName = package.ReceiverName,
            ReceiverAddress = package.ReceiverAddress,
            Weight = package.Weight,
            ShipmentPrice = package.ShipmentPrice,
            DeliveryType = package.DeliveryType,
            Status = package.Status
        };
        return View(editPackageVM);
    }
    [Authorize]
    [Route("edit")]
    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult>
Edit(CreatePackageVM editedPackageVM)
    {
        if (!ModelState.IsValid)
            return View(editedPackageVM);
        var userId = _userManager.GetUserId(User);
        var package = await _context!.Packages
            .FirstOrDefaultAsync(p => (p.Id ==
editedPackageVM.Id && p.SenderId == userId) ||
(User.IsInRole("Admin") && p.Id ==
editedPackageVM.Id));
        if (package == null)
            return Forbid();
        var oldStatus = package.Status;
        package.ReceiverName =
editedPackageVM.ReceiverName;
        package.ReceiverAddress =
editedPackageVM.ReceiverAddress;
        package.Weight = editedPackageVM.Weight;
        package.Status = editedPackageVM.Status!;
```

```
            package.DeliveryType = editedPackageVM.DeliveryType;
            package.ShipmentPrice = _pricingService.CalculatePrice(editedPackageVM.Weight, package.DeliveryType);
            if (oldStatus != editedPackageVM.Status)
            {
                var user = await _userManager.GetUserAsync(User);
                var history = new StatusHistory
                {
                    PackageId = editedPackageVM.Id,
                    OldStatus = oldStatus,
                    NewStatus = editedPackageVM.Status!,
                    ChangedByUserId = user!.Id,
                    ChangedAt = DateTime.UtcNow,
                };
                _context.StatusHistories.Add(history);
            }
            await _context.SaveChangesAsync();
            return RedirectToAction("Index", "Package");
        }
        [Authorize]
        [Route("info")]
        [HttpGet]
        public async Task<IActionResult> Details(int id)
        {
            var userId = _userManager.GetUserId(User);
            var packDetails = await _context!.Packages
                .Include(p => p.Sender)
                .Include(p => p.SenderBranch)
                .Include(p => p.Shipment)
                .FirstOrDefaultAsync(p =>
                    p.Id == id &&
                    (p.SenderId == userId || p.ReceiverId == userId || User.IsInRole("Admin")) );
            if (packDetails == null)
                return Forbid();
            return View(packDetails);
        }
        [Authorize]
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Delete(int id)
        {
            var user = await _userManager.GetUserAsync(User!);
            var package = await _context!.Packages
                .FirstOrDefaultAsync(p => (p.Id == id && p.SenderId == user!.Id.ToString()) || User.IsInRole("Admin"));
            if (package == null)
            {
                TempData["ErrorMessage"] = "Failed to delete package.";
                return RedirectToAction(nameof(Index));
            }
            _context.Packages.Remove(package);
            await _context.SaveChangesAsync();
            return RedirectToAction("Index", "Package");
        }
        [HttpGet]
        [Route("invoice/{id}")]
        public async Task<IActionResult> GenerateInvoice(int id)
        {
            var package = await _context!.Packages
                .Include(p => p.Sender)
                .FirstOrDefaultAsync(p => p.Id == id);
            if (package == null)
                return NotFound();
            var pdfBytes = _pdfService.GenerateInvoice(package);
            return File(pdfBytes,
                    "application/pdf",
                    $"Invoice_{package.TrackingNumber}.pdf")
;
        }
        [Authorize(Roles = "Admin")]
        [HttpGet]
        [Route("bulk-dispatch")]
        public async Task<IActionResult> BulkDispatch()
        {
            var packages = await _context!.Packages
                .Include(p => p.SenderBranch)
                .Include(p => p.Branch)
                .OrderByDescending(p => p.Id)
                .ToListAsync();
            var availableTransports = await _context.Transports
                .Where(t => t.IsActive)
                .OrderBy(t => t.TransportNumber)
