НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«До захисту допущено»
Завідувач кафедри
_____ Євгенія СУЛЕМА
«___» _____ 2025 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем»

спеціальності 121 Інженерія програмного забезпечення

на тему: «Вебзастосунок для обліку та реєстрації поштових відділень»

Виконав:
студент IV курсу, групи КП-11
Лунев Артем Вадимович _____

Керівник:
доцент кафедри ПЗКС, к.т.н., доцент,
Новак Дмитро Сергійович _____

Консультант з нормоконтролю:
доцент кафедри ПЗКС, к.т.н., доцент,
Онай Микола Володимирович _____

Рецензент:
доцент кафедри інформаційних технологій, штучного інтелекту та кібербезпеки Національного університету харчових технологій, к. т. н., доцент
Мошенський Андрій Олександрович _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2025 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет прикладної математики
Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Євгенія СУЛЕМА
«___»_____2024 р.

ЗАВДАННЯ

на дипломний проєкт студенту

Луневу Артему Вадимовичу

1. Тема проєкту «Вебзастосунок для обліку та реєстрації поштових відділень», керівник проєкту Новак Дмитро Сергійович, к.т.н., доцент, затверджені наказом по університету №1808-С від «29» травня 2025 р.

2. Термін подання студентом проєкту «13» червня 2025 р.

3. Вихідні дані до проєкту: див. Технічне завдання.

4. Зміст пояснювальної записки:
   - аналіз предметної області та існуючих рішень;
   - аналіз мов програмування та технологій розроблення вебсайтів;
   - розроблення вебзастосунку для обліку та реєстрації поштових відділень;
   - аналіз розроблених алгоритмів та підпрограм;
   - аналіз розробленого вебзастосунку.

5. Перелік обов'язкового графічного матеріалу:
   - структура бази даних (креслення);
   - алгоритм реєстрації відправлення (креслення);
   - архітектура вебзастосунку (плакат);
   - дерево проблем потенційних користувачів системи (плакат).

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Онай М.В., доцент		

7. Дата видачі завдання «31» жовтня 2024 р.

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	Вивчення літератури за тематикою проєкту	13.11.2024	
2.	Розроблення та узгодження технічного завдання	22.11.2024	
3.	Розроблення структури вебзастосунку	15.12.2024	
4.	Підготовка матеріалів першого розділу дипломного проєкту	28.12.2024	
5.	Розроблення дизайну сторінок та графічних елементів	01.02.2025	
6.	Підготовка матеріалів другого розділу дипломного проєкту	19.02.2025	
7.	Програмна реалізація вебзастосунку	11.03.2025	
8.	Тестування вебзастосунку	18.03.2025	
9.	Підготовка матеріалів третього розділу дипломного проєкту	27.03.2025	
10.	Підготовка матеріалів четвертого розділу дипломного проєкту	15.04.2025	
11.	Підготовка графічної частини дипломного проєкту	22.04.2025	
12.	Оформлення документації дипломного проєкту	27.05.2025	

Студент Артем ЛУНЕВ

Керівник проєкту Дмитро НОВАК

АНОТАЦІЯ

Цей дипломний проєкт присвячений розробці застосунку для обліку та реєстрації поштових відділень. Застосунок містить як статичні, так і динамічні компоненти, що забезпечують інформування користувачів та надання інтерактивних сервісів у режимі онлайн. Основною функціональністю є авторизація користувачів за трьома ролями: клієнт, оператор відділення та адміністратор.

Для клієнтів реалізовано карту відправлень із можливістю перегляду статусу посилок у реальному часі. Оператори відділень мають доступ до інтерфейсу керування відправленнями, а адміністратори можуть реєструвати нові відділення та редагувати дані про наявні.

Окрему увагу приділено інтерактивній мапі поштових відділень із динамічним оновленням локацій кур'єрів, що дозволяє клієнтам відстежувати переміщення їхніх посилок. Також впроваджено систему сповіщення користувачів про зміни статусів відправлень за допомогою електронних листів на email.

У межах проєкту розроблено архітектуру застосунку, алгоритми авторизації та оновлення даних, а також реалізовано зручний та інтуїтивний інтерфейс користувача.

ABSTRACT

This thesis project is dedicated to the development of the PostTrack application for the management and registration of postal branches. The application includes both static and dynamic components that provide users with information and access to interactive online services. The core functionality includes user authentication with three roles: client, branch operator, and administrator.

Clients have access to a shipment map that allows them to track the status of their parcels in real time. Branch operators can manage shipments through a dedicated interface, while administrators are able to register new branches and update existing branch data.

A key feature of the application is an interactive map of postal branches with dynamic courier location updates, enabling clients to follow the movement of their parcels. Additionally, a notification system has been implemented to inform users about shipment status changes via email alerts.

As part of this project, the system architecture was designed, user authentication and data synchronization algorithms were developed, and a user-friendly graphical interface was implemented.

ДП.045440-01-90 Вебзастосунок для обліку та реєстрації поштових відділень.
Відомість проєкту

Позначення	Найменування	Кіл-ть	Примітка
	Документація проєкту		
ДП.045440-02-91	Вебзастосунок для	5	
	обліку та реєстрації		
	поштових відділень.		
	Технічне завдання		
ДП.045440-03-81	Вебзастосунок для	70	
	обліку та реєстрації		
	поштових відділень.		
	Пояснювальна записка		
ДП.045440-04-51	Вебзастосунок для	4	
	обліку та реєстрації		
	поштових відділень.		
	Програма та методика		
	тестування		
ДП.045440-05-34	Вебзастосунок для	10	
	обліку та реєстрації		
	поштових відділень.		
	Керівництво користувача		
ДП.045440-06-99	Вебзастосунок для	1	
	обліку та реєстрації		
	поштових відділень.		
	Реєстрація відправлення.		
	Схема алгоритму		

Позначення	Найменування	Кіл-ть	Примітка
ДП.045440-07-99	Вебзастосунок для	1	
	обліку та реєстрації		
	поштових відділень.		
	Структура бази даних.		
	ER діаграма		
ДП.045440-08-98	Вебзастосунок для	1	
	обліку та реєстрації		
	поштових відділень.		
	Компакт-диск		

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Завідувач кафедри

_____ Євгенія СУЛЕМА

"___" _____ 2024 р.

ВЕБЗАСТОСУНОК ДЛЯ ОБЛІКУ ТА РЕЄСТРАЦІЇ ПОШТОВИХ ВІДДІЛЕНЬ

Технічне завдання

ДП.045440-02-91

"ПОГОДЖЕНО"

Керівник проєкту:

_____ Дмитро НОВАК

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Артем ЛУНЕВ

2024

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: вебзастосунок для обліку та реєстрації поштових відділень.

Галузь застосування: інформаційні технології.

2. ПІДСТАВА ДЛЯ РОЗРОБЛЕННЯ

Підставою для розроблення є завдання на дипломне проєктування, затверджене кафедрою програмного забезпечення комп'ютерних систем Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (КПІ ім. Ігоря Сікорського).

3. ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для використання як інформаційно-сервісна система з обліку поштових відділень та відстеження пересилок. Система забезпечує доступ користувачів до актуальної інформації про місцезнаходження відправлень, стан доставки та надає сервіси взаємодії між клієнтами, операторами поштових відділень та адміністраторами.

4. ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

Вебзастосунок повинен забезпечувати такі основні функції:

1) можливість авторизації користувачів з поділом на ролі: клієнт, оператор відділення, адміністратор;
2) можливість оновлення геолокації кур'єрів у режимі реального часу;
3) підтримка функціоналу перегляду та відстеження відправлень клієнтами;

4) можливість керування відправленнями з боку операторів (створення, оновлення статусів, прив’язка до відділень і кур’єрів);
5) реєстрація та адміністрування відділень адміністраторами системи.

Додаткові вимоги:

1) наявність інтерактивної мапи відділень та посилок;
2) зберігання паролів користувачів у захешованому вигляді;
3) використання захищеного протоколу зв’язку (HTTPS);
4) можливість додавання нових відділень та розширення кількості користувачів без суттєвої зміни архітектури.

5. ВИМОГИ ДО ПРОЄКТНОЇ ДОКУМЕНТАЦІЇ

У процесі виконання проєкту повинна бути розроблена наступна документація:

1) пояснювальна записка;
2) програма та методика тестування;
3) керівництво користувача;
4) креслення:
   – «Реєстрація відправлення. Схема алгоритму»;
   – «Структура бази даних».

6. ЕТАПИ ПРОЄКТУВАННЯ

Вивчення літератури за тематикою роботи..... 16.11.2024
Розроблення та узгодження технічного завдання..... 27.11.2024
Розроблення структури вебзастосунку 15.12.2024
Розроблення дизайну сторінок та графічних елементів..... 05.02.2025
Програмна реалізація вебзастосунку 15.03.2025
Тестування вебзастосунку..... 08.04.2025
Підготовка матеріалів текстової частини проєкту..... 29.04.2025
Підготовка матеріалів графічної частини проєкту 15.05.2025

Оформлення технічної документації проєкту..... 28.05.2025

7. ПОРЯДОК ТЕСТУВАННЯ РОЗРОБКИ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Факультет прикладної математики

Кафедра програмного забезпечення комп’ютерних систем

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Євгенія СУЛЕМА

“___” _____ 2025 р.

ВЕБЗАСТОСУНОК ДЛЯ ОБЛІКУ ТА РЕЄСТРАЦІЇ ПОШТОВИХ ВІДДІЛЕНЬ

Пояснювальна записка

ДП.045440-03-81

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Дмитро НОВАК

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Артем ЛУНЕВ

2025

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

API – Application Programming Interface (інтерфейс прикладного програмування).

CRUD – Create, Read, Update, Delete (базові операції над даними).

DB – Database (база даних).

DOM – Document Object Model (об'єктна модель документа).

GPS – Global Positioning System (глобальна система позиціювання).

HTTP – HyperText Transfer Protocol (протокол передавання гіпертексту).

HTTPS – HyperText Transfer Protocol Secure (захищений протокол HTTP).

ID – Identifier (ідентифікатор).

JSON – JavaScript Object Notation (формат обміну даними).

JWT – JSON Web Token (формат маркерів автентифікації).

MVC – Model-View-Controller (шаблон архітектури: модель-представлення-контролер).

OTP – One-Time Password (одноразовий пароль).

REST – Representational State Transfer (архітектурний стиль взаємодії з API).

RTK – Redux Toolkit (бібліотека для управління станом у React).

RTK Query – Інструмент для запитів до API у Redux Toolkit.

SMTP – Simple Mail Transfer Protocol (простий протокол пересилання пошти).

UI – User Interface (інтерфейс користувача).

UX – User Experience (досвід користувача).

UUID – Universally Unique Identifier (універсальний унікальний ідентифікатор).

UXD – User Experience Design (проєктування користувацького досвіду).

XSS – Cross-site Scripting (міжсайтове скриптування – тип атаки на вебзастосунок).

ВСТУП

У сучасному світі логістика та поштові сервіси відіграють ключову роль у забезпеченні безперебійного функціонування торгівлі, комунікації та обміну інформацією. З кожним роком кількість поштових відправлень зростає, що зумовлено активним розвитком електронної комерції, глобалізацією бізнес-процесів та потребами населення у швидкому й надійному обміні товарами та документацією.

Разом із цим зростає потреба в автоматизації процесів, пов’язаних з обліком поштових відділень, управлінням їхньою структурою та контролем відправлень. Традиційні методи ведення обліку, засновані на ручному введенні даних або застарілих програмних рішеннях, часто призводять до помилок, затримок і зниження ефективності роботи поштових служб. Особливо актуальною є проблема забезпечення швидкого доступу до інформації про переміщення відправлень, стан послуг та ефективність роботи окремих відділень.

Відтак, виникає потреба у створенні сучасної системи обліку поштових відділень і відправлень, яка б відповідала вимогам точності, доступності та зручності використання. Така система повинна спрощувати процеси управління, покращувати аналітику, зменшувати ризики втрати або затримки посилок, а також підвищувати загальну якість обслуговування клієнтів.

Даний дипломний проєкт присвячений розробці рішення, що дозволяє ефективно організувати облік поштової інфраструктури та забезпечити прозору і зручну роботу з інформацією про відправлення, орієнтуючись на актуальні потреби галузі.

1. ОГЛЯД ІСНУЮЧИХ ВЕБСЕРВІСІВ ДЛЯ ОБЛІКУ ПОШТОВИХ ВІДДІЛЕНЬ

1.1. Визначення чітких критеріїв аналізу

Перед тим як розпочати розробку власної інформаційної системи обліку поштових відправлень, доцільно здійснити аналіз існуючих програмних рішень, що вже представлені на ринку. Це дозволяє не лише вивчити наявний досвід та уникнути поширених помилок, але й виявити функціональні обмеження чи переваги конкурентів.

Щоб оцінити доцільність впровадження або адаптації наявних рішень, важливо визначити чіткі критерії аналізу, які базуються на практичних потребах майбутньої системи та проблемах, описаних у попередніх розділах.

Основними критеріями оцінювання таких сервісів є:

1. Функціональне покриття: чи підтримується відстеження статусів відправлень, управління доставленнями, контроль роботи кур’єрів, логістичні маршрути, повідомлення клієнтів, тощо.
2. Можливість адаптації або індивідуалізації: наскільки легко підлаштувати сервіс під індивідуальні потреби організації, змінити структуру даних, додати нові модулі чи інтеграції.
3. Інтерфейс та зручність використання: наскільки зрозумілим є вебінтерфейс для адміністратора, оператора, кур’єра; наявність мобільних додатків; багатомовність тощо.
4. Вартість використання: чи є сервіс платним; які моделі ліцензування пропонуються (щомісячна підписка, ціна за користувача, тощо); наявність безкоштовного плану.
5. Інтеграція з іншими системами: чи підтримуються API, чи є можливість під'єднати систему до баз даних, поштових сервісів, ERP або CRM.

6. Надійність та безпека: які механізми захисту даних передбачені, чи відповідає сервіс сучасним стандартам безпеки (HTTPS, авторизація, резервне копіювання).
7. Наявність технічної підтримки та документації: чи доступна підтримка користувачам, наскільки детально описані функції сервісу.

З урахуванням цих критеріїв буде проведено аналіз таких популярних рішень як Onfleet, Track-POD та Detrack, що позиціонуються як комплексні логістичні платформи для управління доставленням. Це дозволить порівняти їхню функціональність з вимогами до системи, яку планується реалізувати в рамках дипломного проєкту.

1.2. Onfleet

Одним із відомих сучасних рішень у сфері логістики та доставлення є вебсервіс Onfleet. Це комерційна платформа, яка надає комплексні інструменти для управління кур'єрською доставленням в режимі реального часу [1]. Сервіс активно використовується компаніями у сфері електронної комерції, охорони здоров'я, роздрібної торгівлі та продуктового постачання.

Серед основних можливостей Onfleet – відстеження місцеперебування кур'єрів, планування маршрутів, управління завданнями доставлення, облік часу виконання замовлень, а також зворотний зв'язок із клієнтами. Система дозволяє централізовано контролювати процеси доставлення, оптимізовувати навантаження між відділеннями та підвищувати ефективність логістичних операцій.

Onfleet більше орієнтований саме на служби доставлення й кур'єрські сервіси, а не на поштові відділення в класичному розумінні. Він не вирішує повною мірою завдання обліку поштових відділень, структури поштової мережі, або внутрішнього контролю поштових відправлень між відділеннями. Також використання Onfleet пов'язане з щомісячною абонплатою, що робить його недоступним для деяких державних чи

невеликих приватних організацій, які потребують бюджетного або кастомізованого рішення.

У контексті даного дипломного проєкту, Onfleet може бути розглянутий як приклад потужного, але спеціалізованого продукту, який не закриває всіх потреб, пов’язаних із загальним обліком поштової інфраструктури. Саме тому виникає потреба у розробці окремої системи, яка буде адаптована під специфіку поштових служб та враховуватиме як потреби користувачів, так і адміністративний контроль за відправленнями та їх обігом між відділеннями.

1.3. Track-POD

Track-POD – це сучасна платформа для управління доставленням, яка надає рішення для автоматизації логістичних процесів, включаючи планування маршрутів, електронне підтвердження доставлення (Proof of Delivery), моніторинг транспортних засобів та звітність у реальному часі [2]. Сервіс орієнтований переважно на компанії, які здійснюють доставлення товарів клієнтам, і широко застосовується у сфері роздрібної торгівлі, дистрибуції та кур’єрських послуг.

Track-POD надає можливість створювати маршрути з урахуванням геолокації, відстежувати статус виконання доставлення, фіксувати підписи клієнтів, вести облік вантажу та генерувати звіти. Для кур’єрів доступний мобільний додаток, через який вони отримують завдання, відмічають виконання та взаємодіють із системою.

Хоча сервіс має розширений функціонал у сфері кур’єрської логістики, він не призначений для обліку внутрішніх операцій поштових відділень, збереження інформації про структуру поштової мережі, чи детального відслідковування поштових відправлень між відділеннями. Track-POD більше орієнтований на останню милю доставлення і потреби приватних компаній, що не повністю відповідає специфіці державних або змішаних поштових структур.

У контексті даного дипломного проєкту, Track-POD можна вважати прикладом ефективної, але вузькоспеціалізованої системи, яка вирішує окремі логістичні завдання, не охоплюючи при цьому повний цикл обліку та адміністрування поштових відправлень. Це підкреслює потребу у створенні власного вебзастосунку, який буде пристосований саме до вимог поштових відділень, забезпечуватиме внутрішню організацію процесів, прозорий облік відправлень та гнучку адаптацію до різних рівнів користувачів – від працівників до адміністраторів.

1.4. Detrack

Detrack – це вебсервіс для управління доставленням, який дозволяє компаніям здійснювати моніторинг логістичних процесів у реальному часі [3]. Система надає інструменти для відслідковування транспортних засобів, реєстрації доказів доставлення (Proof of Delivery), автоматичної маршрутизації, взаємодії з клієнтами через повідомлення та аналітики на основі зібраних даних. Основна мета сервісу – забезпечення прозорості та ефективності останньої милі доставлення.

Користувачі Detrack отримують змогу створювати замовлення, призначати їх кур’єрам, отримувати фотографії, підписи клієнтів та підтвердження вручення. Через мобільний додаток кур’єри фіксують статуси доставлення та спілкуються з центральною системою. Також платформа підтримує сповіщення клієнтів та формування звітів.

Detrack здебільшого фокусується на зовнішній доставленні до клієнта, а не на внутрішньому обліку поштових відправлень у межах поштової мережі. Сервіс не передбачає ведення структурованої інформації про поштові відділення, реєстрації внутрішніх переміщень між ними, або рольового розподілу прав доступу для співробітників відділень різних рівнів.

Таким чином, хоча Detrack є потужним інструментом у сфері комерційної логістики, він не відповідає усім потребам, що виникають у

контексті автоматизації внутрішніх процесів поштових служб. Це підкреслює актуальність розробки власного вебзастосунку в межах дипломного проєкту, який буде орієнтований саме на задачі обліку поштових відправлень, керування поштовою інфраструктурою та впорядкування внутрішніх адміністративних процесів поштових відділень.

1.5. Результати проведеного аналізу

Нижче наведено таблицю порівняння сервісів за функціональними вимогами.

Таблиця 1.1

Порівняння існуючих програмних рішень

Сервіс / Функціональна вимога	Onfleet	Track-POD	Detrack
Облік поштових відділень	–	–	–
Внутрішнє переміщення відправлень між відділеннями	–	–	–
Призначення кур’єра та контроль виконання доставлення	+	+	+
Відстеження геолокації кур’єра в реальному часі	+	+	+
Рольовий поділ користувачів (оператор, кур’єр, клієнт)	–	–	–
Статус відправлень (прийнято, в дорозі, доставлено)	+	+	+

Для обґрунтованого вибору архітектурних рішень та функціонального складу майбутньої системи доцільно проаналізувати наявні програмні продукти, що реалізують схожі задачі. Такий аналіз дозволяє виявити сильні

сторони існуючих рішень, а також визначити функціональні прогалини, які потребують врахування при проєктуванні власної системи.

Аналіз сучасних вебсервісів – Onfleet, Track-POD та Detrack – показав, що ці системи орієнтовані переважно на управління комерційною доставленням та логістику останньої милі.

Однак, згідно з аналізом у таблиці вище, жодне з цих рішень не охоплює повною мірою ключові вимоги до обліку поштових відділень, внутрішньої передачі посилок між відділеннями, структурного адміністрування мережі або підтримки ролей, специфічних для поштової організації (адміністратор, оператор, клієнт, кур'єр). Також відсутні функції автоматичної реєстрації нових клієнтів або ведення внутрішньої історії маршрутів між підрозділами.

Таким чином, існує очевидна потреба у створенні спеціалізованої системи, яка буде пристосована саме до завдань обліку та внутрішнього управління поштовими відправленнями. Саме цим завданням і присвячено реалізацію дипломного проєкту, який враховує як реальні потреби поштових структур, так і відсутність готових рішень, які їх повноцінно покривають.

1.6. Висновки до розділу

Проведений огляд сучасних вебсервісів для керування доставленням – таких як Onfleet, Track-POD та Detrack – дозволяє зробити висновок, що всі вони пропонують широкий функціонал, орієнтований переважно на комерційну логістику та керування останньою милею доставлення до кінцевого клієнта.

Однак усі розглянуті сервіси мають спільне обмеження – відсутність фокуса на потребах саме поштових служб, зокрема на внутрішньому обліку відправлень між відділеннями, детальній структуризації поштової мережі, рольовому доступі для співробітників та фіксації переміщень усередині системи поштової інфраструктури. Крім того, більшість таких платформ є

платними та не дозволяють гнучко адаптуватися до специфіки організацій, які мають власну структуру поштових відділень та внутрішні регламенти.

Це свідчить про наявність помітного запиту на розробку спеціалізованого рішення, яке буде спрямоване не лише на контроль за доставленням, а й на повноцінне управління внутрішніми поштовими процесами, що включає облік, реєстрацію, маршрут переміщення та доступ на різних рівнях. Саме така задача поставлена в межах цього дипломного проєкту.

2. ОБГРУНТУВАННЯ ВИБОРУ ЗАСОБІВ РЕАЛІЗАЦІЇ

Розробка сучасного вебзастосунка вимагає ретельного підбору інструментів та технологій, які забезпечать ефективність, масштабованість, надійність та зручність подальшого супроводу системи. З огляду на функціональні вимоги до дипломного проєкту, особливо пов’язані з обліком поштових відділень, обробкою відправлень, забезпеченням ролей користувачів та взаємодією з геолокаційними сервісами, критично важливо обрати оптимальний стек технологій.

У цьому розділі розглядаються критерії вибору інструментів, таких як мова програмування, фреймворки, бібліотеки, система керування базами даних, а також допоміжні сервіси, що використовуватимуться при реалізації системи. Вибір кожного засобу обґрунтовується з точки зору його відповідності потребам системи, простоти інтеграції, активності спільноти підтримки та відповідності сучасним практикам розроблення програмного забезпечення.

2.1. Вибір мови програмування

Одним з ключових рішень при розробці інформаційної системи є вибір мови програмування. Оскільки проєкт реалізується у вигляді вебзастосунку, важливою є підтримка роботи з мережею, базами даних, вебінтерфейсами, а також швидкість розробки та доступність екосистеми. Нижче наведена таблиця порівняння мов програмування за критеріями оцінки. Ці критерії є визначальними для забезпечення ефективної, масштабованої та зручної у підтримці розробки системи. Обґрунтований вибір мови програмування дозволяє не лише оптимізувати процес розробки, а й забезпечити подальший розвиток і адаптацію системи до нових вимог.

На таблиці 2 показано порівняння мов програмування за загальними критеріями.

