Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка веб-системи управління рейсами та багажем для транспортної компанії

Виконав: студент IV курсу, групи СНс-41
спеціальності 122 Комп’ютерні науки
(шифр і назва спеціальності)

_____ Маковинський М. Ю.
(підпис) (прізвище та ініціали)

Керівник _____ Липак Г. І.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Шимчук Г. В.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Боднарчук І. О.
(підпис) (прізвище та ініціали)

Рецензент _____ Жаровський Р. О.
(підпис) (прізвище та ініціали)

Тернопіль
2026

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
«_8_» _червня_ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)

за спеціальністю 122 Комп'ютерні науки
(шифр і назва спеціальності)

Студенту Маковинський Микола Юрійович
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка веб-системи управління рейсами та багажем для транспортної компанії

Керівник роботи Липак Галина Ігорівна, кандидат наук із соціальних комунікацій, доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «*14*» *травня* 2026 року № *4/9-237*

2. Термін подання студентом завершеної роботи 23 червня 2026 р.

3. Вихідні дані до роботи література та інтернет-джерела за тематикою дослідження

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. Розділ 1. Аналіз предметної області та постановка завдання розробки веб-системи. Розділ 2. Проєктування веб-системи управління рейсами та багажем. Розділ 3. Практична реалізація та аналіз результатів роботи веб-системи. Розділ 4. Безпека життєдіяльності, основи охорони праці. Висновки. Перелік джерел. Додатки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1. Титульний. 2. Актуальність, мета та завдання роботи. 3. Аналіз предметної області та аналогів. 4. Обґрунтування технологічного стеку. 5. Проєктування бази даних. 6. Демонстрація інтерфейсу: панель управління рейсами. 7. Демонстрація інтерфейсу: деталі рейсу та пасажири. 8. Демонстрація інтерфейсу: управління багажем та посилками. 9. Висновки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	Мариненко С. Ю., доцент		

7. Дата видачі завдання 26 січня 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	26.01.2026	*Виконано*
2.	Підбір та опрацювання літературних джерел по темі	27.01.2026-15.02.2026	*Виконано*
	кваліфікаційної роботи		
3.	Виконання дослідження щодо архітектурних рішень	11.05.2026-17.05.2026	*Виконано*
	вебсистем та алгоритмів оптимізації обліку		
	Проєктування та розробка веб-системи управління		
	рейсами та багажем для транспортної компанії		
4.	Оформлення розділу «Аналіз предметної області та	18.05.2026-24.05.2026	*Виконано*
	постановка завдання розробки веб-системи»		
5.	Оформлення розділу «Проєктування веб-системи	25.05.2026-01.06.2026	*Виконано*
	управління рейсами та багажем»		
6.	Оформлення розділу «Практична реалізація та аналіз	23.05.2026-31.05.2026	*Виконано*
	результатів роботи веб-системи»		
7.	Виконання завдання до підрозділу «Безпека життєдіяльності»	01.06.2026-08.06.2026	*Виконано*
8.	Виконання завдання до підрозділу «Основи охорони	09.06.2026-11.06.2026	*Виконано*
	праці»		
9.	Оформлення кваліфікаційної роботи	09.06.2026-11.06.2026	*Виконано*
10.	Нормоконтроль	12.06.2026-15.06.2026	*Виконано*
11.	Перевірка на плагіат	16.06.2026	*Виконано*
12.	Попередній захист кваліфікаційної роботи	18.06.2026	*Виконано*
13.	Захист кваліфікаційної роботи	26.06.2026	

Студент _____ Маковинський М. Ю.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Липак Г. І
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Розробка веб-системи управліннями рейсами та багажем для транспортної компанії // Кваліфікаційна робота освітнього рівня «Бакалавр» // Маковинський Микола Юрійович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних наук, група СНс-41 // Тернопіль, 2026 // С. 62, рис. – 11, табл. – 10, кресл. – 9, додат. – 3, бібліогр. – 35.

Ключові слова: веб-система, управління рейсами, транспортна компанія, облік багажу, пасажирські перевезення, python, django, postgresql.

Кваліфікаційна робота присвячена дослідженню процесів автоматизації управління рейсами та багажем транспортної компанії В першому розділі кваліфікаційної роботи описано особливості діяльності сучасної транспортної компанії та проаналізовано існуючі програмні аналоги на ринку. Висвітлено недоліки ручного ведення обліку. Розглянуто та обґрунтовано вибір технологічного стеку. Проведено аналіз вимоги до продукту та сформовано технічне завдання.