                .ToListAsync();
            var vm = new BulkAssignPackagesViewModel
            {
                Packages = packages.Select(p => new PackageSelectionItemVM
                {
                    PackageId = p.Id,
                    TrackingNumber = p.TrackingNumber ?? "",
                    SenderCity = p.SenderBranch?.Name ?? "",
                    RecipientCity = p.Branch?.Name ?? "",
                    Weight = p.Weight,
                    Status = p.Status
                }).ToList(),
                TransportOptions = availableTransports.Select(t => new SelectListItem
                {
                    Value = t.Id.ToString(),
                    Text = $"{t.Type} ({t.TransportNumber})"
                }).ToList(),
                StatusOptions = Enum.GetValues(typeof(Status))
                    .Cast<Status>()
                    .Select(s => new SelectListItem
                    {
                        Value = s.ToString(),
                        Text = s.ToString()
                    }).ToList()
            };
            return View(vm);
        }
        [HttpPost]
        [Route("bulk-dispatch")]
        [Authorize(Roles = "Admin")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> BulkDispatch(BulkAssignPackagesViewModel vm)
        {
            var selectedPackageIds = vm.Packages
                .Where(p => p.IsSelected)
                .Select(p => p.PackageId)
                .ToList();
            if (vm.SelectedStatus == null)
            {
                ModelState.AddModelError(nameof(vm.SelectedStatus), "Choose status.");
            }
            if (!selectedPackageIds.Any())
            {
                ModelState.AddModelError(string.Empty, "Choose at least one package.");
            }
            Transport? selectedTransport = null;
            if (vm.SelectedStatus == Status.InTransit)
            {
                if (vm.SelectedTransportId == null)
                {
                    ModelState.AddModelError(nameof(vm.SelectedTransportId), "Choose transport for status InTransit.");
                }
                else
                {
                    selectedTransport = await _context!.Transports
                        .FirstOrDefaultAsync(t => t.Id == vm.SelectedTransportId && t.IsActive);
                    if (selectedTransport == null)
                    {
                        ModelState.AddModelError(nameof(vm.SelectedTransportId), "Choosed transport unavailiable.");
```

```
                }
            }
        }
        if (!ModelState.IsValid)
        {
            var packages = await _context!.Packages
                .Include(p => p.SenderBranch)
                .Include(p => p.Branch)
                .OrderByDescending(p => p.Id)
                .ToListAsync();
            var availableTransports = await
_context.Transports
                .Where(t => t.IsActive)
                .OrderBy(t => t.TransportNumber)
                .ToListAsync();
            vm.Packages = packages.Select(p => new
PackageSelectionItemVM
            {
                PackageId = p.Id,
                IsSelected = selectedPackageIds.Contains(p.Id),
                TrackingNumber = p.TrackingNumber ?? "",
                SenderCity = p.SenderBranch?.Name ?? "",
                RecipientCity = p.Branch?.Name ?? "",
                Weight = p.Weight,
                Status = p.Status
            }).ToList();
            vm.TransportOptions =
availableTransports.Select(t => new SelectListItem
            {
                Value = t.Id.ToString(),
                Text = $"{t.Type} ({t.TransportNumber})"
            }).ToList();
            vm.StatusOptions =
Enum.GetValues(typeof(Status))
                .Cast<Status>()
                .Select(s => new SelectListItem
                {
                    Value = s.ToString(),
                    Text = s.ToString()
                }).ToList();
            return View(vm);
        }
        var packagesToUpdate = await _context!.Packages
            .Where(p => selectedPackageIds.Contains(p.Id))
            .ToListAsync();
        foreach (var package in packagesToUpdate)
        {
            package.Status = vm.SelectedStatus!.Value;
            if (vm.SelectedStatus == Status.InTransit)
            {
                package.TransportId = vm.SelectedTransportId;
            }