Таблиця 2.1

Порівняння мов програмування

Критерій / Мова	JavaScript / TypeScript	Python	PHP	Java	C# (.NET)
Підтримка фронтенду	+	–	–	–	–
Підтримка бекенду	+	+	+	+	+
Статична типізація	+	–	–	+	+
Простота у навчанні	+	+	+	–	–
Швидкість розробки	+	+	+	–	–
Масштабованість	+	–	–	+	+
Підтримка спільнотою	+	+	–	+	+
Уніфікований стек	+	–	–	–	–

Після порівняння основних мов програмування для розробки вебзастосунку можна зробити висновок, що JavaScript у поєднанні з TypeScript є найоптимальнішим вибором.

Інші мови, хоча й мають свої переваги, поступаються в контексті швидкої, сучасної, модульної розробки вебсистем, орієнтованих як на користувача, так і на сервер. Наприклад, Python не підходить для клієнтської частини, а PHP хоч і традиційно використовувався для клієнтської частини, але морально застарів для складних SPA-застосунків. Java та C# мають високу надійність, але значно вищий поріг входу та більші ресурси для підтримки.

Отже, вибір JavaScript/TypeScript був ухвалений як оптимальний баланс між ефективністю, зручністю розробки та функціональністю.

2.2. Вибір фреймворку для клієнтської частини

Одним із важливих аспектів при створенні сучасного вебзастосунка є вибір фреймворку для клієнтської частини, оскільки саме він визначає архітектуру інтерфейсу користувача, взаємодію з сервером та загальний користувацький досвід. У процесі аналізу було розглянуто кілька популярних рішень: React, Vue.js та Angular – кожен із них має свої переваги та недоліки, що впливають на кінцевий вибір.

React є найпопулярнішим фреймворком (технічно – бібліотекою), який підтримується компанією Meta та має надзвичайно велику спільноту [6]. Його перевагами є висока продуктивність, компонентна архітектура, велика кількість сторонніх бібліотек, гнучкість у виборі інструментів для маршрутизації, управління станом та інших аспектів розробки.

Vue.js – це легкий і швидкий фреймворк, що часто обирається завдяки простоті входження та зручному синтаксису. Проте у великих корпоративних застосунках Vue поступається за гнучкістю налаштування та масштабованістю, порівняно з React.

Angular, розроблений Google, надає всеосяжне рішення «з коробки», включаючи вбудовану маршрутизацію, інструменти для форм, HTTP-запитів та управління станом. Проте через свою складність і високу криву навчання він менш зручний для невеликих та середніх проєктів.

З огляду на вищенаведене, React було обрано як оптимальне рішення для реалізації клієнтської частини системи завдяки його популярності, підтримці спільноти, широкому набору інструментів і здатності адаптуватися до складних архітектурних рішень.

2.3. Вибір фреймворку для серверної частини

При виборі фреймворку для серверної частини інформаційної системи ключовими критеріями стали: продуктивність, підтримка REST-архітектури, зручність роботи з базами даних, масштабованість, активність спільноти та легкість у підтримці коду. Розглядалися кілька популярних фреймворків, серед яких Express.js (Node.js), Django (Python), Spring Boot (Java) та Laravel (PHP).

Express.js – мінімалістичний фреймворк для Node.js [4], що дозволяє швидко створювати вебсервери та API. Його головними перевагами є висока гнучкість, простота інтеграції з базами даних та широке розповсюдження в екосистемі JavaScript. Express ідеально підходить для створення RESTful API, що й було ключовою вимогою проєкту [5].

Django – повнофункціональний фреймворк Python, який забезпечує швидку розробку та надійність завдяки суворій структурі, інтегрованій ORM та вбудованій панелі адміністратора. Попри переваги, його використання потребує окремої мови програмування (Python), що створює зайву складність у повному JavaScript-стеку.

Spring Boot – потужний фреймворк для Java, орієнтований на корпоративні застосунки з високими вимогами до безпеки, масштабованості та інтеграції. Проте його складність і перевантаженість функціональністю не є доцільними для вебзастосунку середнього рівня.

Laravel – PHP-фреймворк, зручний у використанні та добре задокументований. Однак, як і Django, він базується на іншій мові програмування, що ускладнює уніфікацію клієнтської та серверної частин.

З огляду на всі порівняльні аспекти, було прийнято рішення використовувати Express.js, оскільки він дозволяє реалізувати гнучку, швидку у розробці та легку у підтримці серверну частину, повністю інтегровану з клієнтським середовищем, побудованим на JavaScript.

2.4. Вибір СУБД

Вибір системи управління базами даних (СУБД) є критично важливим етапом у розробці інформаційної системи, оскільки від нього залежить ефективність зберігання, доступу, цілісність і масштабованість даних. Основними критеріями при виборі СУБД для цього проєкту були:

1) підтримка реляційної моделі;

2) надійність і стабільність;

3) активна спільнота та документація;

4) зручна інтеграція з обраним серверним фреймворком;

5) гнучкість у розгортанні на різних хостингах.

Було проаналізовано кілька популярних рішень: PostgreSQL, MySQL, SQLite та MongoDB.

PostgreSQL – потужна об’єктно-реляційна СУБД з відкритим вихідним кодом, яка підтримує складні запити, транзакції, перевірку цілісності даних, індексацію та розширення [9]. Вона ідеально підходить для систем, що вимагають високої надійності, структурованих зв’язків і масштабованості.

MySQL також є популярною реляційною СУБД, однак поступається PostgreSQL у складності запитів, підтримці стандартів SQL та функціональності, особливо при роботі зі складними реляціями.

SQLite є легкою СУБД, що зберігає всю базу в одному файлі. Підходить для мобільних застосунків або прототипів, але не забезпечує достатньої продуктивності та багатокористувацького доступу для повноцінного вебзастосунку.

MongoDB – нереляційна документо-орієнтована СУБД. Підходить для гнучких, слабо структурованих даних, але не ідеальна у випадках, коли важлива строгість зв’язків та нормалізована структура.

З огляду на вищенаведене, було обрано PostgreSQL як оптимальну СУБД для реалізації даного вебзастосунку. Вона дозволяє ефективно працювати зі складними зв’язками між сутностями (наприклад, відділення,

відправлення, кур’єри), забезпечує високу продуктивність, сумісність з ORM-бібліотеками (такими як Sequelize або Prisma), та легко інтегрується в JavaScript-екосистему через Node.js.

2.5. Вибір мови програмування для мобільного додатку

У межах дипломного проєкту однією з ключових задач стало створення зручного мобільного застосунку для кур’єрів, який дозволить швидко отримувати інформацію про поточні замовлення, маршрути доставлення, а також фіксувати статус виконання кожного відправлення. Для реалізації цієї частини системи необхідно було обрати інструменти, які забезпечать високу стабільність роботи, інтеграцію з серверною частиною, доступ до апаратних можливостей пристрою (геолокація, камера, інтернет-з’єднання) та зручну розробку і супровід застосунку.

Серед наявних варіантів розглядались як нативні підходи (Java, Kotlin для Android), так і кросплатформенні рішення (React Native, Flutter). Попри привабливість ідеї створення єдиного застосунку для кількох платформ, було вирішено зосередитись на розробці саме нативного Android-застосунку з використанням мови Kotlin. Це пояснюється насамперед тим, що основною платформою використання застосунку визначено Android, з огляду на широку поширеність пристроїв з цією ОС у сфері логістики. Kotlin, як сучасна мова програмування, рекомендована самою компанією Google для створення Android-додатків, забезпечує повну підтримку Android API, високу продуктивність та знижену ймовірність помилок завдяки системі null-безпеки.

На відміну від Java, Kotlin має більш лаконічний синтаксис, що дозволяє зменшити обсяг коду без втрати його читабельності, а також підтримує функціональні парадигми, що спрощують обробку даних. У порівнянні з Flutter та React Native, Kotlin не потребує проміжних шарів для доступу до нативних функцій пристрою, що важливо в контексті реалізації таких критичних можливостей, як відстеження геолокації в реальному

часі [12]. Отже, вибір Kotlin як мови розробки мобільного застосунку є обґрунтованим з точки зору продуктивності, стабільності та довгострокового супроводу рішення.

2.6. Висновки до розділу

Загалом, вибір технологій для реалізації інформаційної системи обліку поштових відправлень і роботи кур'єрів був продиктований прагненням до ефективності, масштабованості та зручності підтримки проєкту в довгостроковій перспективі. Усі обрані інструменти мають активну спільноту, добре задокументовані рішення та перевірені практикою можливості, що забезпечує стабільність і надійність системи в реальному використанні.

Кожен компонент проєкту було реалізовано з урахуванням його функціонального навантаження: для клієнтської частини обрано React – сучасний, продуктивний і зручний для побудови динамічних інтерфейсів фреймворк; для серверної частини – Node.js з Express, що забезпечує швидку обробку запитів і гнучку архітектуру REST API; як систему керування базами даних використано PostgreSQL – потужну та надійну реляційну СУБД, що чудово підходить для складних структурованих даних; мобільний застосунок реалізовано мовою Kotlin, яка є стандартом для Android-розробки та дозволяє використовувати всі можливості платформи.

Таким чином, обраний стек технологій дозволяє створити повноцінну, надійну та зручну у користуванні інформаційну систему, яка відповідає сучасним вимогам до логістичних та облікових сервісів, а також забезпечує основу для подальшого масштабування та розвитку.

3. СТРУКТУРНО-АЛГОРИТМІЧНА ОРГАНІЗАЦІЯ СИСТЕМИ

Важливим етапом розробки інформаційної системи є проєктування її внутрішньої структури та логіки взаємодії компонентів. Саме на цьому етапі визначається, як користувацькі запити будуть оброблятись, які модулі відповідатимуть за зберігання та обробку даних, а також яким чином реалізовуватиметься логіка бізнес-процесів. Раціональна структурно-алгоритмічна організація системи забезпечує її стабільну та ефективну роботу, дозволяє уникнути дублювання функціоналу та спрощує подальшу підтримку і масштабування.

У даному розділі розглядаються основні архітектурні рішення, схема взаємодії між модулями, а також детальна логіка виконання ключових операцій. Особливу увагу приділено модульності, повторному використанню коду та дотриманню принципів чистої архітектури. Також буде проаналізовано, яким чином реалізовано обробку запитів, авторизацію, маршрутизацію, валідацію даних і обмін інформацією між серверною, клієнтською та мобільною частинами системи.

Загальна мета цього етапу – забезпечити зрозумілу, логічно обґрунтовану й ефективну організацію роботи системи, яка відповідатиме вимогам до надійності, безпеки та продуктивності, необхідним для успішного функціонування платформи в реальних умовах.

3.1. Аналіз проблематики та цілей вебзастосунку

Для ефективної розробки інформаційної системи необхідно чітко усвідомити основні проблеми та виклики, які існують у сфері обліку поштових відділень і відправлень. Для цього доцільно сформувати дерево проблем – структуровану схему, що відображає взаємозв’язки між основними труднощами, їхніми причинами та наслідками. На верхньому рівні дерева розташована загальна проблема – низька ефективність

управління поштовими відділеннями та доставленням відправлень. Ця проблема проявляється у кількох ключових аспектах:

1. Незручність у контролі та обліку поштових відправлень, що призводить до втрати посилок або затримок у доставленні.
2. Відсутність прозорості інформації для користувачів та співробітників, що ускладнює моніторинг статусу відправлень і планування роботи.
3. Недостатня автоматизація процесів, через що значна частина операцій виконується вручну, що підвищує ризик помилок та знижує продуктивність.

Дерево проблем з декомпозицією перелічених раніше пунктів наведене на рис. 3.1.

Рис. 3.1. Дерево проблем користувачів пошти

Причинами цих проблем є, зокрема, відсутність єдиної інтегрованої системи обліку, недостатня комунікація між різними учасниками процесу (адміністраторами, операторами, кур’єрами), а також недосконаліть існуючих інформаційних систем, які не враховують специфіку роботи поштових сервісів.

Наслідками цих проблем є незадоволеність клієнтів, втрата довіри до поштових сервісів, а також збільшення операційних витрат через неефективність роботи персоналу.

Саме виходячи з аналізу цього дерева проблем було сформовано вимоги до розроблюваної системи, які покликані усунути виявлені недоліки та підвищити якість управління процесами поштових відділень і доставлення.

Для ефективного проєктування інформаційної системи доцільно сформувати дерево цілей, яке базується на раніше проаналізованому дереві проблем. Такий підхід дозволяє перейти від опису існуючих недоліків до формулювання конкретних задач, які система має вирішити. Дерево цілей структурує очікувані результати, розділяючи глобальну мету на підцілі різного рівня деталізації, що в подальшому дає змогу чітко визначити функціональні вимоги до майбутньої системи.

Аналізуючи сформоване дерево проблем, можна визначити низку ключових напрямів, які потребують реалізації. Це зокрема централізований облік поштових відділень, контроль за відправленнями всередині мережі, автоматизація доставлення, підтримка ролей користувачів і мобільна взаємодія з кур’єрами. Всі ці напрями формують підґрунтя для побудови дерева цілей, у якому кожна ціль прямо відповідає на певну проблему і орієнтована на усунення її причин або наслідків.

На рис. 3.2 наведено візуальне представлення дерева цілей.

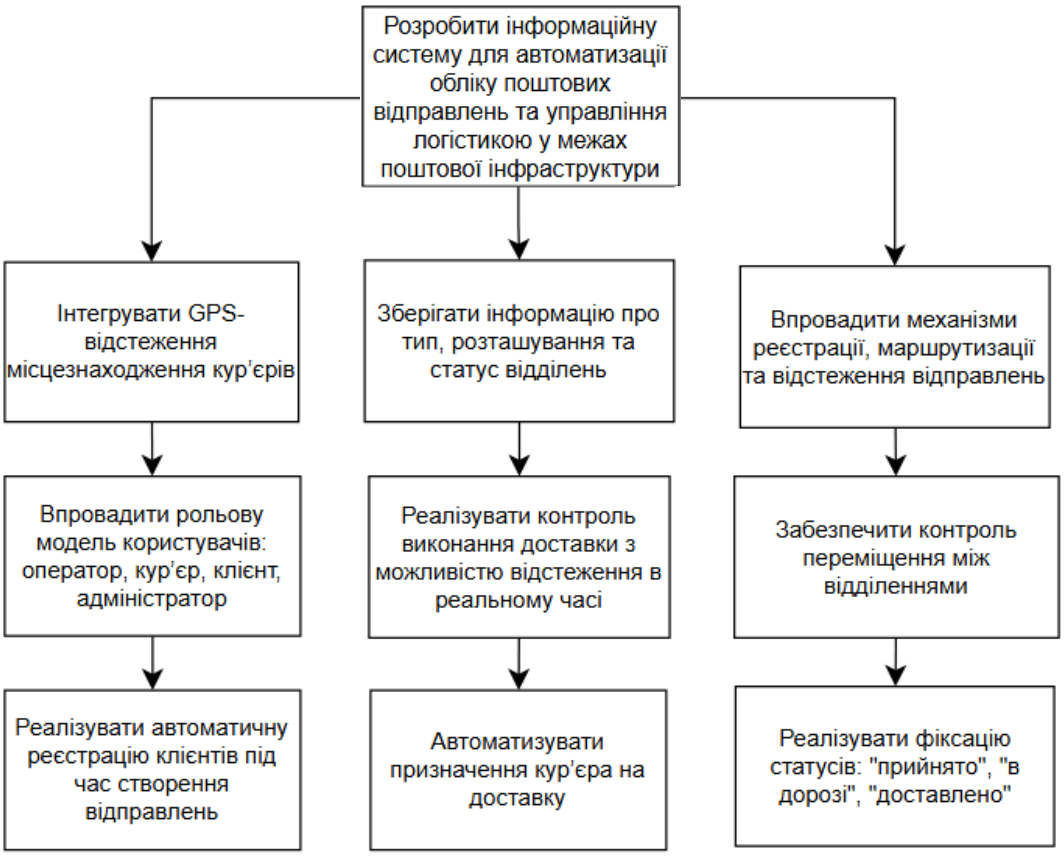


Рис. 3.2. Дерево цілей

Для завершення логічного переходу від виявлених проблем до практичної реалізації рішень, необхідно сформувати дерево результатів. Воно ґрунтується на раніше побудованому дереві цілей і демонструє очікувані наслідки досягнення цих цілей, що дозволяє встановити причинно-наслідкові зв’язки між запланованими діями та очікуваними змінами. Іншими словами, дерево результатів – це уявлення про те, які конкретні позитивні зміни будуть досягнуті внаслідок реалізації проєкту, які потреби користувачів буде задоволено, та які проблеми – усунуто.

Його використання дозволяє оцінити доцільність функціональних рішень, виявити надлишкові або недостатні елементи системи, а також спрогнозувати ефективність розробленої інформаційної системи у коротко- та довгостроковій перспективі. Крім того, дерево результатів дає змогу структуровано обґрунтувати вибір технічних засобів, методів реалізації та

підходів до впровадження, спираючись на чітко окреслені очікування та потреби замовника.

Такий інструмент відіграє важливу роль у стратегічному плануванні ІТ-проєкту, сприяє прозорості розробки, підвищує керованість проєктом на всіх етапах життєвого циклу системи та допомагає уникнути розбіжностей між розробниками, замовником і кінцевими користувачами щодо цілей і очікуваних результатів.

На рис. 3.3 наведено візуальне представлення дерева результатів.

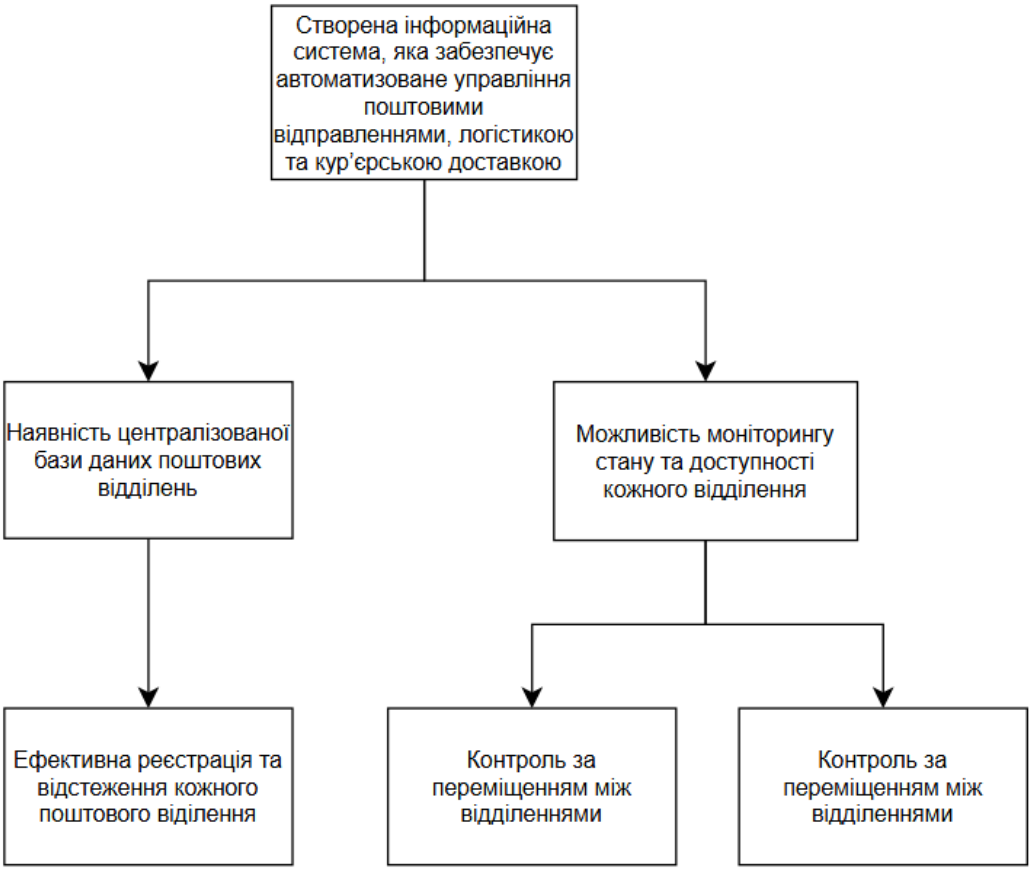


Рис. 3.3. Дерево результатів

3.2. Формулювання вимог до програмного забезпечення

На основі побудованого дерева цілей та дерева результатів було сформовано перелік функціональних і нефункціональних вимог до розроблюваної інформаційної системи. Такий підхід дав змогу системно окреслити очікувані результати впровадження, враховуючи потреби

користувачів, специфіку предметної області та технічні обмеження. Зокрема, було визначено пріоритети щодо підвищення ефективності управління, автоматизації рутинних операцій, зменшення людського фактору у виконанні стандартних процедур, а також забезпечення зручного, інтуїтивно зрозумілого інтерфейсу.

Одним з ключових етапів деталізації вимог є опис сценаріїв використання (use cases), що дозволяє не лише виявити реальні шляхи взаємодії кінцевих користувачів із системою, а й наочно змоделювати типові робочі ситуації.

Завдяки аналізу сценаріїв використання можна чітко визначити основні функціональні можливості, які має реалізовувати система для задоволення поставлених цілей. Це, у свою чергу, дозволяє розробникам приймати обґрунтовані архітектурні рішення, забезпечує повноту покриття прикладних задач, а також полегшує формування критеріїв прийнятності на етапі тестування.

Сценарії використання будуть згруповані за типами користувачів, які взаємодіють із системою. Основними ролями є:

1. Адміністратор – відповідає за створення облікових записів, конфігурацію системи, керування відділеннями та персоналом.
2. Оператор поштового відділення – здійснює реєстрацію відправлень, видачу та приймання посилок, роботу з клієнтами.
3. Кур’єр – отримує завдання на доставлення, підтверджує вручення, відправляє дані про геолокацію.
4. Клієнт – створює запит на відправлення або очікує доставлення, може переглядати статус своєї посилки.
5. Система – обробляє події, генерує сповіщення, синхронізує мобільний додаток і базу даних.

Кожен сценарій включатиме такі структурні елементи:

1. Назва сценарію – коротка, описова назва дії або процесу.

2. Учасники (ролі) – користувачі або системні модулі, які беруть участь у сценарії.
3. Мета – опис цілі, яку намагається досягти ініціатор сценарію.
4. Початкові умови – стан системи або умови, які мають бути виконані до початку сценарію.
5. Очікуваний результат – результат, що вважається успішним завершенням сценарію.
6. Основний потік подій – послідовність дій, яка веде до досягнення мети в нормальних умовах.
7. Альтернативні потоки – можливі відхилення або помилки в ході виконання, а також реакція системи на них.

Нижче в описаному форматі наведено сценарії у вигляді таблиць.

Таблиця 3.1

UC-1 Реєстрація нового клієнта при оформленні відправлення

UC-1	Реєстрація нового клієнта при оформленні відправлення
Актори	Оператор
Початкові умови	Оператор авторизований у системі, оформлює нове відправлення, клієнт ще не зареєстрований у системі
Очікуваний результат	Система автоматично створює профіль клієнта та прив’язує до відправлення
Основний потік	1. Оператор відкриває форму створення нового відправлення. 2. Вводить дані відправника, які ще не існують у базі. 3. Система виконує перевірку по базі клієнтів. 4. Якщо клієнта не знайдено, система створює нового користувача з роллю «Клієнт». 5. Відправлення успішно зберігається з новим клієнтом.
Альтернативні потоки	3а. Дані клієнта вже існують. 3а-1. Система підтягує існуючий профіль клієнта.

Таблиця 3.2

UC-2 Призначення кур’єра на відправлення

UC-2	Призначення кур’єра на відправлення
Актори	Оператор
Початкові умови	Відправлення створене, оператор авторизований за паролем, наданим адміністратором при його реєстрації у відділенні
Очікуваний результат	Кур’єру призначено відправлення для доставлення
Основний потік	1. Оператор відкриває перелік відправлень. 2. Обирає конкретне відправлення. 3. Вказує кур’єра для доставлення. 4. Система оновлює статус відправлення та додає кур’єра. Відправлення успішно зберігається з новим клієнтом.
Альтернативні потоки	3а. Кур’єр зайнятий іншими доставками. 3а-1. Система виводить попередження, оператор обирає іншого.

Таблиця 3.3

UC-3 Відстеження геолокації кур’єра

UC-3	Відстеження геолокації кур’єра
Актори	Система, Адміністратор/Оператор
Початкові умови	Кур’єр знаходиться в дорозі з активним GPS
Очікуваний результат	На мапі в реальному часі показується місцеперебування кур’єра або видається стисле повідомлення про помилку
Основний потік	1. Мобільний додаток передає координати. 2. Сервер обробляє і відображає їх на карті у кабінеті адміністратора/оператора.

Продовження табл. 3.3

Альтернативні потоки	1а. Зв’язок із GPS втратився. 1а-1. Система відображає останню доступну позицію.

Таблиця 3.4

UC-4 Перегляд статусу відправлення клієнтом

UC-4	Перегляд статусу відправлення клієнтом
Актори	Клієнт
Початкові умови	Клієнт має код або номер відправлення
Очікуваний результат	Відображення актуального статусу доставлення
Основний потік	1. Клієнт заходить у вебкабінет або вводить код на сайті. 2. Система знаходить відправлення. 3. Відображає статус і локацію на мапі.
Альтернативні потоки	2а. Код введено з помилкою. 2а-1. Система виводить повідомлення про помилку.

Таблиця 3.5

UC-5 Додавання нового поштового відділення

UC-5	Додавання нового поштового відділення
Актори	Адміністратор
Початкові умови	Адміністратор авторизований, працює в панелі з відділеннями та операторами які в них теперішньо працюють
Очікуваний результат	Нове відділення створено та доступне в системі
Основний потік	1. Адміністратор відкриває меню «Відділення». 2. Натискає «Додати нове».

Продовження табл. 3.5

	3. Заповнює форму (назва, адреса, тощо). 4. Натискає «Зберегти». 5. Система зберігає відділення.
Альтернативні потоки	3а. Адреса вже зайнята. 3а-1. Система виводить помилку та не дозволяє дублювання.

На підставі визначених випадків використання (UC-1 – UC-5), розроблено перелік функціональних вимог до вебзастосунку, який дозволяє користувачам здійснювати авторизацію, переглядати та керувати бронюваннями, а адміністраторам – схвалювати чи відхиляти запити. Наведена таблиця 3.6 допоможе визначити ключові завдання для подальшої реалізації та оцінити їх складність і пріоритетність у розробці.

Таблиця 3.6

Реєстр функціональних вимог до вебзастосунку

Код вимоги	Зміст вимоги	Складність	Пріоритетність
FR-1	Система повинна дозволяти користувачам проходити реєстрацію, яку виконує адміністратор та виконувати автентифікацію за паролем, наданим при реєстрації.	Середня	Висока
FR-2	Авторизований користувач повинен мати змогу переглядати доступні кімнати.	Низька	Висока
FR-3	Користувач може здійснити бронювання вільної клітинки кімнати.	Середня	Висока

Продовження табл. 3.6

FR-4	Користувач може переглядати інформацію про власні бронювання.	Низька	Висока
FR-5	Користувач може редагувати власне бронювання, якщо він є його організатором.	Середня	Висока
FR-6	Користувач може видаляти власне бронювання.	Низька	Висока
FR-7	Адміністратор може переглядати всі бронювання та змінювати їхній статус.	Висока	Середня
FR-8	Адміністратор може схвалювати або відхиляти бронювання користувачів.	Середня	Середня
FR-9	Система повинна валідувати дані під час створення або редагування бронювання.	Середня	Висока

Нефункціональні вимоги визначають загальні характеристики вебзастосунку, які не залежать безпосередньо від функціональності, але мають суттєвий вплив на якість, стабільність, масштабованість та зручність користування. Нижче наведено основні вимоги, розділені за трьома ключовими напрямами.

Таблиця 3.7

Нефункціональні вимоги до безпеки

Код	Зміст вимоги
SEC-1	Дані користувачів повинні передаватися по захищеному протоколу HTTPS.
SEC-2	Паролі користувачів повинні зберігатися у зашифрованому вигляді (наприклад, bcrypt).

Продовження табл. 3.7

SEC-3	Система повинна запобігати несанкціонованому доступу до функцій через авторизацію.
SEC-4	Рольовий доступ повинен обмежувати дії користувачів згідно з їх роллю.
SEC-5	Повторні невдалі спроби входу мають блокувати доступ на певний час.

Таблиця 3.8

Нефункціональні вимоги до продуктивності

Код	Зміст вимоги
PERF-1	Система повинна обробляти запити користувачів не довше ніж за 2 секунди.
PERF-2	Система має витримувати щонайменше 100 одночасних користувачів без втрати стабільності.
PERF-3	Оновлення інформації в інтерфейсі має відбуватися без повного перезавантаження сторінки (через AJAX або WebSockets).

Таблиця 3.9

Нефункціональні вимоги до реалізації

Код	Зміст вимоги
IMPL-1	Система має бути реалізована як вебзастосунок із клієнтською та серверною частинами.
IMPL-2	Для клієнтської частини має використовуватись сучасний фреймворк (наприклад, React).
IMPL-3	Серверна частина повинна бути реалізована із застосуванням REST API.
IMPL-4	Система повинна підтримувати розгортання у контейнерному середовищі (наприклад, Docker).

Продовження табл. 3.9

IMPL-5	База даних має бути збережена у реляційній СУБД (наприклад, PostgreSQL або MySQL).
IMPL-6	Інтерфейс повинен бути адаптованим для різних розмірів екранів (responsive design).

3.3. Опис архітектури системи

Архітектура інформаційної системи – це логічна структура взаємодії її компонентів, яка визначає принципи побудови, способи обміну даними та розмежування функціональності між частинами програми. Грамотно спроєктована архітектура сприяє масштабованості, підтримці й безпечному розвитку системи.

Для розробки системи управління логістичними процесами обрано клієнт-серверну архітектуру, яка передбачає чіткий поділ на окремі компоненти:

1. Вебклієнт (адміністративна панель) – використовується операторами та менеджерами для контролю замовлень, маршрутів, спостереження за кур'єрами, а також клієнтами для відстеження відправлень.
2. Мобільний додаток для кур'єрів – окремий застосунок, який працює на Android/iOS-пристроях і виконує збір GPS-даних у реальному часі, відправляючи їх на сервер.
3. Серверна частина (backend) – центральний логічний компонент, який приймає та обробляє запити, зберігає інформацію в базі даних, забезпечує аутентифікацію користувачів та передачу даних між частинами.
4. База даних – сховище для інформації про користувачів, замовлення, місцеперебування, маршрути, статуси та історію виконаних дій.

Узагальнену архітектуру застосунку наведено на рис. 3.4.

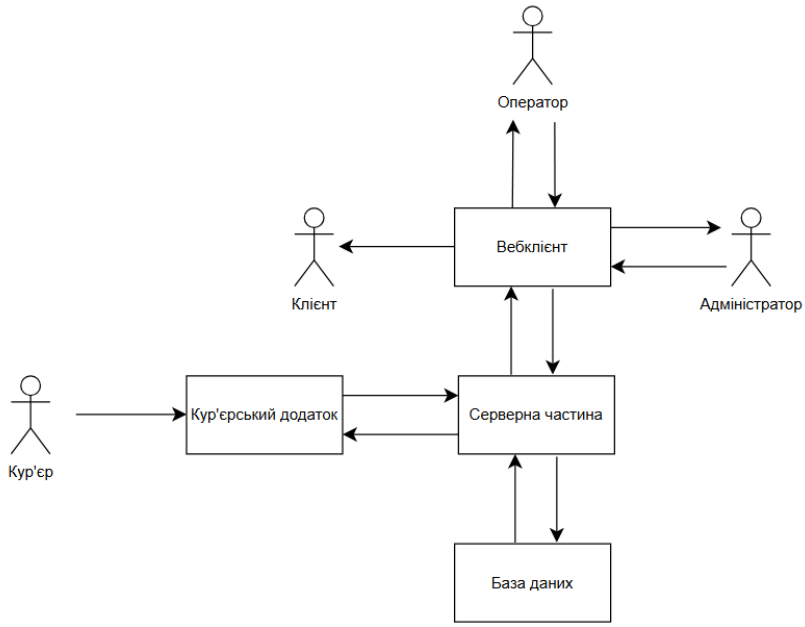


Рис. 3.4. Узагальнена структура застосунку

3.4. Опис структур даних системи

База даних вебзастосунку для відстеження поштових відправлень розроблена з урахуванням принципів нормалізації та передбачає розподілення інформації на логічно зв'язані сутності.

На рис. 3.5 наведено взаємозв'язки та опис таблиць бази даних застосунку.

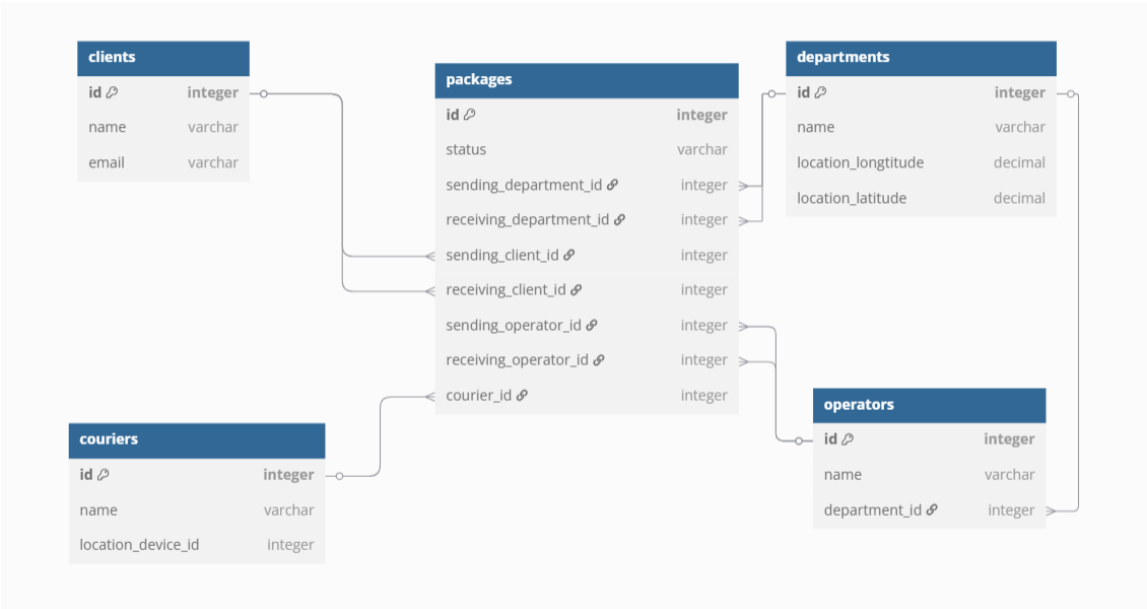


Рис. 3.5. База даних застосунку

Архітектура бази даних складається з наступних сутностей:

1. Клієнти: містить дані про осіб або організації, що надсилають або отримують посилки. Містить наступні поля:
   - id (integer) – унікальний ідентифікатор клієнта;
   - name (varchar) – ім’я або назва клієнта;
   - email (varchar) – електронна адреса для зворотного зв’язку.
2. Оператори: описує працівників, що обробляють посилки в департаментах. Містить наступні поля:
   - id (integer) – унікальний ідентифікатор оператора;
   - name (varchar) – ім’я оператора;
   - department_id (integer) – зовнішній ключ на таблицю departments.
3. Кур'єри: відповідають за фізичне транспортування посилок. Містить наступні поля:
   - id (integer) – унікальний ідентифікатор кур'єра;
   - name (varchar) – ім’я кур'єра;
   - location_device_id (integer) – ідентифікатор пристрою відстеження розташування.
4. Відділення: містять інформацію про логістичні відділення. Містить наступні поля:
   - id (integer) – унікальний ідентифікатор відділення;
   - name (varchar) – назва відділення;
   - location_longitude (decimal) – географічна довгота;
   - location_latitude (decimal) – географічна широта.
5. Посилки: основна сутність, що описує кожну поштову доставлення. Містить наступні поля:
   - id (integer) – унікальний ідентифікатор посилки;
   - status (varchar) – поточний статус доставлення;
   - sending_department_id (integer) – відділення-відправник;
   - receiving_department_id (integer) – відділення-одержувач;

- sending_client_id (integer) – клієнт, який відправляє;
- receiving_client_id (integer) – клієнт, який отримує;
- sending_operator_id (integer) – оператор, який створив відправлення;
- receiving_operator_id (integer) – оператор, який приймає посилку;
- courier_id (integer) – кур'єр, закріплений за доставленням.

3.5. Опис реалізованих алгоритмів функціонування системи

На рис. 3.6 наведено алгоритм реєстрації нового відправлення.

Рис. 3.6. Алгоритм реєстрації відправлення

Розроблена інформаційна система для відстеження поштових відправлень охоплює широкий спектр логістичних і адміністративних процесів, які потребують чіткої реалізації внутрішніх алгоритмів. Основна мета реалізації алгоритмічної логіки – забезпечити коректність, безперервність та надійність функціонування всіх ключових сценаріїв взаємодії користувачів із системою.

У процесі розробки було впроваджено низку алгоритмів, які автоматизують ключові етапи обробки відправлень, включаючи створення, маршрутизацію, відстеження статусу та завершення доставлення. Особливу увагу було приділено автоматичній реєстрації нових клієнтів у разі відсутності їх у базі, що дозволяє уникнути зайвих дій для оператора та забезпечує безперервність процесу.

Процес реєстрації нового відправлення в системі реалізовано з урахуванням автоматичного створення клієнтів і спрощення дій для оператора. На початковому етапі система перевіряє, чи існує у базі даних клієнт з вказаною електронною адресою відправника. Якщо такий клієнт знайдений, система використовує його унікальний ідентифікатор. У разі, якщо клієнта не існує, автоматично створюється новий профіль: генерується тимчасовий пароль, який хешується за допомогою бібліотеки bcrypt, після чого у таблиці profiles створюється новий запис з роллю client. Одразу після цього новому клієнту надсилається електронне повідомлення з обліковими даними для доступу до системи.

Після отримання ідентифікаторів клієнтів система переходить до формування самого відправлення. У таблицю parcels заноситься новий запис, що містить ідентифікатори клієнтів, відділень-відправника та одержувача, а також кур’єра, який відповідатиме за транспортування. Статус відправлення на цьому етапі встановлюється як «being delivered». Дата створення фіксується автоматично через поле created_at.

Після успішного створення відправлення система надсилає електронні сповіщення обом сторонам: відправнику – з підтвердженням

створення або реєстрації в системі, а одержувачу – з повідомленням про надходження посилки. Нарешті, API повертає відповідь клієнту з кодом 201 Created та ідентифікатором створеного відправлення у форматі JSON. Така реалізація дозволяє максимально автоматизувати процес обробки, зменшити навантаження на операторів і підвищити зручність використання системи для клієнтів.

3.6. Висновки до розділу

У межах розділу було здійснено комплексний аналіз та побудову структурної й алгоритмічної організації інформаційної системи для відстеження поштових відправлень. На основі виявлених проблем предметної області сформовано дерево проблем, дерево цілей та дерево результатів, що дозволило логічно обґрунтувати постановку завдання та структуру вимог до системи.

Було визначено ключові сценарії використання, які охоплюють основні дії всіх учасників системи: адміністратора, оператора, кур’єра та клієнта. Для кожного сценарію сформульовано функціональні вимоги з урахуванням складності та пріоритетності. Окремо виділено нефункціональні вимоги до безпеки, продуктивності та реалізації системи, що дозволяє забезпечити надійність і масштабованість під час експлуатації.

Алгоритмічна частина описує внутрішню логіку ключових процесів системи, зокрема реєстрації відправлень, автоматичної реєстрації клієнтів, оновлення статусу доставлення та контролю доступу до дій на основі ролей. Сукупність розроблених структурних компонентів та алгоритмів закладає надійну основу для реалізації системи, що відповідає реальним логістичним процесам у поштовій сфері.

4. ОСОБЛИВОСТІ РЕАЛІЗАЦІЇ ВЕБЗАСТОСУНКУ

4.1. Особливості реалізації

Розробка сучасного вебзастосунка вимагає не лише побудови правильної архітектури, а й практичної реалізації всіх функціональних компонентів системи з урахуванням потреб користувачів, технічних обмежень та вибраного технологічного стеку. У даному розділі розглядаються ключові аспекти реалізації вебзастосунку PostTrack, спрямованого на автоматизацію процесів обліку, доставлення та відстеження поштових відправлень.

Особлива увага приділяється інтеграції з картографічним сервісом для візуалізації геолокаційних даних, реалізації багаторівневої автентифікації та рольового доступу, створенню інтуїтивного користувацького інтерфейсу, а також побудові REST API для взаємодії між клієнтською та серверною частинами. Крім того, описується структура сторінок вебінтерфейсу для кожної ролі – адміністратора, оператора, клієнта та кур’єра.

Метою розділу є деталізований аналіз процесу реалізації застосунку: від логіки обробки запитів до практичної взаємодії з базою даних і зовнішніми сервісами. Такий підхід дозволяє не лише продемонструвати технічну складову проєкту, але й обґрунтувати вибрані рішення з точки зору ефективності, зручності та розширюваності системи.

4.1.1. Серверна частина вебзастосунку

У серверній частині застосунку реалізовано централізовану систему маршрутизації за допомогою фреймворку Express.js. Усі HTTP-запити клієнта спрямовуються до відповідного маршруту, визначеного у модулі router, який об'єднує всі функціональні частини системи: автентифікацію, управління відділеннями, обробку посилок, оновлення координат кур’єрів тощо. Кожен маршрут відповідає за конкретну логіку – наприклад, /register

або /parcel/client – що забезпечує зрозумілу структуру API для frontend-розробників.

Важливою особливістю є використання middleware-функцій, таких як authMiddleware та authorizeAdmin, які перед виконанням основної логіки маршруту перевіряють токен користувача та його роль. Це дозволяє ефективно реалізувати захист маршрутів від несанкціонованого доступу, зберігаючи при цьому чистоту основної бізнес-логіки в контролерах.

Серверна частина реалізована відповідно до патерну MVC (Model-View-Controller), що забезпечує чіткий розподіл відповідальностей між частинами застосунку. Моделі представлені абстракціями бази даних, до яких звертаються контролери за допомогою ORM-інструменту knex [10]. Це дозволяє формувати запити до бази даних у зручному форматі, не використовуючи сирі SQL-інструкції.

Контролери реалізують основну бізнес-логіку кожного модуля (наприклад, departmentsController.ts), опрацьовують запити, виконують перевірки, взаємодіють із моделями та формують відповіді у форматі JSON. У системі відсутній рівень представлення (View), оскільки сервер функціонує як REST API – тому роль View умовно виконує структура відповіді, що передається на frontend. Такий підхід є стандартним у сучасних SPA-архітектурах і добре масштабується.

Усі основні сутності в системі (відділення, профілі користувачів, посилки) підтримують повний набір CRUD-операцій. Наприклад, для departments реалізовано окремі маршрути для створення (POST /departments), читання (GET /departments та GET /departments/:id), оновлення (PUT /departments/:id) та видалення (DELETE /departments/:id). Кожна операція виконує відповідні дії у базі даних через knex і формує відповідь із повідомленням про успіх або помилку.

Окрема увага приділена обробці складніших запитів, які вимагають JOIN-зв'язків між таблицями, зокрема при отриманні вхідних посилок або списку кур’єрів з координатами. Такі запити не лише виводять ID, а й

включають пов’язану інформацію (email, статус, назви відділень), що значно полегшує роботу клієнтської частини і зменшує кількість API-запитів.

Автентифікація реалізована за допомогою JWT (JSON Web Tokens). Під час входу користувача у систему (через маршрут /login) його облікові дані перевіряються, і в разі успіху сервер генерує токен, що містить інформацію про роль і ID користувача. Цей токен зберігається на клієнті (наприклад, у localStorage) і передається у заголовках до наступних запитів.

У системі реалізована інтеграція з поштовим сервісом Gmail через бібліотеку nodemailer. Посилання електронних листів виконується у кількох критичних точках: при створенні відправлення клієнтам надсилаються сповіщення про нову посилку, а у випадку, коли клієнт не зареєстрований – автоматично генерується пароль, і на пошту надходять облікові дані для входу в систему.

Надсилання листів винесено в окремий сервіс emailService.ts, що підвищує гнучкість і дозволяє легко масштабувати логіку в майбутньому (наприклад, додати шаблони HTML, підтримку кількох поштових провайдерів, логування тощо). Такий підхід дозволяє забезпечити інформування користувачів про зміни у системі та покращує зручність взаємодії без необхідності ручної реєстрації або відстеження статусу вручну.

4.1.2. Клієнтська частина

У процесі реалізації клієнтської частини вебзастосунку PostTrack було використано сучасні вебтехнології, що забезпечують зручність, продуктивність та масштабованість інтерфейсу. Основною метою є створення багаторольового застосунку з чіткою структурою сторінок для адміністратора, оператора та клієнта, з динамічною візуалізацією поштових процесів та інтеграцією з картографічними сервісами.

Для побудови вебінтерфейсу використано бібліотеку React, яка забезпечує компонентний підхід до створення інтерфейсів. React дозволяє динамічно змінювати вміст сторінки залежно від стану додатку без потреби в повному перезавантаженні. Завдяки використанню хуків (Hooks) та концепції однонапрямленого потоку даних, реалізовано чисту та зрозумілу структуру компонентів.

Усі сторінки вебзастосунку (включно з панеллю керування адміністратора, формою відправлення, історією відправлень клієнта) побудовані як окремі компоненти React, які взаємодіють із централізованим сховищем стану через RTK Query (Redux Toolkit).

Для організації навігації між сторінками в межах застосунку використано React Router. Це дозволяє реалізувати SPA-архітектуру (Single Page Application), за якої всі переходи між сторінками здійснюються без перезавантаження браузера. Кожна роль (адміністратор, оператор, клієнт) має доступ лише до визначених маршрутів, що узгоджується з логікою авторизації, реалізованої через токени.

Для обробки асинхронних запитів до серверного API застосовано Redux Toolkit з додатком RTK Query [7]. Це рішення дозволяє зручно управляти кешем даних, повторно використовувати відповіді API між компонентами, автоматично оновлювати інтерфейс після мутацій (створення, оновлення, видалення) і відокремити логіку роботи з сервером від компонента інтерфейсу.

Наприклад, всі запити на отримання відділень, створення посилок, оновлення статусу чи отримання профілю користувача реалізовано централізовано в postTrackApi.

Для інтерактивного відображення місцеперебування відділень та кур’єрів інтегровано бібліотеку Leaflet.js – популярний інструмент для роботи з OpenStreetMap.

Для побудови візуального оформлення інтерфейсу застосовано Tailwind CSS – утилітарну CSS-бібліотеку, що дозволяє стилізувати

компоненти без написання окремих CSS-файлів. Завдяки цьому забезпечено адаптивність інтерфейсу, швидке створення стилів і чистий код без надлишкових класів або конфліктів.

При вході користувача в систему отримується токен, який зберігається у localStorage та додається до кожного запиту через заголовки Authorization. Завдяки цьому вебінтерфейс підтримує автоматичну перевірку ролі, відображення доступних розділів меню та захист маршрутів.

Оператор має можливість створити нове доставлення через окрему форму, в якій вказуються email відправника та одержувача, обирається відділення призначення (через мапу) і кур’єр.

Таким чином, клієнтська частина побудована з використанням сучасного технологічного стека, орієнтованого на зручність взаємодії, швидкість рендерингу та зменшення складності підтримки коду. Розділення логіки, асинхронна обробка даних і інтерактивна мапа забезпечують ефективну роботу користувачів з різними ролями у системі.

4.2. Опис реалізованого користувацького інтерфейсу

Користувацький інтерфейс системи PostTrack спроєктований таким чином, щоб забезпечити зручну та інтуїтивно зрозумілу взаємодію для всіх типів користувачів: адміністратора, оператора та клієнта. Основна увага при розробці була зосереджена на простоті навігації, логічній структурі сторінок, відповідності функціоналу ролям користувачів, а також інтерактивній візуалізації логістичних процесів.