            else
            {
                package.TransportId = null;
            }
        }
        await _context.SaveChangesAsync();
        TempData["SuccessMessage"] = $"Status
successfully updated for {packagesToUpdate.Count}
packages on {vm.SelectedStatus}.";
        return RedirectToAction(nameof(BulkDispatch));
    }
  }
}
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
using PackageManagementSystem.Data;
using PackageManagementSystem.Models;
using PackageManagementSystem.ViewModels;
namespace PackageManagementSystem.Controllers
{
  [Authorize]
  [Route("profile")]
  public class ProfileController : Controller
  {
    private readonly UserManager<ApplicationUser>
_userManager;
    private readonly SignInManager<ApplicationUser>
_signInManager;
    private readonly
PackageManagementSystemDbContext _context;
    public ProfileController(
      UserManager<ApplicationUser> userManager,
      SignInManager<ApplicationUser> signInManager,
      PackageManagementSystemDbContext context)
    {
      _userManager = userManager;
      _signInManager = signInManager;
      _context = context;
    }
    [HttpGet("")]
    public async Task<IActionResult> Index()
    {
      var user = await _userManager.GetUserAsync(User);
      if (user == null)
        return NotFound();
      var branches = await _context.Branches
            .AsNoTracking()
            .Where(x => x.IsActive)
            .OrderBy(x => x.City)
            .ThenBy(x => x.Name)
            .ToListAsync();
      var vm = new ProfileViewModel
      {
        UserId = user.Id,
        FullName = user.FullName ?? "",
        Email = user.Email ?? "",
        BranchId = user.BranchId,
        AvailableBranches = branches.Select(x => new
SelectListItem
        {
          Value = x.Id.ToString(),
          Text = $"{x.City} - {x.Name} ({x.Address})"
        }).ToList()
      };
      return View(vm);
    }
    [HttpPost("")]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult>
Index(ProfileViewModel vm)
    {
      var user = await _userManager.GetUserAsync(User);
      if (user == null)
        return NotFound();
      var branches = await _context.Branches
          .AsNoTracking()
          .Where(x => x.IsActive)
          .OrderBy(x => x.City)
          .ThenBy(x => x.Name)
          .ToListAsync();
      vm.AvailableBranches = branches.Select(x => new
SelectListItem
      {
        Value = x.Id.ToString(),
        Text = $"{x.City} - {x.Name} ({x.Address})"
      }).ToList();
      if (!ModelState.IsValid)
        return View(vm);
      user.FullName = vm.FullName;
      user.BranchId = vm.BranchId;
      if (user.Email != vm.Email)
      {
        var setEmailResult = await
_userManager.SetEmailAsync(user, vm.Email);
        if (!setEmailResult.Succeeded)
        {
          foreach (var error in setEmailResult.Errors)
          {
            ModelState.AddModelError(string.Empty,
error.Description);
          }
          return View(vm);
        }
```

```
            user.UserName = vm.Email;
            var setUserNameResult = await
_userManager.SetUserNameAsync(user, vm.Email);
            if (!setUserNameResult.Succeeded)
            {
               foreach (var error in setUserNameResult.Errors)
               {
                  ModelState.AddModelError(string.Empty,
error.Description);
               }
               return View(vm);
            }
         }
         var updateResult = await
_userManager.UpdateAsync(user);
         if (!updateResult.Succeeded)
         {
            foreach (var error in updateResult.Errors)
            {
               ModelState.AddModelError(string.Empty,
error.Description);
            }
            return View(vm);
         }
         if
(!string.IsNullOrWhiteSpace(vm.CurrentPassword) ||
            !string.IsNullOrWhiteSpace(vm.NewPassword) ||
            !string.IsNullOrWhiteSpace(vm.ConfirmNewPass
word))
         {
            if
(string.IsNullOrWhiteSpace(vm.CurrentPassword))
            {
               ModelState.AddModelError(nameof(vm.Curre
ntPassword), "Enter current password.");
               return View(vm);
            }
            if
(string.IsNullOrWhiteSpace(vm.NewPassword))
            {
               ModelState.AddModelError(nameof(vm.NewP
assword), "Enter new password.");
               return View(vm);
            }