Інтерфейс реалізовано з використанням сучасного підходу до побудови односторінкових застосунків (SPA), що забезпечує динамічне оновлення вмісту без перезавантаження сторінки. Структура вебінтерфейсу включає набір спеціалізованих сторінок для кожної ролі користувача, кожна з яких виконує певну функціональну задачу в системі.

До основних сторінок клієнтської частини належать:

- сторінка авторизації (вхід до системи);

- панель адміністратора (перегляд відділень на мапі та у списку);
- сторінка інформації про відділення (з переглядом операторів);
- форма створення нового відділення;
- форма додавання нового оператора;
- форма створення нового кур’єра;
- сторінка оператора для створення відправлень;
- сторінка оператора для перегляду вхідних відправлень;
- сторінка клієнта для перегляду відправлень із інтеграцією мапи.

У наступних підрозділах детально розглянуто функціональне призначення, структуру та інтерфейс кожної з цих сторінок.

Сторінка авторизації є початковою точкою входу в систему PostTrack для всіх типів користувачів: адміністратора, оператора, кур’єра або клієнта. Її основне призначення – забезпечити безпечну і зручну перевірку облікових даних користувача, на основі яких система надає доступ до відповідного функціоналу залежно від ролі.

Інтерфейс сторінки складається з двох основних текстових полів для введення електронної пошти та пароля, а також кнопки «Login». Після введення даних і натискання кнопки, відбувається посилання запиту до серверної частини API для перевірки правильності введених облікових даних. У разі успішної авторизації система отримує з сервера токен доступу та роль користувача, після чого автоматично перенаправляє його на відповідну сторінку: панель адміністратора, інтерфейс оператора або клієнтську панель.

Таким чином, ця сторінка виконує ключову функцію – контроль доступу до системи. Вона реалізована з урахуванням рольової моделі безпеки та забезпечує початкове підключення користувача до особистого функціоналу, який відповідає його статусу в системі.

Візуальний інтерфейс наведений на рис. 4.1.

Рис. 4.1. Сторінка авторизації

Сторінка додавання нового оператора доступна лише для користувачів із роллю адміністратора та призначена для створення нового облікового запису оператора, який буде закріплений за конкретним відділенням. Вона є частиною адміністративного функціоналу з управління персоналом логістичних підрозділів.

Інтерфейс сторінки складається з простої форми, яка включає два поля для введення електронної пошти та пароля нового оператора. Дані з форми відправляються до серверного API разом із інформацією про ID відділення, до якого призначається оператор. У разі успішного створення облікового запису, користувача (тобто адміністратора) автоматично перенаправляє назад на сторінку інформації про це відділення, де новий оператор уже з'являється в списку.

Сторінка також включає обробку помилок та візуальний індикатор процесу створення. Якщо під час реєстрації сталася помилка (наприклад, введено вже існуючий email), інтерфейс виводить відповідне повідомлення. Таким чином, дана сторінка дозволяє швидко і просто здійснювати управління кадровим складом кожного відділення.

Візуальний інтерфейс наведений на рис. 4.2.

Рис. 4.2. Сторінка додавання оператора

Сторінка перегляду відправлень клієнта призначена для авторизованих користувачів із роллю клієнта й надає повну інформацію про всі активні, доставлені та отримані відправлення. Вона виконує роль особистого кабінету, де користувач може зручно відстежувати історію своїх посилок і візуалізувати місце перебування кур’єра на карті в режимі реального часу.

Головною особливістю сторінки є інтерактивна мапа, реалізована за допомогою бібліотеки Leaflet. На мапі відображаються всі активні кур’єри, які перевозять відправлення цього користувача. Клікнувши на маркер кур’єра, користувач бачить спливаючу підказку з інформацією про посилку, включно зі статусом, відділеннями відправника та одержувача, а також адресами електронної пошти обох сторін. Крім цього, при виборі конкретної посилки на мапі додатково позначаються відділення відправлення і призначення.

Візуальний інтерфейс наведений на рис. 4.3.

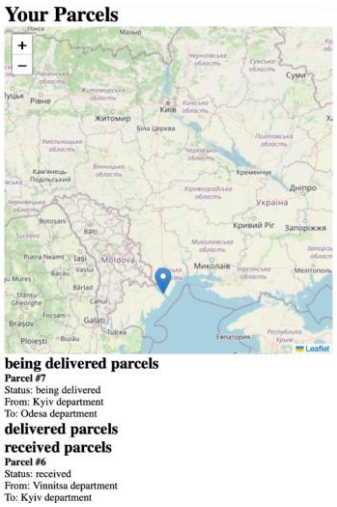


Рис. 4.3. Сторінка перегляду відправлень клієнта

Нижче мапи розміщено список усіх відправлень, згрупованих за статусами: «в дорозі», «доставлено» та «отримано». Кожен блок містить перелік відповідних посилок із короткою інформацією, яка дублюється на мапі. Це дозволяє користувачеві швидко зорієнтуватися у поточному стані кожного відправлення, спростивши процес контролю та взаємодії з системою.

Завдяки поєднанню карти й спискового представлення, сторінка забезпечує клієнту повну прозорість логістичних операцій у зручному та візуально зрозумілому форматі.

Сторінка створення кур’єра надається адміністраторам і призначена для реєстрації нових користувачів із роллю кур’єра. Вона реалізує базову, але необхідну функціональність управління персоналом, що відповідає за фізичне доставлення поштових відправлень.

Інтерфейс сторінки максимально спрощений: форма включає одне поле для введення електронної адреси та кнопку підтвердження створення. Пароль для нового кур’єра встановлюється автоматично на етапі відправлення запиту, що дозволяє пришвидшити процес реєстрації. Після успішного створення облікового запису, користувача перенаправляє на головну панель адміністратора, де він може продовжити керування іншими елементами системи.

Візуальний інтерфейс наведений на рис. 4.4.

Рис. 4.4. Сторінка створення кур’єра

Сторінка створення відділення призначена для адміністраторів системи і дозволяє створювати нові поштові відділення з інтерактивним вибором їхнього місцеперебування на мапі. Основна мета – надати інтуїтивно зрозумілий інтерфейс для реєстрації нових структурних одиниць логістичної мережі без необхідності вводити координати вручну.

Користувач вводить назву відділення у відповідному текстовому полі, а координати визначає, клацнувши на потрібне місце на інтерактивній карті (реалізованій за допомогою Leaflet.js). Одразу після вибору координат на мапі з’являється маркер із підказкою, що позначає майбутнє розташування відділення. На мапі також візуалізуються вже існуючі відділення, що дозволяє адміністраторам уникати дублювань та приймати більш обґрунтовані рішення.

Візуальний інтерфейс наведений на рис. 4.5.

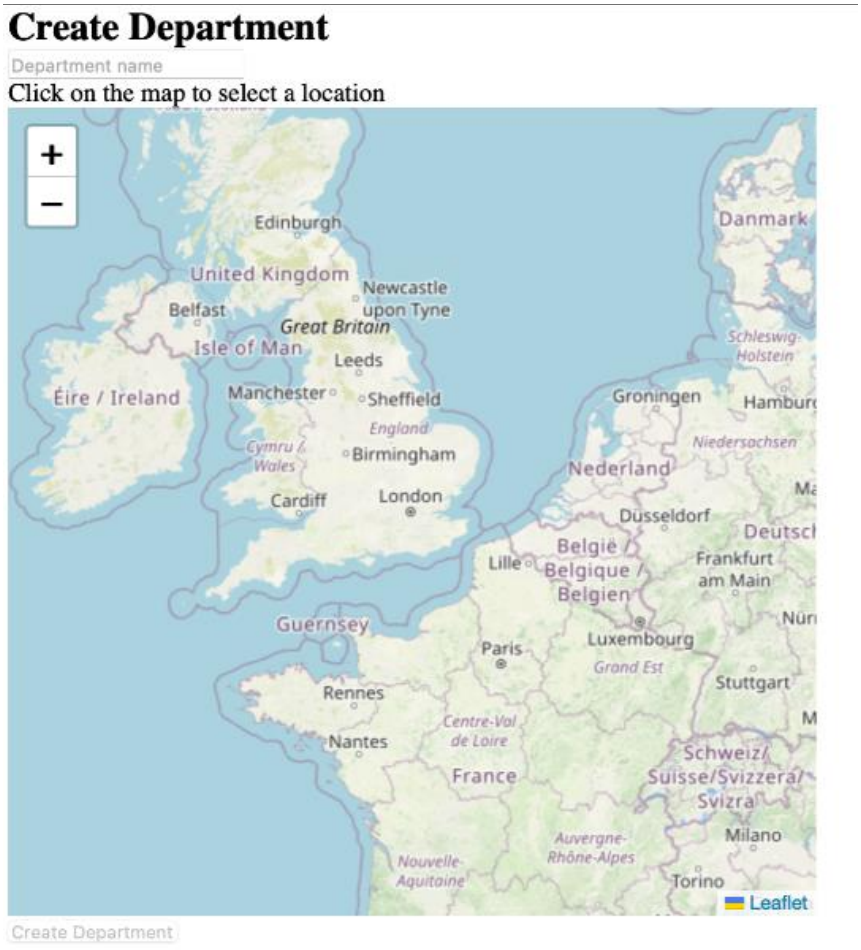


Рис. 4.5. Сторінка створення відділення

Після заповнення форми й підтвердження, дані відправляються на сервер, де створюється новий запис у базі даних. У разі успішної операції користувача повертає на адміністративну панель. Такий підхід поєднує зручність введення інформації з географічною наочністю, що є особливо важливим для системи, пов’язаної з логістикою і просторовими даними.

Сторінка перегляду інформації про відділення доступна адміністраторам і призначена для перегляду детальної інформації про конкретне поштове відділення, а також для керування його персоналом. Вона реалізує функціональність, необхідну для адміністрування структури логістичної мережі.

У верхній частині сторінки відображається назва відділення та його географічна локація. Основна частина інтерфейсу зосереджена на виведенні списку працівників, які закріплені за цим відділенням, зокрема операторів. Кожен оператор виводиться у вигляді окремого елемента списку з його електронною адресою. Якщо у відділенні ще немає жодного оператора, система повідомляє про це користувача у вигляді інформативного тексту.

У нижній частині сторінки розміщено інтерактивну кнопку у вигляді іконки «+», яка дозволяє адміністратору швидко перейти до форми додавання нового оператора. Таким чином, сторінка не лише відображає актуальну інформацію, але й інтегрує інструменти для оперативного управління складом співробітників відділення.

Візуальний інтерфейс наведений на рис. 4.6.

Рис. 4.6. Сторінка перегляду інформації про відділення

Головна сторінка адміністратора – це центральний інструмент управління поштовими відділеннями в системі PostTrack. Вона поєднує візуальне представлення об’єктів на мапі з текстовим списком для ефективного перегляду, контролю та навігації.

У верхній частині сторінки розміщена інтерактивна карта, реалізована за допомогою Leaflet.js, яка відображає всі наявні відділення у вигляді маркерів. Кожен маркер містить підказку з назвою та координатами відповідного підрозділу. При натисканні на маркер, користувача автоматично перенаправляє на сторінку з детальною інформацією про обране відділення.

Візуальний інтерфейс наведений на рис. 4.7.

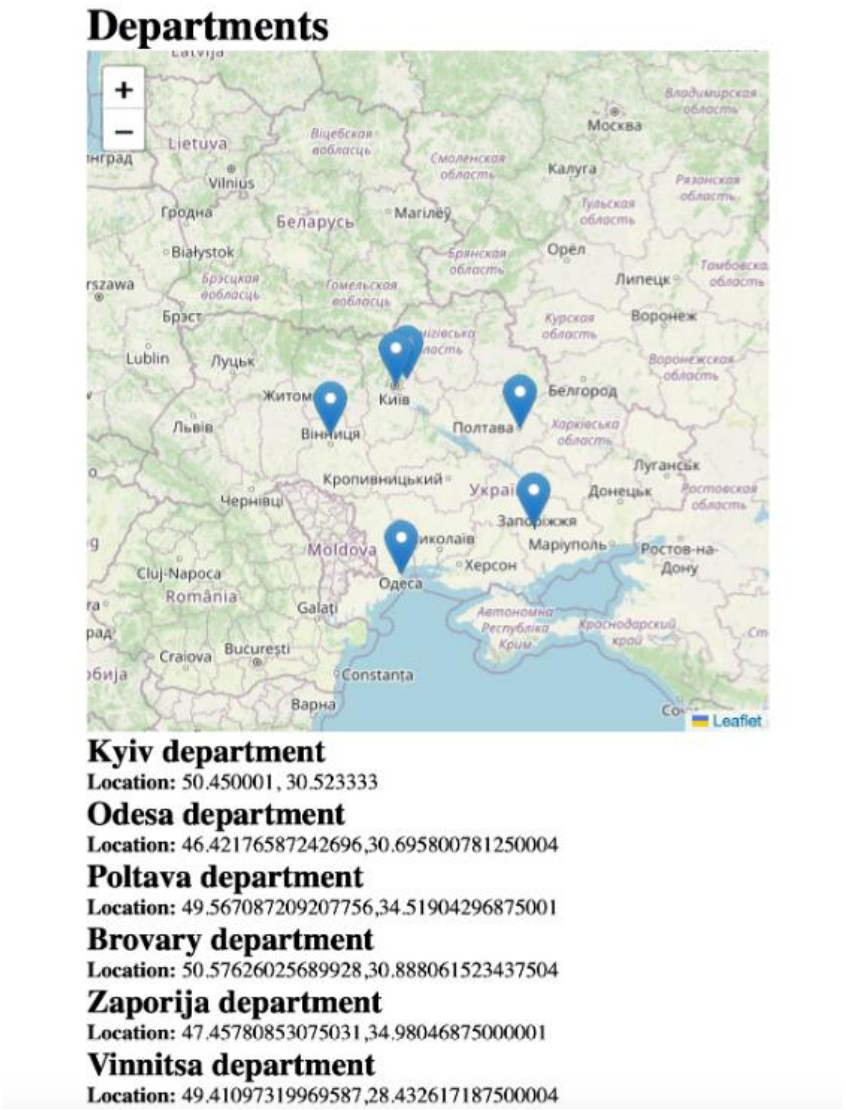


Рис. 4.7. Сторінка відділень

Під картою представлено текстовий список усіх зареєстрованих відділень із зазначенням їх назв і місцеперебувань. Елементи списку інтерактивні – натискання на будь-який з них також веде до відповідної сторінки з повною інформацією.

У нижній частині сторінки розміщено кнопку з іконкою «+», яка дозволяє створити нове відділення, та окрему кнопку для додавання кур'єра. Така структура дозволяє адміністратору зручно керувати мережею відділень та персоналом без потреби перемикатися між численними розділами, забезпечуючи цілісне управління логістичною інфраструктурою.

Сторінка перегляду вхідних відправлень призначена для користувачів із роллю оператора, які працюють у певному відділенні, і дає змогу переглядати всі вхідні відправлення, що мають бути оброблені в межах цього підрозділу. Вона реалізує важливу частину внутрішньої логістики – контроль і оновлення статусів доставок. Візуальний інтерфейс наведений на рис. 4.8.

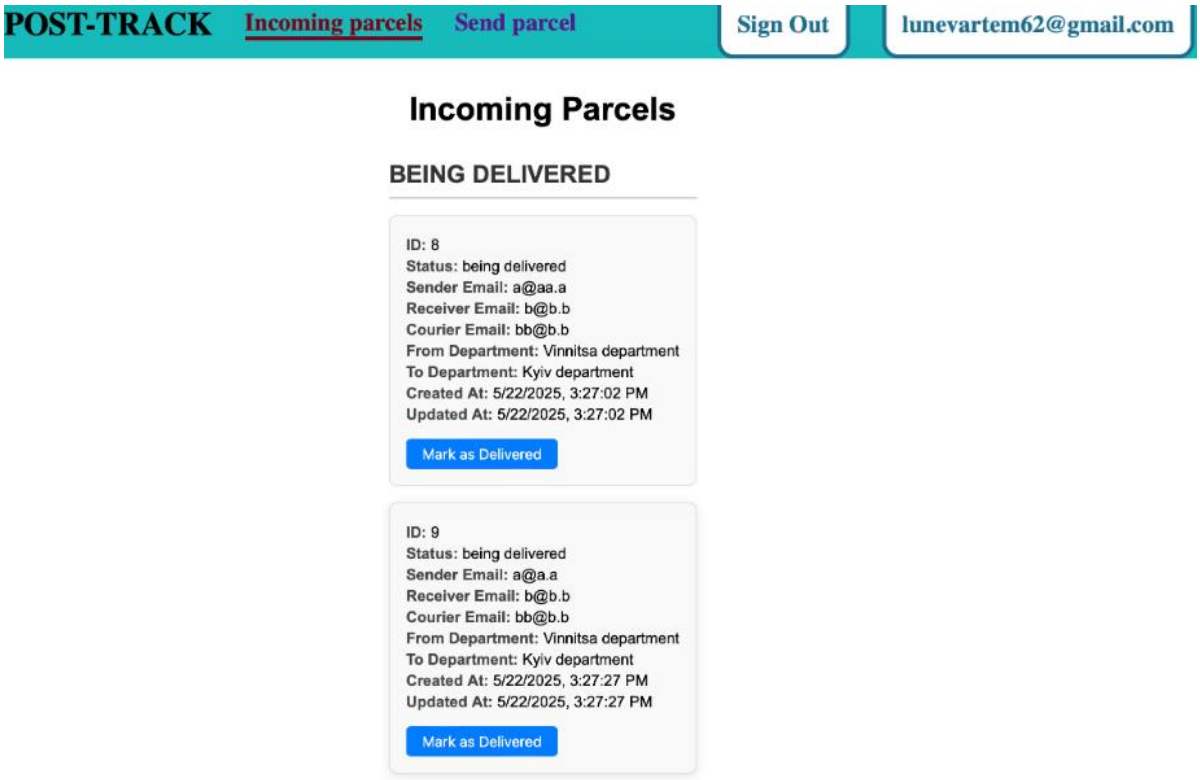


Рис. 4.8. Вхідні посилки

Система автоматично отримує список усіх посилок, що призначені до відділення оператора, та групує їх за статусом: «в дорозі», «доставлено» та «отримано». Для кожної посилки відображається детальна інформація: ID, статус, електронні адреси відправника, одержувача й кур’єра, а також дати створення й останнього оновлення.

Ключова функціональність цієї сторінки полягає у можливості змінювати статус посилки. Якщо посилка перебуває «в дорозі», оператор може натиснути кнопку для позначення її як «доставлена». Далі, після прибуття й підтвердження, можна оновити статус на «отримано». Таким чином, сторінка служить як інструмент контролю за рухом відправлень, так і як точка завершення процесу доставлення в межах відділення. Це значно підвищує прозорість логістичних операцій і спрощує взаємодію з системою.

Сторінка OperatorSendParcel призначена для оператора системи доставлення, який може відправляти посилки між відділеннями компанії. На сторінці відображається інтерактивна карта з позначеннями всіх доступних відділень, де кожне відділення позначене маркером. Відділення, до якого належить оператор (поточне відділення профілю), показується сірим маркером без можливості вибору, щоб уникнути помилкового вибору власного відділення як отримувача. Вибране відділення підсвічується червоним маркером. Крім того, нижче карти є список відділень, де оператор також може вибрати потрібне відділення отримувача. Власне відділення у списку позначене і неактивне для вибору.

Користувач заповнює форму, вказуючи електронну пошту відправника й отримувача, а також вибирає кур’єра зі списку. Після вибору усіх необхідних полів і відділення отримувача, оператор може надіслати посилку, натиснувши кнопку «Send Parcel». Під час відправлення кнопка блокується і змінює текст на «Sending…» для індикації процесу. Якщо якась із необхідних інформацій не заповнена, користувач отримає повідомлення з проханням заповнити всі поля. У випадку помилки з’являється відповідне повідомлення, а помилка логуються в консолі для подальшого аналізу.

Візуальний інтерфейс наведений на рис. 4.9.

Рис. 4.9. Відправлення посилки

З технічної сторони, для взаємодії зі стороною сервера використовується RTK Query: завантажуються списки відділень, кур’єрів,

дані профілю користувача, а також виконується мутація для відправлення посилки. Для карти застосовується бібліотека React Leaflet з кастомними іконками маркерів. Стилі оформлені у SCSS-файлі, що імпортується в компонент.

Таким чином, ця сторінка є зручним і надійним інструментом для оператора, що дозволяє легко обрати отримувача, призначити кур'єра, ввести необхідні дані та швидко відправити посилку, мінімізуючи помилки завдяки інтерактивній візуалізації та валідації форми.

4.3. Особливості проведеного тестування

Для перевірки коректності роботи вебзастосунку PostTrack було проведено модульне, інтеграційне та ручне функціональне тестування основних сценаріїв використання системи. Тестування охопило ключові ролі: адміністратора, оператора та клієнта, а також процеси створення, обробки та перегляду поштових відправлень.

Тестування проводилося у тестовому середовищі з використанням реальних API-запитів до серверної частини, що дозволило перевірити повноцінну взаємодію між клієнтським і серверним застосунками. Особливу увагу приділено перевірці бізнес-логіки, прав доступу, коректності валідації форм та відображення геоданих на мапі. Також оцінювалась стабільність застосунку при послідовному виконанні кількох дій та реакція інтерфейсу на помилки введення. Крім того, було проведено оцінку стабільності роботи застосунку при послідовному виконанні низки дій користувачем, включаючи навігацію, зміну налаштувань, надсилання запитів тощо. Тестувалася також реакція інтерфейсу на помилки введення – чи відображаються повідомлення, чи не порушується робота додатку внаслідок некоректних дій користувача.

Результати тестування зведено у таблицю 4.1.

Таблиця 4.1

Таблиця тестування

№	Назва тесту	Передумова	Сценарій дій	Очікуваний результат
1	Авторизація користувача	Користувач зареєстрований у системі	Ввести email та пароль, натиснути «Login»	Користувача перенаправлено на відповідну панель
2	Додавання нового відділення	Авторизований адміністратор	Ввести назву відділення, обрати локацію на мапі, натиснути «Create»	Відділення створено, відображається на карті
3	Створення оператора	Відкрито сторінку додавання оператора, задано ID відділення	Ввести email та пароль, натиснути «Add Operator»	Оператор створений і з’являється у списку відділення
4	Створення кур’єра	Авторизований адміністратор на відповідній сторінці	Ввести email, натиснути «Create Courier»	Кур’єр успішно зареєстрований
5	Відправлення посилки	Оператор авторизований, є заповнена форма і вибране відділення-призначення	Заповнити email відправника, одержувача, обрати кур’єра, натиснути «Send Parcel»	Посилка зареєстрована, клієнтам надіслано сповіщення
6	Відображення мапи з відправленнями клієнта	Авторизований клієнт	Перейти на сторінку відправлень	Виводиться карта з маркерами кур’єрів, можна побачити деталі посилки

Продовження табл. 4.1

7	Оновлення статусу посилки	Оператор бачить вхідну посилку зі статусом «being delivered»	Натиснути «Mark as Delivered» або «Mark as Received»	Статус змінено відповідно до натиснутої кнопки
8	Відображення операторів відділення	Адміністратор відкрив сторінку певного відділення	Перейти на сторінку → побачити список працівників	Список операторів відображено коректно
9	Обробка некоректної авторизації	Користувач вводить неправильний пароль або email	Ввести невірні дані → натиснути «Login»	Виведено повідомлення про помилку, вхід не дозволено
10	Валідація полів при реєстрації кур’єра	Введено порожнє поле email	Натиснути «Create Courier»	Виведено повідомлення про обов’язковість заповнення

4.4. Висновки до розділу

У рамках даного розділу надано особливості реалізації клієнтської та серверної частин вебзастосунку, описано створений користувацький інтерфейс і проведено тестування.

ВИСНОВКИ

Метою даного дипломного проєкту було створення вебзастосунку для автоматизації процесу обліку поштових відділень та спрощення відправки посилок між ними. Результатом роботи став програмний продукт, доступний у мережі Інтернет, який дозволяє операторам легко керувати інформацією про відділення, вибирати кур’єрів та здійснювати відправлення посилок.

У процесі виконання проєкту було проаналізовано предметну область логістики та поштових послуг, визначено ключові проблеми, що виникають у процесі обробки відправлень, а також досліджено існуючі рішення на ринку. Зібрано вимоги до системи, обрано технологічний стек, до якого увійшли мова програмування JavaScript, бібліотека React.js для розробки користувацького інтерфейсу, а також RTK Query для роботи з серверними API. Вебзастосунок побудовано за принципами клієнт-серверної архітектури.

Реалізована система забезпечує зручний інтерфейс для перегляду розташування поштових відділень на інтерактивній карті, вибору відділення отримувача, а також вибору кур’єра для доставлення посилки. Впроваджено валідацію даних у формі відправлення, яка дозволяє уникнути помилок при введенні інформації. Тестування вебзастосунку підтвердило його стабільність та відповідність технічним вимогам.

Використання розробленого вебзастосунка дозволить значно підвищити ефективність роботи операторів поштових відділень, спростити процес оформлення та відправлення посилок, а також знизити кількість помилок завдяки інтерактивним інструментам і контролю введених даних.

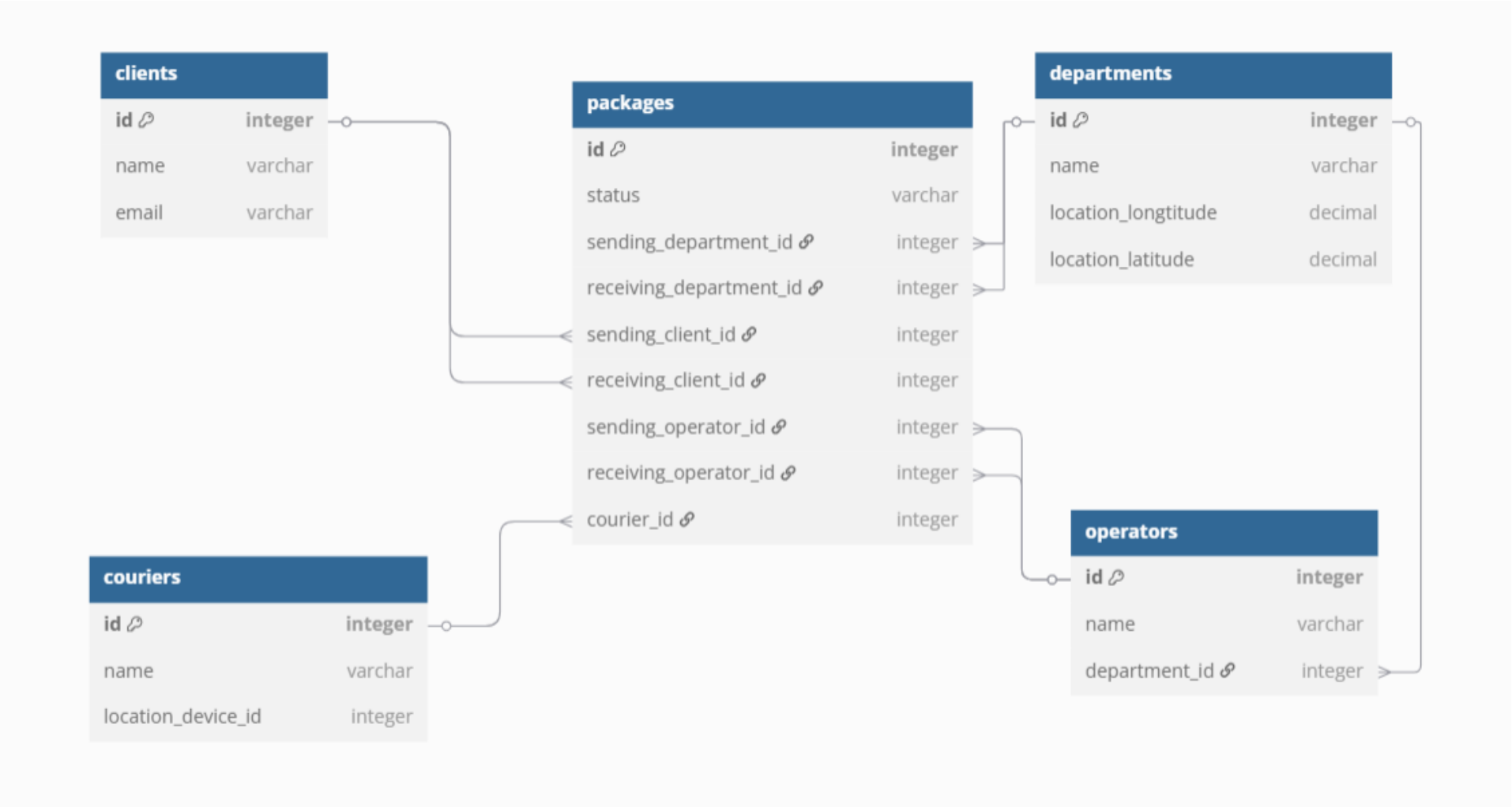
СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Onfleet [Електронний ресурс]. – Режим доступу: https://onfleet.com. – Дата доступу: 28.03.2025 р.
2. Track-POD [Електронний ресурс]. – Режим доступу: https://www.track-pod.com. – Дата доступу: 28.03.2025 р.
3. Detrack [Електронний ресурс]. – Режим доступу: https://www.detrack.com. – Дата доступу: 28.03.2025 р.
4. Node.js Documentation [Електронний ресурс]. – Режим доступу: https://nodejs.org/en/docs. – Дата доступу: 28.03.2025 р.
5. Express.js Documentation [Електронний ресурс]. – Режим доступу: https://expressjs.com. – Дата доступу: 28.03.2025 р.
6. React Documentation [Електронний ресурс]. – Режим доступу: https://react.dev/. – Дата доступу: 28.03.2025 р.
7. Redux Toolkit Documentation [Електронний ресурс]. – Режим доступу: https://redux-toolkit.js.org/. – Дата доступу: 28.03.2025 р.
8. React Router Documentation [Електронний ресурс]. – Режим доступу: https://reactrouter.com/. – Дата доступу: 28.03.2025 р.
9. PostgreSQL Documentation [Електронний ресурс]. – Режим доступу: https://www.postgresql.org/docs/. – Дата доступу: 28.03.2025 р.
10. Knex.js Documentation [Електронний ресурс]. – Режим доступу: https://knexjs.org/. – Дата доступу: 28.03.2025 р.
11. TypeScript Handbook [Електронний ресурс]. – Режим доступу: https://www.typescriptlang.org/docs/. – Дата доступу: 28.03.2025 р.
12. Kotlin Language Documentation [Електронний ресурс]. – Режим доступу: https://kotlinlang.org/docs/home.html. – Дата доступу: 28.03.2025 р.

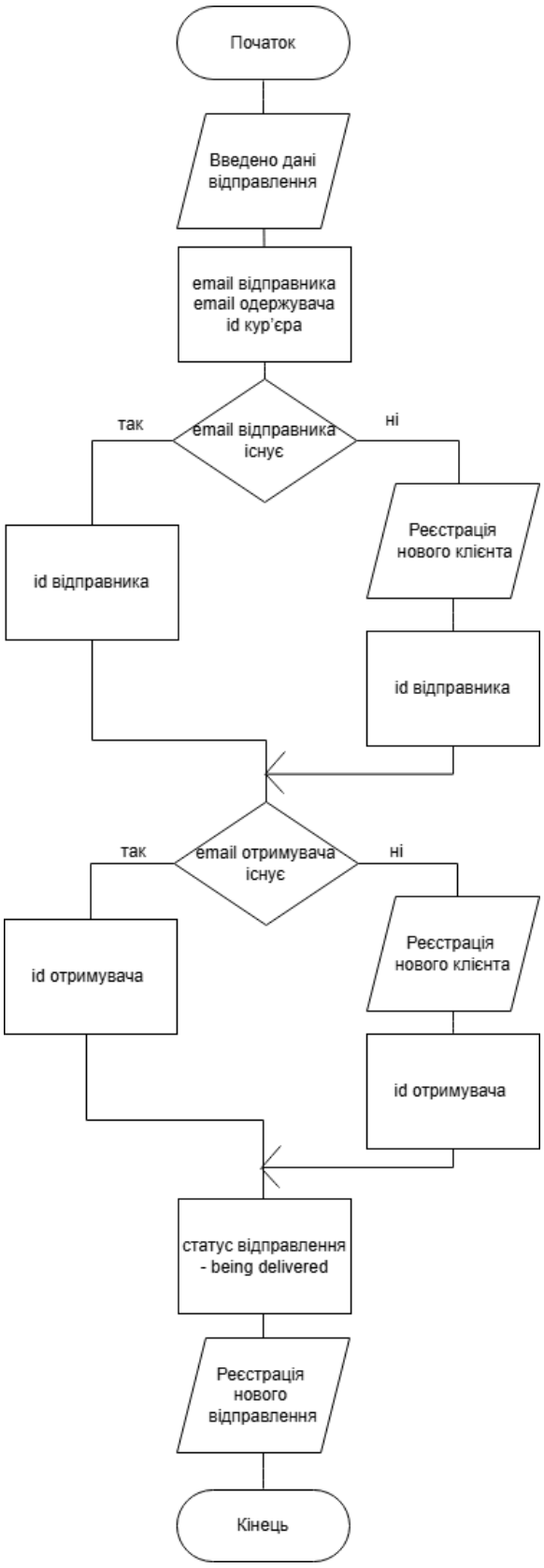
ДОДАТКИ

Додаток 1

Копії графічних матеріалів



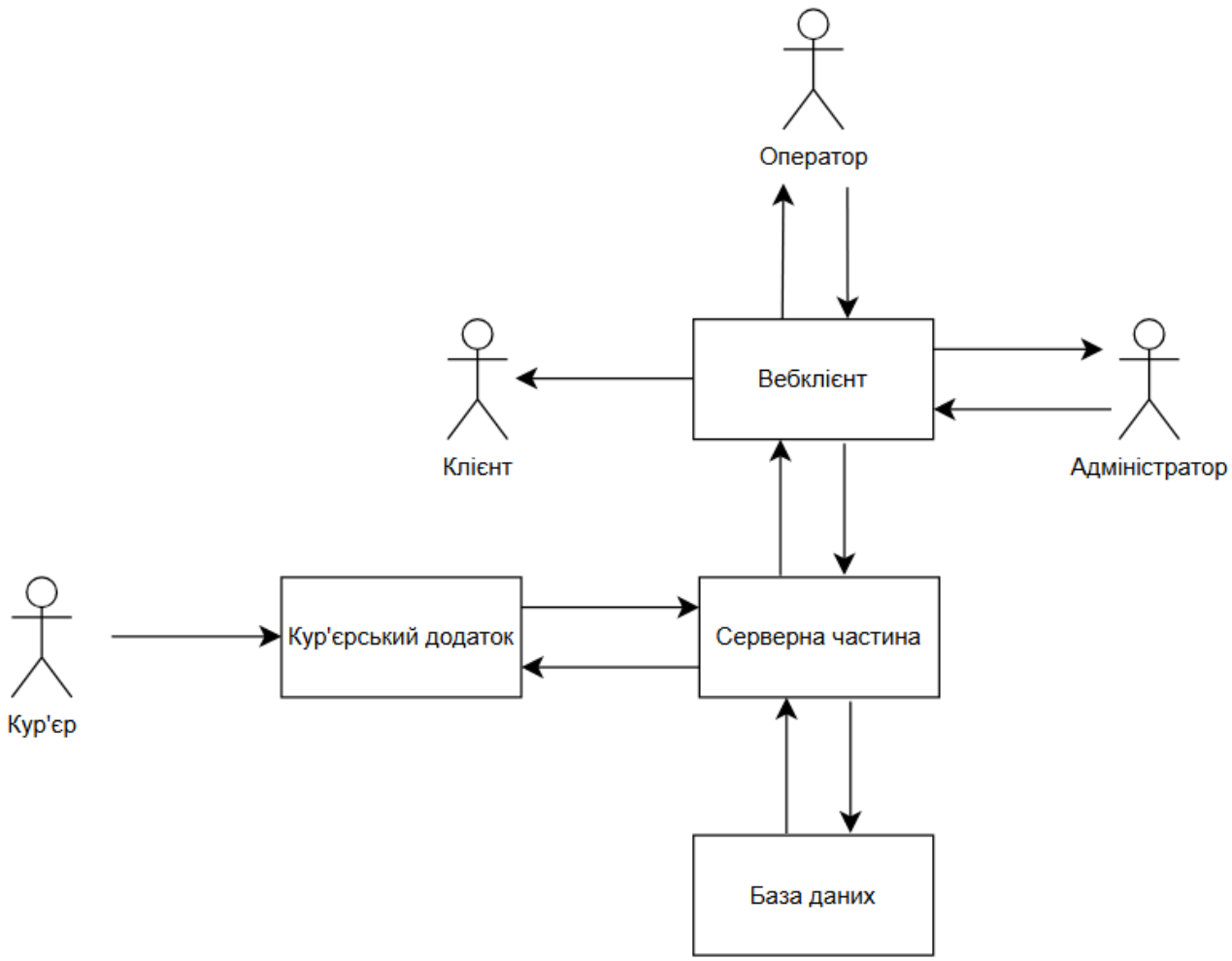
ДП.045440-07-99. Вебзастосунок для обліку та реєстрації поштових відділень. Структура бази даних. ERD-діаграма



ДП.045440-06-99. Вебзастосунок для обліку та реєстрації поштових відділень. Реєстрація відправлення. Схема алгоритму