            var changePasswordResult = await
_userManager.ChangePasswordAsync(
               user,
               vm.CurrentPassword,
               vm.NewPassword);
            if (!changePasswordResult.Succeeded)
            {
               foreach (var error in
changePasswordResult.Errors)
               {
                  ModelState.AddModelError(string.Empty,
error.Description);
               }
               return View(vm);
            }
         }
         await _signInManager.RefreshSignInAsync(user);
         TempData["SuccessMessage"] = "Profile updated
successfully.";
         return RedirectToAction(nameof(Index));
      }
   }
}
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using PackageManagementSystem.Data;
using PackageManagementSystem.ViewModels;
namespace PackageManagementSystem.Controllers
{
   [Route("pages")]
   public class PageController : Controller
   {
      private readonly
PackageManagementSystemDbContext _context;
      public
PageController(PackageManagementSystemDbContext
context)
      {
         _context = context;
      }
      [HttpGet("about")]
      public async Task<IActionResult> About() => await
GetPage("about");
      [HttpGet("contact")]
      public async Task<IActionResult> Contact() => await
GetPage("contact");
      [HttpGet("privacy-policy")]
      public async Task<IActionResult> PrivacyPolicy() =>
await GetPage("privacy");
      private async Task<IActionResult> GetPage(string
slug)
      {
         var page = await _context.Pages!
            .AsNoTracking()
            .FirstOrDefaultAsync(x => x.Slug == slug);
         if (page == null)
            return NotFound();
         var vm = new PageVM
         {
            Id = page.Id,
            Title = page.Title,
            Slug = page.Slug,
            ShortDescription = page.ShortDescription,
            Description = page.Description,
            PhotoUrl = page.PhotoUrl
         };
         return View("Page", vm);
      }
   }
}
using QuestPDF.Fluent;
using QuestPDF.Helpers;
using QuestPDF.Infrastructure;
using PackageManagementSystem.Models;
using QRCoder;
using System.IO;
namespace PackageManagementSystem.Services.Pdf
{
   public class InvoiceDocument : IDocument
   {
      private readonly Package _package;
      public InvoiceDocument(Package package)
      {
         _package = package;
      }
      public DocumentMetadata GetMetadata() =>
DocumentMetadata.Default;
      public void Compose(IDocumentContainer container)
      {
         container.Page(page =>
         {
            page.Margin(30);
            page.Header().Row(row =>
            {
               row.RelativeItem().MaxWidth(150).Image("w
wwroot/images/logo.png", ImageScaling.FitWidth);
               row.RelativeItem().AlignRight().Text("Invoice
")
                  .FontSize(20).SemiBold().WrapAnywhere();
            });
            page.Content().PaddingVertical(20).Column(colu
mn =>
            {
               column.Spacing(15);
               column.Item().Table(table =>
               {
                  table.ColumnsDefinition(columns =>
                  {
                     columns.RelativeColumn();
                     columns.RelativeColumn();
                  });
                  var rows = new (string label, string value)[]
                  {
```

```
                    ("Tracking number", _package.TrackingNumber),
                    ("Sender", _package.Sender!.FullName),
                    ("Reciver", _package.ReceiverName),
                    ("Address", _package.ReceiverAddress),
                    ("WEight", $"{_package.Weight} kg"),
                    ("Status", _package.Status.ToString()),
                    ("Date of creation", _package.CreatedAt.ToString("dd.MM.yyyy"))
                };
                var cellStyle = new Action<IContainer>(c => c.Padding(5).Border(1).BorderColor(Colors.Grey.Lighten2));
                foreach (var row in rows)
                {
                    table.Cell()
                        .Padding(5)
                        .Border(1)
                        .BorderColor(Colors.Grey.Lighten2)
                        .Text(row.label);
                    table.Cell()
                        .Padding(5)
                        .Border(1)
                        .BorderColor(Colors.Grey.Lighten2)
                        .Text(row.value);