Дерево проблем потенційних користувачів системи

Лунев А.В., група КП-11



Узагальнена структура вебзастосунку

Лунев А.В., група КП-11

Додаток 2

Лістинг програми

```
import bcrypt from "bcrypt";
import jwt from "jsonwebtoken";
import { knexConnection } from "../config/config";
import { v4 } from "uuid";
import { sendEmail } from "../emailService";

const JWT_SECRET = "your-secret-key";

export const register = async (req, res) => {
  const { email, password, role, device_id, department_id } = req.body;

  try {
    const existing = await knexConnection("profiles").where({ email
}).first();
    if (existing) return res.status(400).json({ error: "Username already
exists" });

    const hashed = await bcrypt.hash(password, 10);
    const [user] = await knexConnection("profiles")
      .insert({
        id: v4(),
        email,
        password: hashed,
        role,
        department_id,
      })
      .returning("*");

    if (user) {
      if (user.role == "courier")
        await sendEmail({
          to: email,
          subject: 'Registration',
          text: `Welcome to PostTrack!\n\nYour registration was
successful.\nRole: ${role}\nID: ${user.id}\n\nYou can now log in and start
using the system.`,
        });
      else
        await sendEmail({
          to: email,
          subject: 'Registration',
          text: `Welcome to PostTrack!\n\nYour registration was
successful.\nRole: ${role}\nPassword: ${password}\n\nYou can now log in and
start using the system.`,
        });
    }

    res.status(201).json({ id: user.id, email: user.email, role: user.role
});
  } catch (error) {
    console.error('Registration error:', error);
    res.status(500).json({ error: 'Something went wrong during
registration.' });
  }
};



export const login = async (req, res) => {
  const { email, password } = req.body;

  const user = await knexConnection("profiles").where({ email }).first();
  if (!user) return res.status(401).json({ error: "Invalid credentials" });
```

```
  const valid = await bcrypt.compare(password, user.password);
  if (!valid) return res.status(401).json({ error: "Invalid credentials"
});

  const token = jwt.sign({ id: user.id, email: user.email }, JWT_SECRET, {
    expiresIn: "1d",
  });

  res.json({ token, role: user.role });
};

export const getProfile = async (req, res) => {
    const user = await knexConnection('profiles').where({ id: req.user.id
}).first();

    if (!user) {
      return res.status(404).json({ message: 'User not found' });
    }

    res.json(user);
  };
import jwt from "jsonwebtoken";
import dotenv from "dotenv";
import { knexConnection } from "../config/config";

dotenv.config();

const JWT_SECRET = process.env.JWT_SECRET || "your-secret-key";

export const authMiddleware = (req, res, next) => {
  const authHeader = req.headers.authorization;
  if (!authHeader?.startsWith("Bearer ")) {
    return res.status(401).json({ error: "Token missing" });
  }

  const token = authHeader.split(" ")[1];
  try {
    const payload = jwt.verify(token, JWT_SECRET);
    req.user = payload;
    next();
  } catch (err) {
    res.status(401).json({ error: "Invalid token" });
  }
};

export const authorizeAdmin = async (req, res, next) => {
    const userId = req.user.id;

    const user = await knexConnection('profiles').where({ id: userId
}).first();
    if (!user || user.role !== 'ADMIN') {
      return res.status(403).json({ message: 'Forbidden: Admins only' });
    }

    next();
  };

  import type { Knex } from 'knex';

export async function up(knex: Knex): Promise<void> {
    return knex.schema.createTable('clients', function (table) {
        table.uuid('id').primary().unique().notNullable();
        table.string('email').notNullable();
    });
```

```
}

export async function down(knex: Knex): Promise<void> {
    return knex.schema.dropTable('clients');
} import type { Knex } from "knex";


export async function up(knex: Knex): Promise<void> {
    return knex.schema.alterTable('clients', function (table) {
        table.string('password').notNullable();
    });
}


export async function down(knex: Knex): Promise<void> {
    return knex.schema.alterTable('clients', function (table) {
        table.dropColumn('password');
    });
} import type { Knex } from "knex";


export async function up(knex: Knex): Promise<void> {
    return knex.schema.renameTable('clients', 'profiles');
}


export async function down(knex: Knex): Promise<void> {
    return knex.schema.renameTable('profiles', 'clients');
} import type { Knex } from "knex";


export async function up(knex: Knex): Promise<void> {
    return knex.schema.alterTable('profiles', function (table) {
        table.string('role').notNullable();
        table.string('device_id');
        table.string('department_id');
    });
}


export async function down(knex: Knex): Promise<void> {
    return knex.schema.alterTable('profiles', function (table) {
        table.dropColumn('role');
        table.dropColumn('device_id');
        table.dropColumn('department_id');
    });
} import type { Knex } from "knex";


export async function up(knex: Knex): Promise<void> {
    return knex.schema.alterTable('profiles', function (t) {
        t.unique('email');
      });
}


export async function down(knex: Knex): Promise<void> {
    return knex.schema.alterTable('profiles', function (t) {
        t.dropUnique(['email']);
      });
} import type { Knex } from "knex";


export async function up(knex) {
```

```
    await knex.schema.createTable('departments', (table) => {
      table.increments('id').primary();
      table.string('name').notNullable();
      table.string('location').notNullable();
      table.timestamps(true, true);
    });
  }

  export async function down(knex) {
    await knex.schema.dropTableIfExists('departments');
  } import type { Knex } from "knex";


export async function up(knex: Knex): Promise<void> {
    await knex.schema.createTable('parcels', (table) => {
        table.increments('id').primary();
        table.string('status').notNullable();
        table.integer('sending_department_id').notNullable();
        table.integer('receiving_department_id').notNullable();
        table.string('sending_client_id').notNullable();
        table.string('receiving_client_id').notNullable();
        table.string('courier_id').notNullable();
        table.timestamps(true, true);
      });
}


export async function down(knex: Knex): Promise<void> {
    await knex.schema.dropTableIfExists('parcels');
} import type { Knex } from "knex";


export async function up(knex: Knex): Promise<void> {
    await knex.schema.alterTable('parcels', (table) => {
      table.uuid('sending_client_id').alter().notNullable();
      table.uuid('receiving_client_id').alter().notNullable();
      table.uuid('courier_id').alter().notNullable();
    });
  }

  export async function down(knex: Knex): Promise<void> {
    await knex.schema.alterTable('parcels', (table) => {
      table.string('sending_client_id').alter();
      table.string('receiving_client_id').alter();
      table.string('courier_id').alter();
    });
  }
  import type { Knex } from "knex";


export async function up(knex: Knex): Promise<void> {
    return knex.schema.alterTable('profiles', function (table) {
        table.renameColumn('device_id', 'latitude');
        table.string('longtitude');
    });
}


export async function down(knex: Knex): Promise<void> {
} import type { Knex } from "knex";


export async function up(knex: Knex): Promise<void> {
    return knex.schema.alterTable('profiles', function (table) {
```

```
        table.renameColumn('longtitude', 'longitude');
    });
}


export async function down(knex: Knex): Promise<void> {
} import knex from 'knex';
import KNEX_CONFIG from './knex.config';

export const knexConnection = knex(KNEX_CONFIG); import { Knex } from
'knex';

const KNEX_CONFIG: Knex.Config = {
    client: 'pg',
    connection: {
        host: 'localhost',
        port: 5432,
        database: 'myappdb',
        user: 'postgres',
        password: 'password',
        timezone: 'UTC',
    },
    migrations: {
        tableName: 'knex_migrations',
        directory: './migrations',
    },
};

export default KNEX_CONFIG; import { Request, Response } from 'express';
import { knexConnection } from '../config/config';
import { v4 } from 'uuid';
import { sendEmail } from '../emailService';
import crypto from 'crypto';
import bcrypt from 'bcrypt';

export const createDepartment = async (req: Request, res: Response) => {
  const { name, location } = req.body;
  const [id] = await knexConnection('departments').insert({ name, location
}).returning('id');
  res.status(201).json({ id, name, location });
};

export const getAllDepartments = async (_: Request, res: Response) => {
  const departments = await knexConnection('departments').select();
  res.json(departments);
};

export const getDepartmentById = async (req: Request, res: Response) => {
  const { id } = req.params;
  const department = await knexConnection('departments').where({ id
}).first();
  if (!department) res.status(404).json({ message: 'Not found' });
  res.json(department);
};

export const updateDepartment = async (req: Request, res: Response) => {
  const { id } = req.params;
  const { name, location } = req.body;
  await knexConnection('departments').where({ id }).update({ name, location
});
  res.json({ message: 'Updated successfully' });
};

export const deleteDepartment = async (req: Request, res: Response) => {
```

```
  const { id } = req.params;
  await knexConnection('departments').where({ id }).del();
  res.json({ message: 'Deleted successfully' });
};

export const getOperatorsByDepartment = async (req: Request, res: Response)
=> {
  const { id: departmentId } = req.params;
  const operators = await knexConnection('profiles')
    .where({ department_id: departmentId, role: 'operator' })
    .select('id', 'email', 'role', 'department_id');
  res.json(operators);
};


export const getAllcouriers = async (req: Request, res: Response) => {
  const couriers = await knexConnection('profiles')
    .where({ role: 'courier' })
    .select('id', 'email', 'role', 'longitude', 'latitude');
  res.json(couriers);
};

export const sendParcel = async (req: Request, res: Response) => {
  const {
    sending_department_id,
    receiving_department_id,
    sending_client_email,
    receiving_client_email,
    courier_id
  } = req.body;

  try {
    const getOrRegisterClient = async (email: string): Promise<string> => {
      const existing = await knexConnection('profiles').where({ email
}).first();

      if (existing) {
        await sendEmail({
          to: email,
          subject: 'Parcel Sent Notification',
          text: `Hello!\n\nA parcel was sent involving your profile. Please
check your dashboard.`,
        });
        return existing.id;
      } else {
        const password = crypto.randomBytes(6).toString('hex');
        const hashed = await bcrypt.hash(password, 10);
        const [user] = await knexConnection('profiles')
          .insert({
            id: v4(),
            email,
            password: hashed,
            role: 'client',
            department_id: null,
          })
          .returning('*');

        await sendEmail({
          to: email,
          subject: 'Welcome to PostTrack!',
          text: `You have been automatically registered to
PostTrack!\n\nEmail: ${email}\nPassword: ${password}\nRole: CLIENT\n\n`,
        });
```

```
        return user.id;
      }
    };

    const sendingClientId = await
getOrRegisterClient(sending_client_email);
    const receivingClientId = await
getOrRegisterClient(receiving_client_email);

    const [id] = await knexConnection('parcels')
      .insert({
        status: 'being delivered',
        sending_department_id,
        receiving_department_id,
        sending_client_id: sendingClientId,
        receiving_client_id: receivingClientId,
        courier_id,
      })
      .returning('id');

    res.status(201).json({ id });
  } catch (error) {
    console.error('Error sending parcel:', error);
    res.status(500).json({ error: 'Failed to send parcel.' });
  }
};

export const getIncoming = async (req: Request, res: Response) => {
  const departmentId = parseInt(req.params.id, 10);
  const parcels = await knexConnection('parcels')
    .where({ receiving_department_id: departmentId })
    .join('profiles as sender', 'parcels.sending_client_id', 'sender.id')
    .join('profiles as receiver', 'parcels.receiving_client_id',
'receiver.id')
    .join('profiles as courier', 'parcels.courier_id', 'courier.id')
    .join('departments as from_dep', 'parcels.sending_department_id',
'from_dep.id')
    .join('departments as to_dep', 'parcels.receiving_department_id',
'to_dep.id')
    .select(
      'parcels.*',
      'sender.email as sender_email',
      'receiver.email as receiver_email',
      'courier.email as courier_email',
      'from_dep.name as sending_department_name',
      'to_dep.name as receiving_department_name'
    )
    .orderBy('parcels.created_at');

  res.json(parcels);
};

export const updateStatus = async (req: Request, res: Response) => {
  const { id } = req.params;
  const { status } = req.body;

  const validStatuses = ['being delivered', 'delivered', 'received'];
  if (!validStatuses.includes(status)) {
    res.status(400).json({ error: 'Invalid status' });
  }

  await knexConnection('parcels').where({ id }).update({ status });
  res.status(200).json({ message: 'Status updated' });
};
```

```
export const updateLocation = async (req: Request, res: Response) => {
  const { id, longitude, latitude } = req.body;
  await knexConnection('profiles').where({ id }).update({ longitude,
latitude });
  res.json({ message: 'Updated successfully' });
};

export const getClientParcels = async (req, res: Response) => {
  const sendingParcels = await knexConnection('parcels')
    .where({ sending_client_id: req.user.id })
    .join('profiles as sender', 'parcels.sending_client_id', 'sender.id')
    .join('profiles as receiver', 'parcels.receiving_client_id',
'receiver.id')
    .join('profiles as courier', 'parcels.courier_id', 'courier.id')
    .join('departments as from_dep', 'parcels.sending_department_id',
'from_dep.id')
    .join('departments as to_dep', 'parcels.receiving_department_id',
'to_dep.id')
    .select(
      'parcels.*',
      'sender.email as sender_email',
      'receiver.email as receiver_email',
      'courier.email as courier_email',
      'courier.longitude as courier_longitude',
      'courier.latitude as courier_latitude',
      'from_dep.name as sending_department_name',
      'to_dep.name as receiving_department_name'
    )
    .orderBy('parcels.created_at');
    const receivingParcels = await knexConnection('parcels')
    .where({ receiving_client_id: req.user.id })
    .join('profiles as sender', 'parcels.sending_client_id', 'sender.id')
    .join('profiles as receiver', 'parcels.receiving_client_id',
'receiver.id')
    .join('profiles as courier', 'parcels.courier_id', 'courier.id')
    .join('departments as from_dep', 'parcels.sending_department_id',
'from_dep.id')
    .join('departments as to_dep', 'parcels.receiving_department_id',
'to_dep.id')
    .select(
      'parcels.*',
      'sender.email as sender_email',
      'receiver.email as receiver_email',
      'courier.email as courier_email',
      'courier.longitude as courier_longitude',
      'courier.latitude as courier_latitude',
      'from_dep.name as sending_department_name',
      'to_dep.name as receiving_department_name'
    )
    .orderBy('parcels.created_at');

    const allParcels = [...sendingParcels, ...receivingParcels];

    res.json(allParcels);
}; import nodemailer from 'nodemailer';

const transporter = nodemailer.createTransport({
  service: 'gmail',
  auth: {
    user: 'lariluna2003@gmail.com',
    pass: 'gber sbtn vibz fynp',
  },
});
```

```
interface MailOptions {
  to: string;
  subject: string;
  text?: string;
}

export const sendEmail = async ({ to, subject, text }: MailOptions):
Promise<void> => {
  const mailOptions = {
    from: '"PostTrack"',
    to,
    subject,
    text,
  };

  try {
    const info = await transporter.sendMail(mailOptions);
    console.log('Email sent:', info.response);
  } catch (error) {
    console.error('Error sending email:', error);
    throw error;
  }
}; import express from 'express';
import cors from 'cors';
import routes from './routes';
const app = express();
const port = 3000;

app.use(cors({
  origin: 'http://localhost:5173',
  credentials: true
}))

app.use(express.json());
app.use("/api", routes);
app.listen(port, '0.0.0.0', () => {
  return console.log(`Express is listening at http://localhost:${port}`);
}); import express, { Request, Response } from "express";
import { register, login, getProfile } from "./auth/authController";
import { authMiddleware, authorizeAdmin } from "./auth/authMiddleware.js";
import { createDepartment, getAllDepartments, getDepartmentById,
updateDepartment, deleteDepartment, getOperatorsByDepartment,
getAllcouriers, sendParcel, getIncoming, updateStatus, updateLocation,
getClientParcels } from "./departments/departmentsController";

const router = express.Router();

router.post("/register", register);
router.post("/login", login);
router.get("/profile", authMiddleware, getProfile);
router.get("/departments/:id/operators", authMiddleware,authorizeAdmin,
getOperatorsByDepartment);
router.post('/departments', authMiddleware, createDepartment);
router.get('/departments', authMiddleware, getAllDepartments);
router.get('/departments/:id', authMiddleware, getDepartmentById);
router.put('/departments/:id', authMiddleware, authorizeAdmin,
updateDepartment);
router.delete('/departments/:id', authMiddleware, deleteDepartment);
router.get('/couriers', getAllcouriers);
router.post('/parcel', sendParcel)
router.get('/parcels/:id/incoming', authMiddleware, getIncoming)
router.patch('/parcels/:id/status', authMiddleware, updateStatus)
router.post('/location', updateLocation)
```

```
router.get('/parcel/client', authMiddleware, getClientParcels);

export default router; node_modules
dist
.env services:
  db:
    image: postgres:15
    container_name: myapp_postgres
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: password
      POSTGRES_DB: myappdb
    ports:
      - '5432:5432'
    volumes:
      - pgdata:/var/lib/postgresql/data

volumes:
  pgdata: import { defineConfig } from "eslint/config";
import js from "@eslint/js";
import globals from "globals";
import tseslint from "typescript-eslint";


  } import { FC } from "react";
import styles from './Button.module.scss'

export type ButtonProps = {
  label?: string
  onClick?: () => void
}

export const Button: FC<ButtonProps> = ({ label, onClick }) => {
  return (
    <button className={styles.button} onClick={onClick}>
      {label}
    </button >
  )
} export { Button } from './Button' import { FC } from "react";
import styles from './Header.module.scss';
import { useNavigate } from "react-router-dom";
import { useDispatch } from "react-redux";
import { useGetProfileQuery, useLogOutMutation, postTrackApi } from
"../../store/api";
import { HeaderLink } from "./HeaderLink";

const CLIENT_MENU_ITEMS = [
  { text: 'Incoming parcels', url: '/operator/incoming' },
  { text: 'Send parcel', url: '/operator/send' }
];

const ADMIN_MENU_ITEMS = [
  { text: 'Departments', url: '/admin' }
];

const OPERATOR_MENU_ITEMS = [
  { text: 'Incoming parcels', url: '/operator/incoming' },
  { text: 'Send parcel', url: '/operator/send' }
];

export const Header: FC = () => {
  const navigate = useNavigate();
```

```
  const dispatch = useDispatch();
  const { data: loggedProfile } = useGetProfileQuery();
  const [logOut] = useLogOutMutation();

  const onLogOut = async () => {
    await logOut();
    localStorage.removeItem('token');
    dispatch(postTrackApi.util.invalidateTags(['Profile']));
    navigate('/')
  };

  const getMenuItems = () => {
    switch (loggedProfile?.role.toLowerCase()) {
      case 'client':
        return CLIENT_MENU_ITEMS;
      case 'admin':
        return ADMIN_MENU_ITEMS;
      case 'operator':
        return OPERATOR_MENU_ITEMS;
      default:
        return [];
    }
  };

  return (
    <div className={styles.header}>
      <div className={styles.leftContainer}>
          <img
          src='/images/logo.jpg' width={80} height={60} alt="Post-Track
Logo" />
          <h1>POST-TRACK</h1>
        {getMenuItems().map(item => (
          <HeaderLink key={item.url} label={item.text} to={item.url} />
        ))}
      </div>

      {loggedProfile && (
        <div className={styles.rightContainer}>
          <div className={styles.login} onClick={onLogOut}>
            <h2>Sign Out</h2>
          </div>
          <div className={styles.signUp}>
            <h2>{loggedProfile.email}</h2>
          </div>
        </div>
      )}
    </div>
  );
}; import { FC } from "react"
import styles from './HeaderLink.module.scss'
import cn from 'classnames'
import { Link, useLocation } from "react-router-dom"

type HeaderLinkProps = {
  to: string
  label: string
}

export const HeaderLink: FC<HeaderLinkProps> = ({ to, label }) => {

  const location = useLocation()
  const isActive = location.pathname === to

  return (
```

```
    <Link to={to} className={cn(styles.link, isActive &&
styles.activeLink)}>
      <h2 className={styles.label}>{label}</h2>
      <div className={styles.line} />
    </Link>
  )
} .layout {
    background-image: url('../../../public/images/background.jpg');
    width: 100%;
    display: flex;
    flex-direction: column;
    margin: 0;
    padding-bottom: 0;
    height: 100%;
    box-sizing: border-box;
    background-attachment: scroll;
    background-size: cover;
    background-position: center;
    background-size: cover;
    background-position: center;
    background-repeat: no-repeat;
    height: 100vh;
    background-attachment: fixed;

    .scrollingContainer {
      width: 100%;
      display: flex;
      flex-direction: column;
      justify-content: space-between;
      z-index: 0;
      height: 100vh;
      overflow-y: auto;
      align-items: center;
    }
  } import React from 'react'
import { Outlet } from 'react-router-dom'
import { Header } from './Header/Header'
import styles from './Layout.module.scss'

export const Layout: React.FC = () => {
  return (
    <div className={styles.layout}>
      <Header />
      <div className={styles.scrollingContainer}>
        <Outlet />
      </div>
    </div>
  )
} import React from 'react';

type PlusButtonProps = {
  onClick: () => void;
};

const PlusButton: React.FC<PlusButtonProps> = ({ onClick }) => {
  return (
    <button
      onClick={onClick}
      className="w-10 h-10 rounded-full bg-blue-600 text-white text-2xl
flex items-center justify-center shadow-md hover:bg-blue-700 active:scale-
95 transition-transform duration-150"
    >
      +
    </button>
```

```
  );
};

export default PlusButton; // import { AccountRole, Profile } from
"../store/api/types/Profile.types"
// import { useGetLoggedProfileQuery } from 'store/api'

// type LoggedProfile = {
//   profile?: Profile
//   isAdmin: boolean
//   isClient: boolean,
//   isOperator: boolean,
//   isLoading: boolean
// }

// export const useLoggedProfile = (): LoggedProfile => {
//   const { data: profile, isLoading } = useGetLoggedProfileQuery()
//   const isClient = profile?.role === AccountRole.CLIENT
//   const isAdmin = profile?.role === AccountRole.ADMIN
//   const isOperator = profile?.role === AccountRole.OPERATOR

//   return {
//     profile,
//     isClient,
//     isAdmin,
//     isOperator,
//     isLoading,
//   }
// } import { createContext, ReactNode, useContext } from 'react'
import { useGetProfileQuery } from '../store/api'

type AuthenticationContextData = {
  isAuthenticated: boolean
}

const AuthenticationContext = createContext<AuthenticationContextData>({
  isAuthenticated: false,
})

type AuthProviderProps = {
  children: ReactNode
}

export const AuthenticationProvider = ({ children }: AuthProviderProps) =>
{
  const { data: profile, isLoading } = useGetProfileQuery()
  return (
    <AuthenticationContext.Provider
      value={{
        isAuthenticated: Boolean(profile),
      }}
    >
      {!isLoading && children}
    </AuthenticationContext.Provider>
  )
}

export const useAuthentication = () => useContext(AuthenticationContext)
import { FC, useState } from "react";
import { TextField } from "../../components/TextField";
import styles from './Login.module.scss'
import { Button } from "../../components/Button";
import { AccountRole, LoginBody } from
"../../store/api/types/Profile.types";
```

```
import { useNavigate } from "react-router-dom";
import { postTrackApi, useLoginMutation } from "../../store/api";
import { useDispatch } from "react-redux";

export const Login: FC = () => {
  const navigate = useNavigate();
  const [login] = useLoginMutation()
  const dispatch = useDispatch()

  const [profile, setProfile] = useState<LoginBody>({
    email: '',
    password: '',
  });

  const handleChange = (field: keyof LoginBody, value: string) => {
    setProfile(prev => ({
      ...prev,
      [field]: value
    }));
  };

  const handleClick = async () => {
    const response = await login(profile)
    console.log(response.data)
    if (!response.error) {
      dispatch(postTrackApi.util.invalidateTags(['Profile']))
      if(response.data.role.toLowerCase() == AccountRole.ADMIN)
        navigate('/admin')
      if(response.data.role.toLowerCase() == AccountRole.OPERATOR)
        navigate('/operator/send')
      if(response.data.role.toLowerCase() == AccountRole.CLIENT)
        navigate('/client')
    }
  };
  return (
    <div className={styles.login}>
      <TextField
        label="Email"
        onChange={(value) => handleChange('email', value)}
      />
      <TextField
        label="Password"
        type="password"
        onChange={(value) => handleChange('password', value)}
      />
      <Button label="Login" onClick={() => handleClick()} />
    </div>
  )
} import React, { useState } from 'react';
import { useParams, useNavigate } from 'react-router-dom';
import { useRegisterMutation } from '../store/api';

const AddOperator: React.FC = () => {
  const { id } = useParams<{ id: string }>();
  const navigate = useNavigate();
  const [createOperator, { isLoading, error }] = useRegisterMutation();

  const [form, setForm] = useState({
    email: '',
    password: '',
  });

  const handleChange = (e: React.ChangeEvent<HTMLInputElement>) => {
    setForm({ ...form, [e.target.name]: e.target.value });
```

```
  };

  const handleSubmit = async (e: React.FormEvent) => {
    e.preventDefault();
    if (!id) return;

    try {
      await createOperator({ ...form, role: 'operator', department_id: id
}).unwrap();
      navigate(`/admin/${id}`);
    } catch (err) {
      console.error('Failed to create operator:', err);
    }
  };

  return (
    <div className="max-w-md mx-auto p-6 bg-white shadow rounded-xl space-
y-6">
      <h2 className="text-2xl font-semibold mb-4">Add New Operator</h2>

      <form onSubmit={handleSubmit} className="space-y-4">

        <input
          name="email"
          type="email"
          placeholder="Email"
          value={form.email}
          onChange={handleChange}
          className="w-full p-2 border rounded"
          required
        />

        <input
          name="password"
          type="password"
          placeholder="Password"
          value={form.password}
          onChange={handleChange}
          className="w-full p-2 border rounded"
          required
        />

        <button
          type="submit"
          disabled={isLoading}
          className="w-full bg-blue-600 text-white py-2 rounded hover:bg-
blue-700 transition"
        >
          {isLoading ? 'Creating...' : 'Add Operator'}
        </button>

        {error && (
          <p className="text-red-500 text-sm mt-2">Failed to add operator.
Try again.</p>
        )}
      </form>
    </div>
  );
};

export default AddOperator; import React, { useState } from 'react';
import { MapContainer, TileLayer, Marker, Popup, Tooltip } from 'react-
leaflet';
import L from 'leaflet';
```

```
import { useGetDepartmentsQuery, useGetClientParcelsQuery } from
'../store/api';

const defaultIcon = new L.Icon({
  iconUrl: 'https://unpkg.com/leaflet@1.9.3/dist/images/marker-icon.png',
  iconSize: [25, 41],
  iconAnchor: [12, 41],
});

const fromIcon = new L.Icon({
    iconUrl: 'https://raw.githubusercontent.com/pointhi/leaflet-color-
markers/master/img/marker-icon-red.png',
    shadowUrl: 'https://unpkg.com/leaflet@1.9.3/dist/images/marker-
shadow.png',
    iconSize: [25, 41],
    iconAnchor: [12, 41],
    popupAnchor: [1, -34],
    shadowSize: [41, 41],
});

const toIcon = new L.Icon({
    iconUrl: 'https://raw.githubusercontent.com/pointhi/leaflet-color-
markers/master/img/marker-icon-red.png',
    shadowUrl: 'https://unpkg.com/leaflet@1.9.3/dist/images/marker-
shadow.png',
    iconSize: [25, 41],
    iconAnchor: [12, 41],
    popupAnchor: [1, -34],
    shadowSize: [41, 41],
});

export const ClientParcelsPage: React.FC = () => {
  const { data: departments = [] } = useGetDepartmentsQuery();
  const { data: incomingParcels = [] } = useGetClientParcelsQuery();

  const [selectedParcel, setSelectedParcel] = useState<number |
null>(null);

  const getDepartmentById = (id: number) => departments.find(dep => dep.id
=== id);

  return (
    <div className="p-6 space-y-6 max-w-6xl mx-auto">
      <h1 className="text-2xl font-bold">Your Parcels</h1>

      <div className="h-96 w-full rounded shadow">
        <MapContainer center={[50.4501, 30.5234]} zoom={6} className="h-
full w-full z-0 rounded">
          <TileLayer
url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png" />

          {incomingParcels.map(parcel => {

            return (
              <Marker
                key={parcel.id}
                position={[+parcel.courier_latitude,
+parcel.courier_longitude]}
                icon={defaultIcon}
                eventHandlers={{
                  click: () => setSelectedParcel(parcel.id),
                }}
              >
                <Tooltip direction="top" offset={[0, -20]} opacity={1}>
```

```
                  <div className="text-sm">
                    <b>Parcel #{parcel.id}</b><br />
                    Status: {parcel.status}<br />
                    From Dept:
{getDepartmentById(parcel.sending_department_id)?.name}<br />
                    To Dept:
{getDepartmentById(parcel.receiving_department_id)?.name}<br />
                    Sender: {parcel.sender_email}<br />
                    Receiver: {parcel.receiver_email}<br />
                  </div>
                </Tooltip>
              </Marker>
            );
          })}

          {selectedParcel !== null && (() => {
            const parcel = incomingParcels.find(p => p.id ===
selectedParcel);
            if (!parcel) return null;

            const from = getDepartmentById(parcel.sending_department_id);
            const to = getDepartmentById(parcel.receiving_department_id);
            if (!from || !to) return null;

            const [fromLat, fromLng] =
from.location.split(',').map(Number);
            const [toLat, toLng] = to.location.split(',').map(Number);

            return (
              <>
                <Marker position={[fromLat, fromLng]} icon={fromIcon}>
                  <Popup>Sending Department: {from.name}</Popup>
                </Marker>
                <Marker position={[toLat, toLng]} icon={toIcon}>
                  <Popup>Receiving Department: {to.name}</Popup>
                </Marker>
              </>
            );
          })()}
        </MapContainer>
      </div>

      {/* Parcel List */}
      <div className="space-y-4">
        {['being delivered', 'delivered', 'received'].map(status => (
          <div key={status}>
            <h2 className="text-xl font-semibold capitalize">{status}
parcels</h2>
            <div className="space-y-2">
              {incomingParcels.filter(p => p.status === status).map(parcel
=> (
                <div
                  key={parcel.id}
                  className="p-4 border rounded hover:bg-gray-50 cursor-
pointer transition"
                  onClick={() => setSelectedParcel(parcel.id)}
                >
                  <p><b>Parcel #{parcel.id}</b></p>
                  <p>Status: {parcel.status}</p>
                  <p>From:
{getDepartmentById(parcel.sending_department_id)?.name}</p>
                  <p>To:
{getDepartmentById(parcel.receiving_department_id)?.name}</p>
                </div>
```

```
            ))}
          </div>
        </div>
      ))}
    </div>
  </div>
  );
}; import React, { useState } from 'react';
import { useNavigate } from 'react-router-dom';
import { useRegisterMutation } from '../store/api';

const CreateCourier: React.FC = () => {
  const [createProfile, { isLoading, error }] = useRegisterMutation();
  const navigate = useNavigate();

  const [form, setForm] = useState({
    email: '',
    password: 'a',
  });

  const handleChange = (e: React.ChangeEvent<HTMLInputElement>) => {
    setForm((prev) => ({
      ...prev,
      [e.target.name]: e.target.value,
    }));
  };

  const handleSubmit = async (e: React.FormEvent) => {
    e.preventDefault();
    try {
      await createProfile({
        ...form,
        role: 'courier',
      }).unwrap();
      navigate('/admin');
    } catch (err) {
      console.error(err);
    }
  };

  return (
    <div className="max-w-lg mx-auto p-6 bg-white shadow rounded-xl space-
y-6">
      <h2 className="text-2xl font-semibold">Create Courier</h2>

      <form onSubmit={handleSubmit} className="space-y-4"><br/>
        <input
          name="email"
          value={form.email}
          onChange={handleChange}
          required
          placeholder="Courier Email"
          className="w-full p-2 border rounded"
        /><br/>

        <button
          type="submit"
          disabled={isLoading}
          className="w-full bg-blue-600 text-white py-2 rounded hover:bg-
blue-700"
        >
          {isLoading ? 'Creating...' : 'Create Courier'}
        </button>
```

```
        {error && <p className="text-red-500 text-sm">Failed to create
courier</p>}
      </form>
    </div>
  );
};

export default CreateCourier; import React, { useState } from 'react';
import { useNavigate } from 'react-router-dom';
import { MapContainer, TileLayer, useMapEvents, Marker, Popup } from
'react-leaflet';
import L from 'leaflet';
import { useCreateDepartmentMutation, useGetDepartmentsQuery } from
'../store/api'; // adjust path if needed

// Fix Leaflet icon loading
const defaultIcon = new L.Icon({
  iconUrl: 'https://unpkg.com/leaflet@1.9.3/dist/images/marker-icon.png',
  iconSize: [25, 41],
  iconAnchor: [12, 41],
});

const CreateDepartment: React.FC = () => {
  const navigate = useNavigate();
  const [createDepartment, { isLoading, error }] =
useCreateDepartmentMutation();
  const { data: departments = [] } = useGetDepartmentsQuery();

  const [name, setName] = useState('');
  const [location, setLocation] = useState<{ lat: number; lng: number } |
null>(null);

  const handleSubmit = async (e: React.FormEvent) => {
    e.preventDefault();
    if (!location || !name) return;

    try {
      await createDepartment({
        name,
        location: `${location.lat},${location.lng}`,
      }).unwrap();
      navigate('/admin');
    } catch (err) {
      console.error(err);
    }
  };

  return (
    <div className="max-w-2xl mx-auto p-6 bg-white rounded-xl shadow space-
y-6">
      <h2 className="text-2xl font-semibold">Create Department</h2>

      <form onSubmit={handleSubmit} className="space-y-4">
        <input
          type="text"
          value={name}
          onChange={(e) => setName(e.target.value)}
          placeholder="Department name"
          required
          className="w-full p-2 border rounded"
        />

        <div>
```

```
          <p className="mb-1 text-sm text-gray-600">Click on the map to
select a location</p>
          <div className="h-64">
            <MapContainer center={[51.505, -0.09]} zoom={5} className="h-
full w-full rounded z-0">
              <TileLayer
                url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
              />

              {/* Existing departments as markers */}
              {departments.map((dep) => {
                const [lat, lng] = dep.location?.split(',').map(Number);
                if (!lat || !lng) return null;

                return (
                  <Marker key={dep.id} position={[lat, lng]}
icon={defaultIcon}>
                    <Popup>
                      <strong>{dep.name}</strong><br />
                      Location: {lat.toFixed(3)}, {lng.toFixed(3)}
                    </Popup>
                  </Marker>
                );
              })}

              {/* New location selector */}
              <LocationSelector setLocation={setLocation} />

              {/* Marker for newly selected location */}
              {location && (
                <Marker position={[location.lat, location.lng]}
icon={defaultIcon}>
                  <Popup>New department location</Popup>
                </Marker>
              )}
            </MapContainer>
          </div>
        </div>

        <button
          type="submit"
          disabled={isLoading || !location}
          className="w-full bg-blue-600 text-white py-2 rounded hover:bg-
blue-700 transition"
        >
          {isLoading ? 'Creating...' : 'Create Department'}
        </button>

        {error && <p className="text-red-500 text-sm">Failed to create
department</p>}
      </form>
    </div>
  );
};

export default CreateDepartment;

// Helper component to capture click on the map
const LocationSelector: React.FC<{
  setLocation: (coords: { lat: number; lng: number }) => void;
}> = ({ setLocation }) => {
  useMapEvents({
    click(e) {
      setLocation(e.latlng);
```

```
    },
  });
  return null;
}; import React from 'react';
import { useNavigate, useParams } from 'react-router-dom';
import { useGetDepartmentByIdQuery, useGetDepartmentOperatorsQuery } from
'../store/api';

const DepartmentInfo: React.FC = () => {
  const { id } = useParams<{ id?: string }>()
  const departmentIdString = id ?? ''
  const navigate = useNavigate();
  const {data: department} = useGetDepartmentByIdQuery(departmentIdString)
  const {data: operators} =
useGetDepartmentOperatorsQuery(departmentIdString)
  const handleAddOperator = () => {
    navigate(`/departments/${department?.id}/add-operator`);
  };

  return (
    <div className="max-w-xl mx-auto p-6 bg-white shadow rounded-xl space-
y-6">
      {/* Department Info */}
      <div>
        <h2 className="text-2xl font-semibold mb-1">{department?.name}</h2>
        <p className="text-gray-600">Location: {department?.location}</p>
      </div>

      {/* Operators List */}
      <div>
        <h3 className="text-xl font-medium mb-2">Operators</h3>
        {operators && operators?.length > 0 ? (
          <ul className="space-y-2">
            {operators?.map((op) => (
              <li
                key={op.id}
                className="p-3 bg-gray-100 rounded-lg shadow-sm flex
justify-between items-center"
              >
                <span className="text-sm text-gray-500">{op.email}</span>
              </li>
            ))}
          </ul>
        ) : (
          <p className="text-gray-500 italic">No operators yet.</p>
        )}
      </div>

      {/* Plus Button */}
      <div className="flex justify-end">
        <button
          onClick={handleAddOperator}
          className="w-10 h-10 rounded-full bg-blue-600 text-white text-2xl
flex items-center justify-center shadow-md hover:bg-blue-700 active:scale-
95 transition-transform"
          title="Add new operator"
        >
          +
        </button>
      </div>
    </div>
  );
};
```

```
export default DepartmentInfo; import React from 'react';
import { useGetDepartmentsQuery } from '../store/api';
import { useNavigate } from 'react-router-dom';
import { MapContainer, TileLayer, Marker, Popup } from 'react-leaflet';
import L from 'leaflet';
import PlusButton from '../components/PlusButton';

// Fix Leaflet icon loading
const defaultIcon = new L.Icon({
  iconUrl: 'https://unpkg.com/leaflet@1.9.3/dist/images/marker-icon.png',
  iconSize: [25, 41],
  iconAnchor: [12, 41],
});

export const AdminDashboard: React.FC = () => {
  const { data: departments = [] } = useGetDepartmentsQuery();
  const navigate = useNavigate();

  return (
    <div className="p-6 space-y-6 max-w-4xl mx-auto">
      <h1 className="text-2xl font-bold">Departments</h1>

      {/* Map with existing departments */}
      <div className="h-96 w-full rounded shadow">
      <MapContainer center={[50.4501, 30.5234]} zoom={6} className="h-full
w-full z-0 rounded">
          <TileLayer
            url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
          />
          {departments.map((dep) => {
            const [lat, lng] = dep.location?.split(',').map(Number);
            if (!lat || !lng) return null;

            return (
              <Marker
                key={dep.id}
                position={[lat, lng]}
                icon={defaultIcon}
                eventHandlers={{
                  click: () => navigate(`/admin/${dep.id}`),
                }}
              >
                <Popup>
                  <strong>{dep.name}</strong><br />
                  Location: {lat.toFixed(3)}, {lng.toFixed(3)}
                </Popup>
              </Marker>
            );
          })}
        </MapContainer>
      </div>

      {/* Department List */}
      {departments.length > 0 ? (
        <div className="space-y-3">
          {departments.map((result) => (
            <div
              key={result.id}
              onClick={() => navigate(`/admin/${result.id}`)}
              className="cursor-pointer p-4 border rounded hover:bg-gray-50
transition"
            >
              <h2 className="text-lg font-medium">{result.name}</h2>
              <p><b>Location:</b> {result.location}</p>
```

```
            </div>
          ))}
        </div>
      ) : (
        <p>No departments found.</p>
      )}

      {/* Floating Plus Button */}
      <PlusButton onClick={() => navigate('/admin/createDepartment')} />
      <button onClick={() => navigate('/admin/addCourier')}>
        Add courier
      </button>
    </div>
  );
}; .incoming-parcels {
    padding: 2rem;
    max-width: 1000px;
    margin: 0 auto;
    font-family: Arial, sans-serif;

    h1 {
      text-align: center;
      margin-bottom: 2rem;
      font-size: 2rem;
    }

    .status-group {
      margin-bottom: 3rem;

      h2 {
        font-size: 1.5rem;
        color: #333;
        border-bottom: 2px solid #ccc;
        padding-bottom: 0.5rem;
        margin-bottom: 1rem;
      }

      .parcel-card {
        background-color: #f9f9f9;
        border: 1px solid #ddd;
        border-radius: 8px;
        padding: 1rem;
        margin-bottom: 1rem;
        transition: box-shadow 0.2s ease;

        &:hover {
          box-shadow: 0 2px 8px rgba(0, 0, 0, 0.1);
        }

        p {
          margin: 0.25rem 0;
          font-size: 0.95rem;

          strong {
            color: #444;
          }
        }

        button {
          margin-top: 0.75rem;
          padding: 0.4rem 1rem;
          background-color: #007bff;
          color: white;
          border: none;
```

```
          border-radius: 5px;
          cursor: pointer;
          font-size: 0.9rem;

          &:hover {
            background-color: #0056b3;
          }
        }
      }
    }
  }
  import React from 'react';
import {
  useGetIncomingParcelsQuery,
  useGetProfileQuery,
  useUpdateParcelStatusMutation,
} from '../store/api';
import './IncomingParcels.scss';

export const IncomingParcels: React.FC = () => {
  const { data: profile } = useGetProfileQuery();
  const { data: parcels = [], refetch } =
useGetIncomingParcelsQuery(profile?.department_id || 0);
  const [updateStatus] = useUpdateParcelStatusMutation();

  const handleStatusUpdate = async (id: number, newStatus: string) => {
    await updateStatus({ id, status: newStatus });
    refetch();
  };

  const categorizedParcels = {
    being_delivered: parcels.filter((p) => p.status === 'being delivered'),
    delivered: parcels.filter((p) => p.status === 'delivered'),
    received: parcels.filter((p) => p.status === 'received'),
  };

  const renderParcel = (parcel: any) => (
    <div key={parcel.id} className="parcel-card">
      <p><strong>ID:</strong> {parcel.id}</p>
      <p><strong>Status:</strong> {parcel.status}</p>
      <p><strong>Sender Email:</strong> {parcel.sender_email}</p>
      <p><strong>Receiver Email:</strong> {parcel.receiver_email}</p>
      <p><strong>Courier Email:</strong> {parcel.courier_email}</p>
      <p><strong>From Department:</strong>
{parcel.sending_department_name}</p>
      <p><strong>To Department:</strong>
{parcel.receiving_department_name}</p>
      <p><strong>Created At:</strong> {new
Date(parcel.created_at).toLocaleString()}</p>
      <p><strong>Updated At:</strong> {new
Date(parcel.updated_at).toLocaleString()}</p>

      {parcel.status === 'being delivered' && (
        <button onClick={() => handleStatusUpdate(parcel.id, 'delivered')}>
          Mark as Delivered
        </button>
      )}
      {parcel.status === 'delivered' && (
        <button onClick={() => handleStatusUpdate(parcel.id, 'received')}>
          Mark as Received
        </button>
      )}
    </div>
  );
```

```
  return (
    <div className="incoming-parcels">
      <h1>Incoming Parcels</h1>

      {Object.entries(categorizedParcels).map(([status, list]) => (
        <div key={status} className="status-group">
          <h2>{status.replace('_', ' ').toUpperCase()}</h2>
          {list.length > 0 ? (
            list.map(renderParcel)
          ) : (
            <p>No parcels in this category.</p>
          )}
        </div>
      ))}
    </div>
  );
}; .send-parcel {
  padding: 1.5rem;
  max-width: 64rem;
  margin: 0 auto;

  .title {
    font-size: 2rem;
    font-weight: bold;
    margin-bottom: 1rem;
  }

  .map-container {
    height: 24rem;
    width: 100%;
    margin-bottom: 2rem;
    border-radius: 0.5rem;
    overflow: hidden;
    box-shadow: 0 2px 8px rgba(0, 0, 0, 0.1);

    .map {
      height: 100%;
      width: 100%;
    }
  }

  .department-list {
    display: flex;
    flex-direction: column;
    gap: 0.75rem;
    margin-bottom: 2rem;

    .department-item {
      padding: 1rem;
      border: 1px solid #ddd;
      border-radius: 0.5rem;
      cursor: pointer;
      transition: background-color 0.2s;

      &:hover {
        background-color: #f9f9f9;
      }

      &.selected {
        border-color: #007bff;
        background-color: #e6f0ff;
      }
```

```
      .name {
        font-size: 1.125rem;
        font-weight: 500;
      }

      .location {
        font-size: 0.9rem;
        margin-top: 0.25rem;
      }
    }
  }

  .parcel-form {
    border-top: 1px solid #ddd;
    padding-top: 2rem;

    .form-title {
      font-size: 1.5rem;
      font-weight: 600;
      margin-bottom: 1rem;
    }

    .form-group {
      display: flex;
      flex-direction: column;
      margin-bottom: 1rem;

      label {
        font-weight: 500;
        margin-bottom: 0.5rem;
      }

      input,
      select {
        padding: 0.5rem;
        border: 1px solid #ccc;
        border-radius: 0.375rem;
        font-size: 1rem;
      }
    }

    .submit-button {
      background-color: #007bff;
      color: white;
      font-weight: 500;
      border: none;
      padding: 0.75rem 1.5rem;
      border-radius: 0.375rem;
      cursor: pointer;
      transition: background-color 0.2s ease;

      &:hover {
        background-color: #0056b3;
      }
    }
  }
}

.department-item.disabled {
  opacity: 0.5;
  cursor: not-allowed;
  pointer-events: none;

  .your-department {
```

```
    font-style: italic;
    color: gray;
  }
}

.marker-disabled {
  background: gray;
  width: 20px;
  height: 20px;
  border-radius: 50%;
} import React, { useState } from 'react';
import { useGetDepartmentsQuery, useGetCouriersQuery } from '../store/api';
import { MapContainer, TileLayer, Marker, Popup } from 'react-leaflet';
import L from 'leaflet';
import './OperatorSendParcel.scss';
import { useSendParcelMutation } from '../store/api';
import { useGetProfileQuery } from '../store/api';


const defaultIcon = new L.Icon({
  iconUrl: 'https://unpkg.com/leaflet@1.9.3/dist/images/marker-icon.png',
  shadowUrl: 'https://unpkg.com/leaflet@1.9.3/dist/images/marker-
shadow.png',
  iconSize: [25, 41],
  iconAnchor: [12, 41],
  popupAnchor: [1, -34],
  shadowSize: [41, 41],
});

const highlightedIcon = new L.Icon({
  iconUrl: 'https://raw.githubusercontent.com/pointhi/leaflet-color-
markers/master/img/marker-icon-red.png',
  shadowUrl: 'https://unpkg.com/leaflet@1.9.3/dist/images/marker-
shadow.png',
  iconSize: [25, 41],
  iconAnchor: [12, 41],
  popupAnchor: [1, -34],
  shadowSize: [41, 41],
});

export const OperatorSendParcel: React.FC = () => {
  const { data: departments = [] } = useGetDepartmentsQuery();
  const { data: couriers = [] } = useGetCouriersQuery();
  const [sendParcel, { isLoading }] = useSendParcelMutation();
  const { data: profile } = useGetProfileQuery();

  const [selectedDepartmentId, setSelectedDepartmentId] = useState<number |
null>(null);
  const [formData, setFormData] = useState({
    senderEmail: '',
    receiverEmail: '',
    courierId: '',
  });

  const handleSelectDepartment = (id: number) => {
    setSelectedDepartmentId(id);
  };

  const handleChange = (e: React.ChangeEvent<HTMLInputElement |
HTMLSelectElement>) => {
    const { name, value } = e.target;
    setFormData(prev => ({ ...prev, [name]: value }));
  };
```

```
  const handleSubmit = async (e: React.FormEvent) => {
    e.preventDefault();

    if (!selectedDepartmentId || !formData.senderEmail ||
!formData.receiverEmail || !formData.courierId) {
      alert('Please fill out all fields and select a department.');
      return;
    }

    try {
      await sendParcel({
        sending_department_id: profile?.department_id || 0,
        receiving_department_id: selectedDepartmentId, // You can change
this logic if receiver dept differs
        sending_client_email: formData.senderEmail,
        receiving_client_email: formData.receiverEmail,
        courier_id: formData.courierId,
      }).unwrap();

      alert('Parcel sent successfully!');
      setFormData({ senderEmail: '', receiverEmail: '', courierId: '' });
      setSelectedDepartmentId(null);
    } catch (err) {
      console.error('Failed to send parcel:', err);
      alert('Failed to send parcel. Please try again.');
    }
  };


  return (
    <div className="send-parcel">
      <h1 className="title">Departments</h1>

      <div className="map-container">
        <MapContainer center={[50.4501, 30.5234]} zoom={6} className="map">
          <TileLayer
url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png" />
          {departments.map((dep) => {
            const [lat, lng] = dep.location?.split(',').map(Number);
            if (!lat || !lng) return null;

            return (
              <Marker
  key={dep.id}
  position={[lat, lng]}
  icon={
    profile?.department_id === dep.id
      ? L.divIcon({ className: 'marker-disabled' }) // greyed-out icon
      : dep.id === selectedDepartmentId
      ? highlightedIcon
      : defaultIcon
  }
  eventHandlers={
    profile?.department_id == dep.id
      ? {} // no interaction if it's operator's own dept
      : {
          click: () => handleSelectDepartment(dep.id),
        }
  }
>
  <Popup>
    <strong>{dep.name}</strong><br />
    Location: {lat.toFixed(3)}, {lng.toFixed(3)}
    {profile?.department_id == dep.id && (
```

```
      <div style={{ color: 'gray' }}><em>Your department</em></div>
    )}
  </Popup>
</Marker>
            );
          })}
        </MapContainer>
      </div>

      <div className="department-list">
        {departments.map((dep) => (
          <div
          key={dep.id}
          className={`department-item ${dep.id === selectedDepartmentId ?
'selected' : ''} ${profile?.department_id === dep.id ? 'disabled' : ''}`}
          onClick={() => {
            if (profile?.department_id !== dep.id) {
              handleSelectDepartment(dep.id);
            }
          }}
        >
          <h2 className="name">{dep.name}</h2>
          <p className="location"><strong>Location:</strong>
{dep.location}</p>
          {profile?.department_id == dep.id && (
            <p className="your-department">(Your department)</p>
          )}
        </div>

        ))}
      </div>

      <form className="parcel-form" onSubmit={handleSubmit}>
        <h2 className="form-title">Send Parcel</h2>

        <div className="form-group">
          <label>Sender Email</label>
          <input
            type="email"
            name="senderEmail"
            value={formData.senderEmail}
            onChange={handleChange}
            required
          />
        </div>

        <div className="form-group">
          <label>Receiver Email</label>
          <input
            type="email"
            name="receiverEmail"
            value={formData.receiverEmail}
            onChange={handleChange}
            required
          />
        </div>

        <div className="form-group">
          <label>Courier</label>
          <select
            name="courierId"
            value={formData.courierId}
            onChange={handleChange}
            required
```

```
          >
            <option value="">Select Courier</option>
            {couriers.map(courier => (
              <option key={courier.id} value={courier.id}>
                {courier.email}
              </option>
            ))}
          </select>
        </div>

        <button type="submit" className="submit-button"
disabled={isLoading}>
          {isLoading ? 'Sending...' : 'Send Parcel'}
        </button>

      </form>
    </div>
  );
}; import { FC } from "react";
import { Provider } from "react-redux";
import { BrowserRouter, Routes, Route } from "react-router-dom";
import { Layout } from "./components/Layout";
import { AuthenticationProvider } from
"./providers/AuthenticationProvider";
import { Login } from "./scenes/Login";
import store from "./store";
import { AdminDashboard } from "./scenes/Departments";
import DepartmentInfo from "./scenes/DepartmentInfo";
import AddOperator from "./scenes/AddOperator";
import CreateDepartment from "./scenes/CreateDepartment";
import './global.css'
import 'leaflet/dist/leaflet.css';
import CreateCourier from "./scenes/CreateCourier";
import { OperatorSendParcel } from "./scenes/OperatorSendParcel";
import {IncomingParcels} from "./scenes/IncomingParcels";
import { ClientParcelsPage } from "./scenes/ClientParcelsPage";

export const App: FC = () => {


    return(
    <Provider store={store}>
    <AuthenticationProvider>
      <BrowserRouter>
        <Routes>
          <Route path="/" element={<Layout />}>
            <Route index element={<Login />} />
            <Route path="/admin" element={<AdminDashboard />} />
            <Route path="/admin/:id" element={<DepartmentInfo />} />
            <Route path="/departments/:id/add-operator"
element={<AddOperator />} />
            <Route path="/admin/createDepartment"
element={<CreateDepartment />} />
            <Route path="/admin/addCourier" element={<CreateCourier />} />
            <Route path="/operator/send" element={<OperatorSendParcel />}
/>
            <Route path="/operator/incoming" element={<IncomingParcels />}
/>
            <Route path="/client" element={<ClientParcelsPage />} />

          </Route>
        </Routes>
      </BrowserRouter>
    </AuthenticationProvider>
```

```
  </Provider>
    )
} import { createApi, fetchBaseQuery } from '@reduxjs/toolkit/query/react'
import { LoggedUserBody, LoginBody, Profile, RegProfile } from
'./types/Profile.types';
import { CreateDepartmentBody, Department } from
'./types/Department.types';
import { CreateParcel, Parcel } from './types/Parcel.types';

export const postTrackApi = createApi({
    reducerPath: 'api',
    baseQuery: fetchBaseQuery({
        baseUrl: 'http://localhost:3000/api',
        prepareHeaders: (headers) => {
            const token = localStorage.getItem('token');
            console.log(token)
            if (token) {
                headers.set('Authorization', `Bearer ${token}`);
              }
            return headers;
        },
    }),
    tagTypes: [
        'Profile',
        'Department',
        'Parcel'
    ],
    endpoints: (builder) => ({
        login: builder.mutation<LoggedUserBody, LoginBody>({
            query: (body) => ({
                url: `/login`,
                method: 'POST',
                body,
            }),
            async onQueryStarted(_arg, { queryFulfilled }) {
                try {
                    const { data } = await queryFulfilled;
                    console.log(data)
                    localStorage.setItem('token', data.token);
                } catch (error) {
                    console.error("Login error:", error);
                }
            },
            invalidatesTags: ['Profile']
        }),

        getProfile: builder.query<Profile, void>({
            //@ts-ignore
            queryFn: async (_arg, _queryApi, _extraOptions, fetchWithBQ) =>
{
                const token = localStorage.getItem('token');
                if (!token) {
                    return { data: undefined }; // Prevent the query if
token isn't available
                }
                const responce = await fetchWithBQ(`/profile/`)
                if (responce.error) {
                    return { data: undefined }
                }
                return responce
            },
            providesTags: ['Profile']
        }),
```

```
        logOut: builder.mutation<void, void>({
            query: (body) => ({
                url: `/auth/logout/`,
                method: 'POST',
                body,
            }),
            invalidatesTags: ['Profile']
        }),

        register: builder.mutation<any, RegProfile>({
            query: (body) => ({
                url: `/register/`,
                method: 'POST',
                body,
            }),
            invalidatesTags: ['Profile', 'Department']
        }),

        getDepartments: builder.query<Department[], void>({
            query: () => '/departments',
            providesTags: (result) =>
                result
                    ? [
                        ...result.map(({ id }) => ({ type: 'Department' as
const, id })),
                        'Department',
                    ]
                    : ['Department'],
        }),

        getDepartmentById: builder.query<Department, string>({
            query: (id) => `/departments/${id}`,
            providesTags: ['Department'],
        }),

        getDepartmentOperators: builder.query<Profile[], string>({
            query: (id) => `/departments/${id}/operators`,
            providesTags: (result) =>
                result
                    ? [
                        ...result.map(({ id }) => ({ type: 'Department' as
const, id })),
                        'Department',
                    ]
                    : ['Department'],
        }),

        createDepartment: builder.mutation<void, CreateDepartmentBody>({
            query: (body) => ({
                url: `/departments/`,
                method: 'POST',
                body,
            }),
            invalidatesTags: ['Department']
        }),

        getCouriers: builder.query<Profile[], void>({
            query: () => '/couriers',
            providesTags: (result) =>
                result
                    ? [
                        ...result.map(({ id }) => ({ type: 'Profile' as
const, id })),
                        'Profile',
```

```
                    ]
                    : ['Profile'],
        }),

        sendParcel: builder.mutation<{id: number}, CreateParcel>({
            query: (body) => ({
              url: '/parcel',
              method: 'POST',
              body,
            }),
            invalidatesTags: ['Parcel'],
          }),
          getIncomingParcels: builder.query<Parcel[], number>({
            query: (id) => `/parcels/${id}/incoming`,
          }),
          updateParcelStatus: builder.mutation<void, { id: number; status:
string }>({
            query: ({ id, status }) => ({
              url: `/parcels/${id}/status`,
              method: 'PATCH',
              body: { status },
            }),
          }),
          getClientParcels: builder.query<Parcel[], void>({
            query: () => '/parcel/client',
            providesTags: ['Parcel'],
          }),

          getCourierById: builder.query<Profile, number>({
            query: (id) => `/couriers/${id}`,
            providesTags: ['Profile'],
          }),

    }),
})

export const {
    useGetProfileQuery,
    useRegisterMutation,
    useLoginMutation,
    useLogOutMutation,
    useGetDepartmentsQuery,
    useGetDepartmentByIdQuery,
    useGetDepartmentOperatorsQuery,
    useCreateDepartmentMutation,
    useGetCouriersQuery,
    useSendParcelMutation,
    useGetIncomingParcelsQuery,
    useUpdateParcelStatusMutation,
    useGetClientParcelsQuery,
    useGetCourierByIdQuery,
} = postTrackApi
```