                }
            });
            column.Item().PaddingTop(20).AlignCenter().Element(container => GenerateQrCode(container) );
        });


        page.Footer()
            .AlignCenter()
            .Text(text =>
            {
                text.Span("Package Management System • ");
                text.Span(DateTime.Now.ToString("dd.MM.yyyy HH:mm"));
            });
    });
}
private void GenerateQrCode(IContainer container)
{
    using var qrGenerator = new QRCodeGenerator();
    using var qrCodeData = qrGenerator.CreateQrCode(_package.TrackingNumber, QRCodeGenerator.ECCLevel.Q);
    var qrCode = new PngByteQRCode(qrCodeData);
    var qrBytes = qrCode.GetGraphic(20);
    container.Width(150)
     .Height(150)
     .Image(qrBytes, ImageScaling.FitArea);
}
}
}
using PackageManagementSystem.Models;
namespace PackageManagementSystem.Services.Pdf
{
  public interface IPdfService
  {
    byte[] GenerateInvoice(Package package);
  }
}
using QuestPDF.Fluent;
using PackageManagementSystem.Models;
namespace PackageManagementSystem.Services.Pdf
{
  public class PdfService : IPdfService
  {
    public byte[] GenerateInvoice(Package package) {
      var document = new InvoiceDocument(package);
      return document.GeneratePdf();
    }
  }
}
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.SignalR;
using Microsoft.EntityFrameworkCore;
using PackageManagementSystem.Data;
using PackageManagementSystem.Models;
namespace PackageManagementSystem.Services
{
  [Authorize]
  public class ChatHub : Hub
  {
    private readonly PackageManagementSystemDbContext _context;
    private readonly UserManager<ApplicationUser> _userManager;
    public ChatHub(
      PackageManagementSystemDbContext context,
      UserManager<ApplicationUser> userManager)
    {
      _context = context;
      _userManager = userManager;
    }
    public async Task JoinShipmentChat(int shipmentId)
    {
      var user = await _userManager.GetUserAsync(Context.User!);
      var shipment = await _context.Shipments
        .Include(s => s.Package)
        .FirstOrDefaultAsync(s => s.Id == shipmentId);
      if (shipment == null)
        throw new HubException("Shipment not found");
      if (user!.Id != shipment.Package.ReceiverId &&
        user.Id != shipment.DeliveryAgentId)
        throw new HubException("Access denied");
      await Groups.AddToGroupAsync(Context.ConnectionId, $"shipment-{shipmentId}");
    }
    public async Task SendMessage(int shipmentId, string text)
    {
      if (string.IsNullOrWhiteSpace(text))
        return;
      var user = await _userManager.GetUserAsync(Context.User!);
      var shipment = await _context.Shipments
        .Include(s => s.Chat)
        .Include(s => s.Package)
        .FirstOrDefaultAsync(s => s.Id == shipmentId);
      if (shipment == null)
        throw new HubException("Shipment not found");
      if (user!.Id != shipment.Package.ReceiverId &&
        user.Id != shipment.DeliveryAgentId)
        throw new HubException("Access denied");
      if (shipment.Chat == null)
        throw new HubException("Chat not found");
      var message = new Message
      {
        ChatId = shipment.Chat.Id,
        SenderId = user.Id,
        Text = text.Trim(),
        SentAt = DateTime.UtcNow
      };
      _context.Messages.Add(message);
      await _context.SaveChangesAsync();
      var senderName = !string.IsNullOrWhiteSpace(user.FullName)
        ? user.FullName
        : user.UserName ?? "User";
      await Clients
        .Group($"shipment-{shipmentId}")
        .SendAsync("ReceiveMessage", new
        {
          senderId = user.Id,
          senderName = senderName,
          text = message.Text,
          sentAt = message.SentAt
        });
    }
  }
}
using PackageManagementSystem.Models;
```

```
namespace PackageManagementSystem.Services
{
   public class DeliveryPricingService : IDeliveryPricingService
   {
      public decimal CalculatePrice(double weight, DeliveryType deliveryType)
      {
         if (weight <= 0)
            throw new ArgumentException("Weight should be above zero");
         return deliveryType switch
         {
            DeliveryType.Pickup => CalculateBranchToBranch(weight),
            DeliveryType.Courier => CalculateCourier(weight),
            _ => throw new ArgumentOutOfRangeException(nameof(deliveryType), "Unknown delivery type.")
         };
      }
      private decimal CalculateBranchToBranch(double weight)
      {
         if (weight <= 2) return 50m;
         if (weight <= 5) return 70m;
         if (weight <= 10) return 100m;
         if (weight <= 20) return 140m;


         return 200m;
      }
      private decimal CalculateCourier(double weight)
      {
         if (weight <= 2) return 90m;
         if (weight <= 5) return 150m;
         if (weight <= 10) return 200m;
         if (weight <= 20) return 250m;
         return 280m;
      }
   }
}
using MailKit.Net.Smtp;
using MailKit.Security;
using Microsoft.AspNetCore.Identity.UI.Services;
using MimeKit;
using PackageManagementSystem.Models;
namespace PackageManagementSystem.Services
{
   public class EmailSender : IEmailSender
   {
      private readonly EmailConfig? _config;
      public EmailSender(EmailConfig? config)
      {
         _config = config;
      }
      public async Task SendEmailAsync(string to, string subject, string message)
      {
         var email = new MimeMessage();
         email.From.Add(MailboxAddress.Parse(_config?.From));
         email.To.Add(MailboxAddress.Parse(to));
         email.Subject = subject;
         email.Body = new TextPart(MimeKit.Text.TextFormat.Html)
         {
            Text = message
         };
         using var smtp = new SmtpClient();
         await smtp.ConnectAsync(_config?.SmtpServer, _config!.Port, SecureSocketOptions.Auto);
         await smtp.AuthenticateAsync(_config.UserName, _config.Password);
         await smtp.SendAsync(email);
         await smtp.DisconnectAsync(true);
      }
   }
}
namespace PackageManagementSystem.Services
{
   public interface IEmailSender
   {
      Task SendEmailAsync(string to, string subject, string message);
   }
}
using PackageManagementSystem.Models;
namespace PackageManagementSystem.Services
{
   public interface IRewardService
   {
      Task<RewardApplyResult> ApplyRewardAsync(
         string userId,
         DeliveryType deliveryType,
         decimal basePrice
       );
   }
}
using Microsoft.EntityFrameworkCore;
using PackageManagementSystem.Data;
using PackageManagementSystem.Models;
namespace PackageManagementSystem.Services
{
   public class RewardService : IRewardService
   {
      private readonly PackageManagementSystemDbContext _context;
      public RewardService(PackageManagementSystemDbContext context)
      {
         _context = context;
      }
      public async Task<RewardApplyResult> ApplyRewardAsync(string userId, DeliveryType deliveryType, decimal basePrice)