Додаток 3

Копія презентації

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
"КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО"

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП'ЮТЕРНИХ СИСТЕМ

ВЕБЗАСТОСУНОК ДЛЯ ОБЛІКУ ТА РЕЄСТРАЦІЇ ПОШТОВИХ ВІДДІЛЕНЬ

Виконав: Лунев Артем Вадимович

Керівник: доцент кафедри ПЗКС, к.т.н., доцент, Новак Дмитро Сергійович

Київ – 2025

ПОСТАНОВКА ЗАДАЧІ

Мета проєкту: створення спеціалізованого вебзастосунку для обліку поштових відправлень та управління логістичними процесами в межах мережі поштових відділень**.**

Завдання:

1. Проаналізувати ринок ПЗ для відстеження відправлень та обліку поштових відділень
2. Порівняти конкурентів
3. Визначити вимоги до ПЗ
4. Обґрунтувати вибір технологій
5. Реалізувати вебзастосунок
6. Протестувати систему на відповідність вимогам

АКТУАЛЬНІСТЬ

У порівнянні з 2023 роком, кількість здійснених поштових відправлень по Україні зросла майже на 30% за 2024 рік.

Період	Доходи, тис. грн				Обсяги, тис. од.		
	кур'єрські послуги	внутрішні	міжнародні	всього	внутрішні	міжнародні	всього
9 місяців 2023 року	535 647,80	21 629 729,90	605 134,10	22 770 511,80	309 387,90	9 885,44	319273,32
9 місяців 2024 року	2 080 793,12	25 152 202,84	1 367 205,00	28 600 200,96	427 412,20	20 104,21	447 516,41

Національна комісія, що здійснює державне регулювання у сферах електронних комунікацій, радіочастотного спектра та надання послуг поштового зв`язку

ІСНУЮЧІ РІШЕННЯ

UIS

Onfleeet

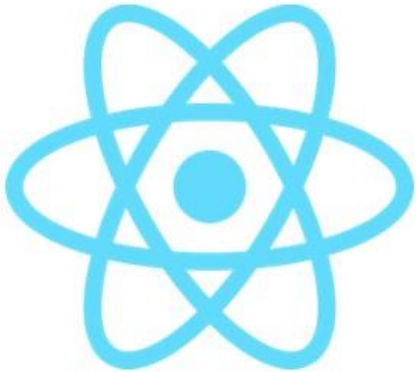
onfleet

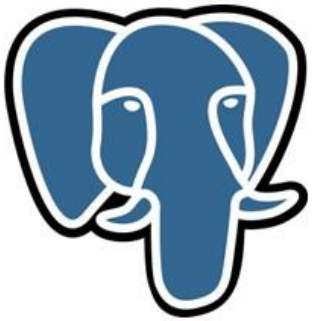
ДЕРЕВО ПРОБЛЕМ

ВИБІР ТЕХНОЛОГІЙ РОЗРОБКИ

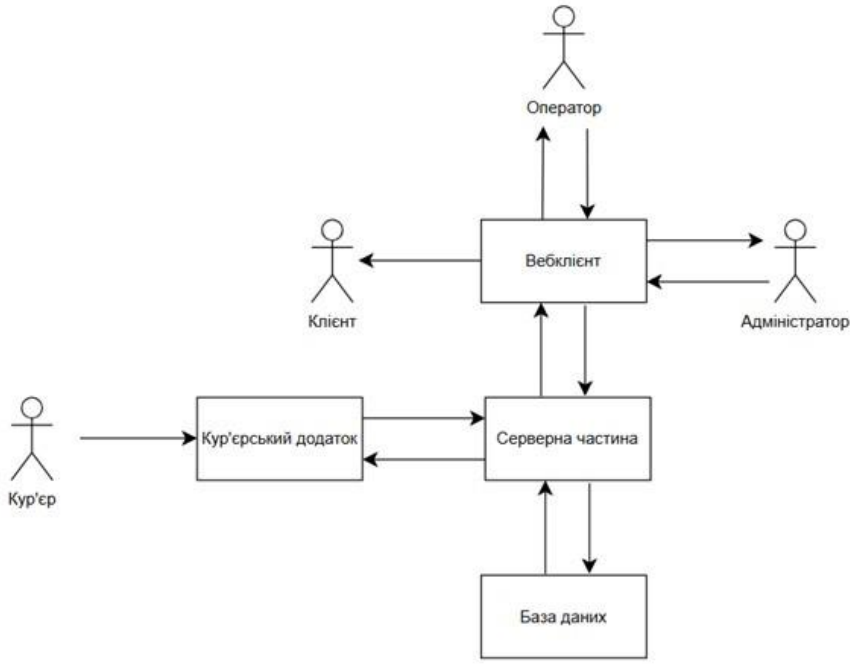
React

PosrgreSQL

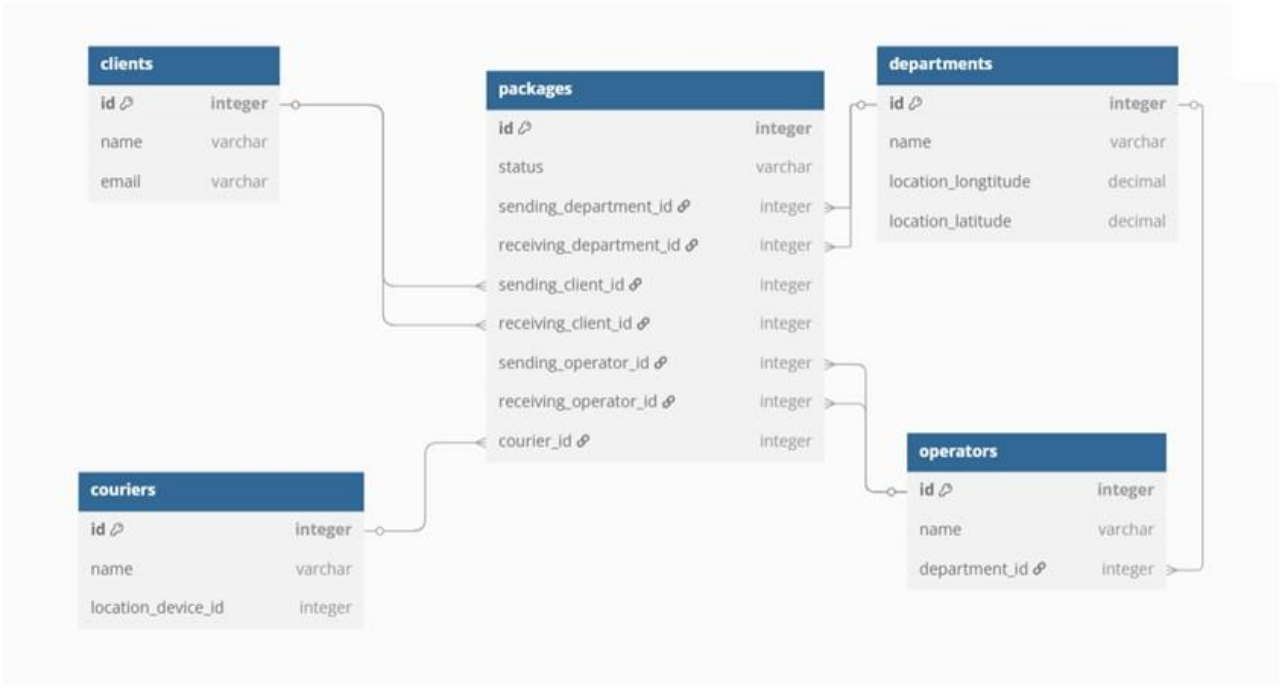




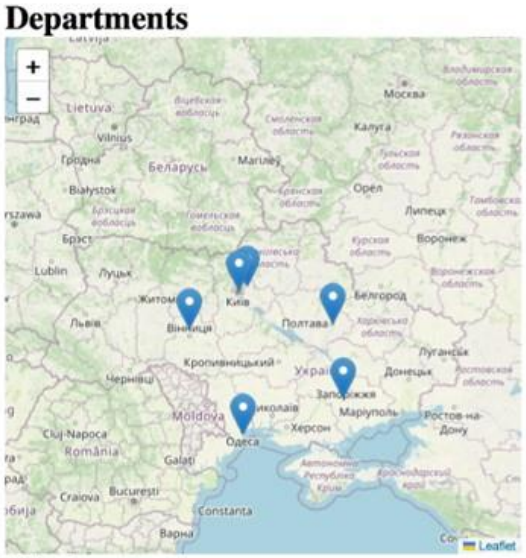
СТРУКТУРА ДОДАТКУ



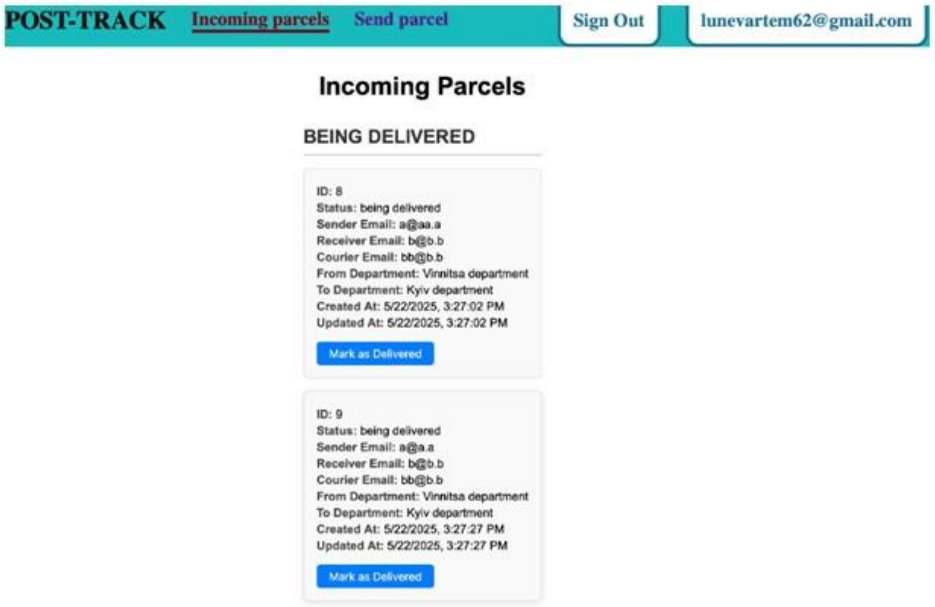
СТРУКТУРА БАЗИ ДАНИХ



СТОРІНКИ АДМІНІСТРАТОРА

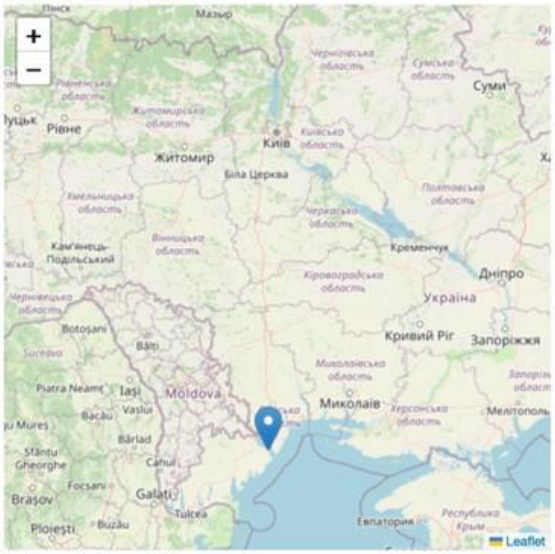


СТОРІНКИ ОПЕРАТОРА ВІДДІЛЕННЯ



СТОРІНКА КЛІЄНТА

Your Parcels



being delivered parcels

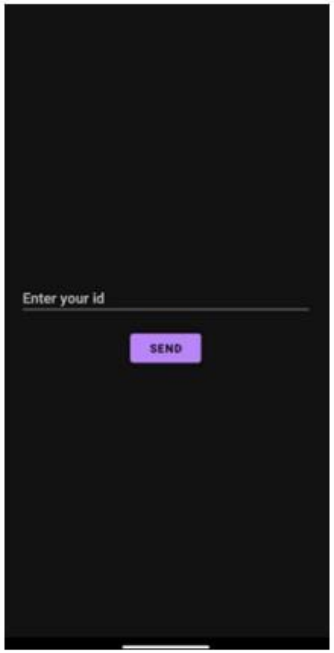
Parcel #7
Status: being delivered
From: Kyiv department
To: Odesa department

delivered parcels

received parcels

Parcel #6
Status: received
From: Vinnitsa department
To: Kyiv department

КУР’ЄРСЬКИЙ МОБІЛЬНИЙ ДОДАТОК



EMAIL НОТИФІКАЦІЯ КОРИСТУВАЧІВ

Parcel Sent Notification Вхідні ×

lariluna2003@gmail.com 22 трав. 2025 р., 15:27 (8 днів тому)
кому а

Hello!

A parcel was sent involving your profile. Please check your dashboard.

Welcome to PostTrack! Вхідні ×

lariluna2003@gmail.com чт, 22 трав., 15:27 (8 днів тому)
кому а

You have been automatically registered to PostTrack!

Email: a@aa.a
Password: b5b411c99b42
Role: CLIENT

РЕЗУЛЬТАТИ ТЕСТУВАННЯ ПЗ

Сервіс / Функціональна вимога	Onfleet	UIS TSM	Postrack
Облік поштових відділень	–	+	+
Внутрішнє переміщення відправлень між відділеннями	–	+	+
Призначення кур'єра та контроль виконання доставлення	+	+	+
Відстеження геолокації кур'єра в реальному часі	+	–	+
Рольовий поділ користувачів (оператор, кур'єр, клієнт)	–	+	+
Статус відправлень (прийнято, в дорозі, доставлено)	+	+	+

ВИСНОВКИ

•Було розроблено **вебзастосунок для обліку та реєстрації поштових відділень**.

•Проаналізовано **існуючі рішення** та виявлено, що вони не задовольняють специфічні потреби внутрішньої логістики поштових сервісів.

•Обґрунтовано вибір **технологій**: React для фронтенду, Node.js для бекенду, PostgreSQL — для зберігання даних, Kotlin — для мобільного застосунку кур'єра.

•Реалізовано **рольовий поділ користувачів** (адміністратор, оператор, клієнт, кур’єр), підтримку інтерактивної мапи та **email-нотифікацій**.

•Проведено **функціональне тестування**, що підтвердило відповідність системи поставленим вимогам.

Дякую за увагу!

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Завідувач кафедри

_____ Євгенія СУЛЕМА

"___" _____ 2024 р.

ВЕБЗАСТОСУНОК ДЛЯ ОБЛІКУ ТА РЕЄСТРАЦІЇ ПОШТОВИХ ВІДДІЛЕНЬ

Програма та методика тестування

ДП.045440-04-51

"ПОГОДЖЕНО"

Керівник проєкту:

_____ Дмитро НОВАК

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Артем ЛУНЕВ

2024

ЗМІСТ

1. ОБ’ЄКТ ВИПРОБУВАНЬ

Об’єктом випробувань є вебзастосунок PostTrack – система для обліку поштових відправлень, керування відділеннями, реєстрації клієнтів, моніторингу кур’єрів у реальному часі та забезпечення прозорої логістики між поштовими підрозділами. Вебзастосунок реалізований з використанням JavaScript/TypeScript, з використанням бібліотек React (фронтенд), Node.js + Express (бекенд) і PostgreSQL (база даних).

2. МЕТА ТЕСТУВАННЯ

Метою тестування є перевірка відповідності поведінки системи її функціональним та нефункціональним вимогам. Під час тестування має бути перевірено:

1) коректність автентифікації та маршрутизації користувачів за ролями (адмін, оператор, кур’єр, клієнт);
2) правильність створення, перегляду та оновлення поштових відправлень;
3) стабільність обміну інформацією між клієнтською та серверною частиною через REST API;
4) взаємодію з геолокаційною мапою (відображення відділень і посилок);
5) надсилання електронних сповіщень про події (реєстрація, надходження відправлення тощо).

3. МЕТОДИ ТЕСТУВАННЯ

Тестування виконувалося методом Gray Box Testing, що дозволяє перевірити як зовнішню поведінку інтерфейсу, так і внутрішню логіку обробки даних у системі.

Основні методи тестування:

1. Функціональне тестування – перевірка поведінки інтерфейсу та бізнес-логіки відповідно до сценаріїв використання для кожної ролі користувача.
2. Інтеграційне тестування – тестування взаємодії фронтенда, API та бази даних.
3. Тестування продуктивності – оцінка стабільності роботи під час послідовного створення та оновлення великої кількості відправлень.
4. Інтерфейсне тестування – оцінка коректності елементів UI (поля, кнопки, повідомлення, карти).
5. Тестування безпеки – перевірка доступу до захищених маршрутів, використання токенів авторизації та обмеження доступу між ролями.

4. ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування виконувалося вручну з використанням функціональності вебзастосунку PostTrack у тестовому середовищі.

Порядок тестування:

1) динамічне ручне тестування – введення граничних, порожніх і некоректних значень у форми (email, координати, ролі тощо);
2) функціональне тестування на відповідність сценаріям використання, зокрема створення відділення, оператора, посилки тощо;
3) перевірка інтерфейсу в різних браузерах (Chrome, Firefox);
4) тестування навантаженням – послідовне створення великої кількості записів відправлень;
5) оцінка зручності інтерфейсу для користувачів різних ролей;
6) верифікація надсилання листів та оновлення статусів.

Факультет прикладної математики

Кафедра програмного забезпечення комп’ютерних систем

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Євгенія СУЛЕМА

“___” _____ 2025 р.

ВЕБЗАСТОСУНОК ДЛЯ ОБЛІКУ ТА РЕЄСТРАЦІЇ ПОШТОВИХ ВІДДІЛЕНЬ

Керівництво користувача

ДП.045440-05-34

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Дмитро НОВАК

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Артем ЛУНЕВ

2025

ЗМІСТ

1. Опис структури вебзастосунку

Вебзастосунок PostTrack складається з динамічних вебсторінок, контент яких формується залежно від ролі авторизованого користувача: адміністратор, оператор, кур’єр, клієнт. Після входу в систему кожен користувач потрапляє на відповідну панель управління, що містить функції, специфічні для його прав доступу.

До клієнтської частини застосунку належать такі вебсторінки:

– сторінка авторизації користувача;

– головна панель адміністратора;

– сторінка перегляду інформації про конкретне відділення;

– сторінка створення нового відділення;

– сторінка створення нового оператора;

– сторінка створення нового кур’єра;

– панель оператора для реєстрації нових відправлень;

– сторінка перегляду вхідних відправлень оператора;

– панель клієнта для перегляду відправлень.

Застосунок інтегровано з інтерактивною картою (Leaflet), яка дозволяє відображати розташування поштових відділень, кур’єрів, а також відображати маршрут переміщення відправлень.

2. Опис вмісту статичних вебсторінок

Сторінка входу містить два поля для введення електронної пошти та пароля, а також кнопку «Login». Після успішної авторизації система автоматично перенаправляє користувача на панель відповідно до його ролі (рис. 1).

Рис. 1. Сторінка входу

Панель адміністратора включає список усіх зареєстрованих поштових відділень, відображення їх на мапі та доступ до кожного окремого відділення. Доступна кнопка додавання нового відділення та кнопка переходу до форми створення нового кур’єра (рис. 2).

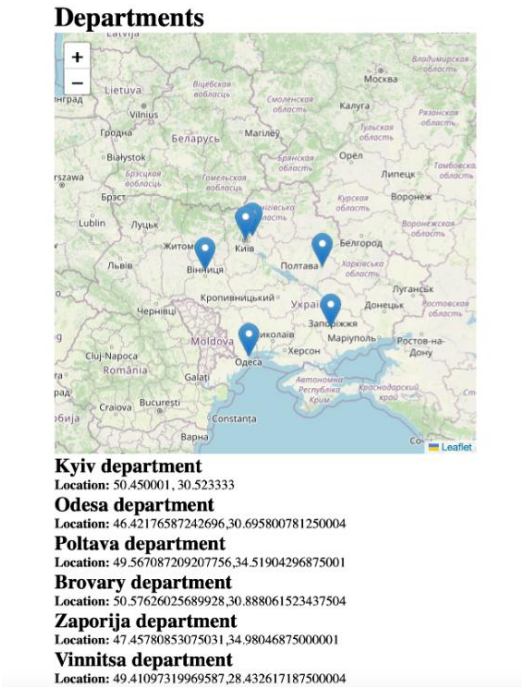


Рис. 2. Сторінка адміністратора

Сторінка інформації про відділення відображає назву та розташування конкретного відділення, а також список усіх прив’язаних до нього операторів. Доступна кнопка для переходу до форми додавання нового оператора (рис. 3).

Рис. 3. Сторінка відділення

Сторінка створення відділення містить форму для введення назви відділення та вибору його розташування за допомогою інтерактивної мапи. Після вибору координат і підтвердження – відділення зберігається у базі даних (рис. 4).

Рис. 4. Сторінка створення відділення

Сторінка створення нового оператора (рис. 5) – форма, яка дозволяє адміністратору створити обліковий запис нового оператора. Потрібно вказати email та пароль. Після збереження оператор додається до відділення.

Рис. 5. Сторінка створення оператора

Панель оператора – реєстрація відправлення (рис. 6), де оператор обирає відділення призначення на мапі, вводить email відправника, одержувача та вибирає кур’єра для доставки. Після підтвердження створюється нове відправлення, а користувачі отримують сповіщення.

Рис. 6. Сторінка реєстрації посилки

Панель клієнта (рис. 7), де клієнт бачить на мапі актуальні відправлення, які стосуються його профілю. Для кожної посилки вказано статус, маршрут (відділення відправника та одержувача), і дані кур’єра. Є можливість перегляду історії доставки.

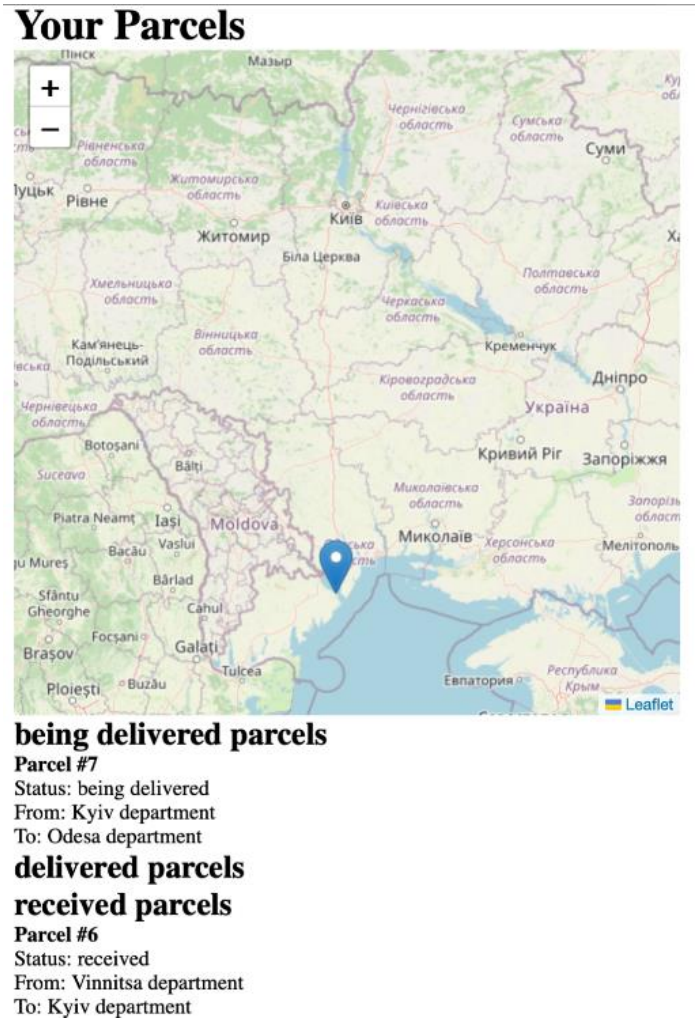


Рис. 7. Панель клієнта

3. Процедура авторизації користувача

Авторизація здійснюється на головній сторінці входу. Користувач вводить свій email і пароль, після чого система перевіряє наявність облікового запису.

Якщо дані правильні – користувач автоматично перенаправляється на відповідну панель:

– адміністратор – на сторінку зі списком відділень;

– оператор – на сторінку створення нового відправлення;

– клієнт – на сторінку перегляду власних посилок.

У разі некоректного введення з’являється повідомлення про помилку.