      {
         var result = new RewardApplyResult
         {
            FinalPrice = basePrice,
            RewardApplied = false,
            AppliedRewardType = null
         };
         var rewards = await _context.UserRewards
            .Where(r => r.UserId == userId && !r.IsUsed)
            .OrderBy(r => r.CreatedAt)
            .ToListAsync();
         foreach (var reward in rewards)
         {
            switch (reward.RewardType)
            {
               case RewardType.FreeBranchDelivery:
                  if (deliveryType == DeliveryType.Pickup)
                  {
                     reward.IsUsed = true;
                     reward.UsedAt = DateTime.UtcNow;
                     await _context.SaveChangesAsync();
                     result.FinalPrice = 0m;
                     result.RewardApplied = true;
                     result.AppliedRewardType = RewardType.FreeBranchDelivery;
                     return result;
                  }
                  break;
               case RewardType.FreeCourierDelivery:
                  if (deliveryType == DeliveryType.Courier)
                  {
                     reward.IsUsed = true;
                     reward.UsedAt = DateTime.UtcNow;
                     await _context.SaveChangesAsync();
                     result.FinalPrice = 0m;
                     result.RewardApplied = true;
                     result.AppliedRewardType = RewardType.FreeCourierDelivery;
                     return result;
                  }
```

```
                break;
            case RewardType.DiscountPercent:
                var discounted = basePrice - (basePrice *
reward.Value / 100m);
                if (discounted < 0)
                    discounted = 0;
                reward.IsUsed = true;
                reward.UsedAt = DateTime.UtcNow;
                await _context.SaveChangesAsync();
                result.FinalPrice = discounted;
                result.RewardApplied = true;
                result.AppliedRewardType =
RewardType.DiscountPercent;
                return result;
        }
            }
            return result;
        }
    }
}
using PackageManagementSystem.Models;
namespace PackageManagementSystem.Services
{
    public class RewardApplyResult
    {
        public decimal FinalPrice { get; set; }
        public RewardType? AppliedRewardType { get; set; }
        public bool RewardApplied { get; set; }
    }
}
```

ДОДАТОК В. СХЕМА БАЗИ ДАНИХ ДОДАТКУ

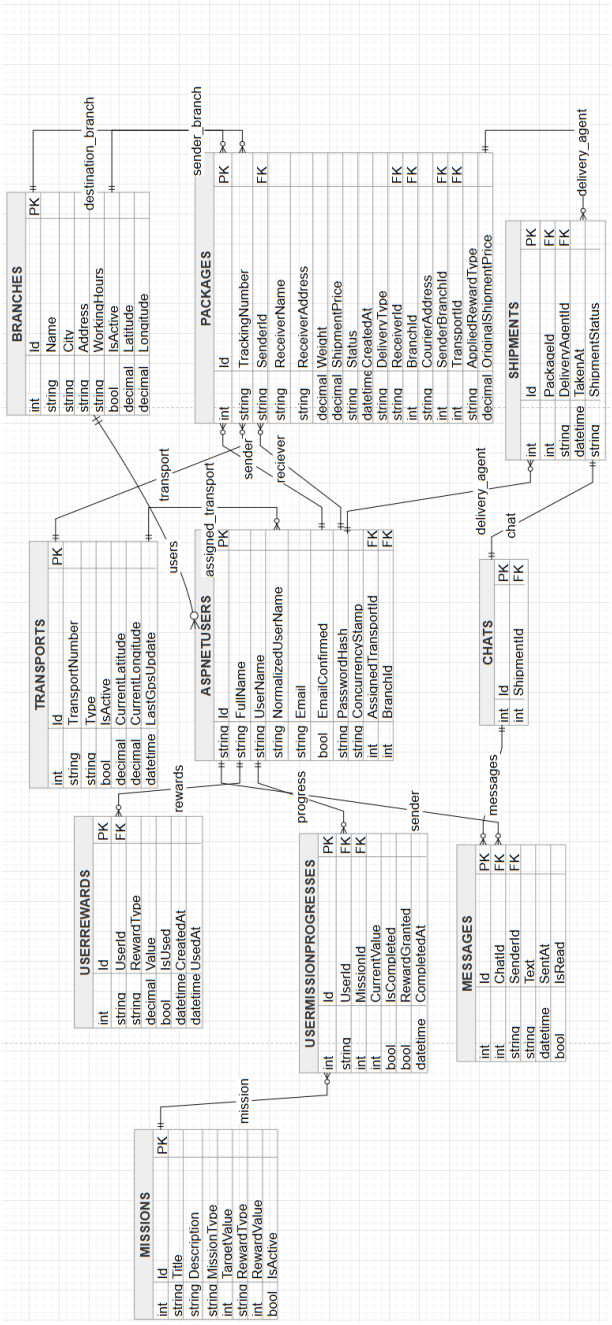


Рис. В.1 Структура бази даних