В другому розділі кваліфікаційної роботи спроєктовано загальну архітектуру веб-системи на базі патерна MVT. Досліджено структуру даних предметної області та розроблено концептуальну ER-діаграму бази даних із забезпеченням суворих правил цілісності й каскадного видалення.

В третьому розділі кваліфікаційної роботи описано програмну реалізацію серверної та клієнтської частин веб-системи. Проаналізовано структуру моделей даних, форм із динамічною фільтрацією та класів представлення. Проведено ручне тестування інтерфейсу, перевірено роботу алгоритмів контролю пасажиромісткості автотранспорту, генерації штрих-кодів для посилок та підсистеми Email-нотифікацій.

Об'єкт дослідження: бізнес-процеси обліку, планування рейсів та логістики в компаніях міжнародних пасажирських перевезень.

Предмет дослідження: методи, архітектурні патерни та веб-технології розробки інтегрованих інформаційних систем управління

ANNOTATION

Development of a Web-Based Travel and Baggage Management System for a Transport Company // Qualification work of the educational level «Bachelor» // Makovynskyi Mykola // Ternopil Ivan Pulyu National Technical University, Computer and Information Systems and Software Engineering Faculty, Computer Sciences Department, group SNs-41 // Ternopil, 2026 // P. 62, fig. – 11, tabl. – 10, chair. – 9, annexes. – 3, references – 35.

Keywords: web system, travel management, transport company, baggage tracking, passenger transportation, python, Django, postgresql.

The qualification thesis focuses on studying the automation of transportation route management and baggage handling processes within a transport company. The first chapter of the thesis describes the contemporary transport company operations and provides an analysis of existing software solutions available on the market. Also, the limitations of manual data management are identified and discussed. The selection of the technological stack is reviewed and substantiated. In addition, requirements for the system are analyzed, and the technical specification is prepared.

The second chapter focuses on the design of the overall web system architecture based on the MVT pattern. The data model of the domain is examined, and a conceptual ER database diagram is developed, incorporating strict integrity constraints and cascade deletion rules.

The third chapter presents the implementation of the server-side and client-side components of the web system. The structure of data models, dynamically filtered forms, and view classes is analyzed. Manual interface testing is performed, and the functionality of vehicle passenger capacity control algorithms, parcel barcode generation, and the email notification subsystem is verified.

Research Object: business processes related to accounting, route planning, and logistics in international passenger transport companies.

Research Subject: methods, architectural patterns, and web technologies used for developing integrated information management systems.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

CSS (англ. Cascading Style Sheets) – каскадні таблиці стилів, що використовуються для опису зовнішнього вигляду веб-сторінок.

ER (англ. Entity-Relationship model) – модель «сутність-зв'язок», графічне представлення структури даних.

HTML (англ. HyperText Markup Language) – стандартизована мова розмітки гіпертексту для створення веб-сторінок.

JS (англ. JavaScript) – мультипарадигмова мова програмування, що використовується для створення інтерактивних та динамічних елементів на стороні клієнта.

MVT (англ. Model-View-Template) – архітектурний патерн (шаблон) проєктування програмного забезпечення, що використовується у фреймворку Django.

ORM (англ. Object-Relational Mapping) – об'єктно-реляційне відображення, технологія, що зв'язує бази даних із концепціями об'єктно-орієнтованих мов програмування.

SQL (англ. Structured Query Language) – декларативна мова структурованих запитів для роботи з реляційними базами даних.

UML (англ. Unified Modeling Language) – уніфікована мова моделювання для графічного опису та проєктування програмних систем.

БД – база даних.

СУБД – система управління базами даних.

ЗМІСТ

ВСТУП

Актуальність теми. Сучасний ринок міжнародних та міжміських пасажирських перевезень динамічно розвивається, при цьому більшість малих та середніх приватних перевізників використовує гібридну бізнес-модель: поєднують транспортування пасажирів із доставкою приватних та комерційних посилок. Проте рівень автоматизації внутрішніх процесів у таких компаніях залишається низьким. Облік рейсів, бронювань та прийнятого багажу часто ведеться вручну за допомогою паперових журналів або найпростіших електронних таблиць. Це призводить до критичних ризиків: втрати інформації, плутанини з посилками на проміжних зупинках, фінансових збитків та логістичних збоїв у графіках руху.

Існуючі на ринку готові рішення або є великими закритими агрегаторами квитків, які не підтримують внутрішній операційний облік компанії, або є надто складними і дорогими логістичними системами, не адаптованими під специфіку невеликих компаній. Тому розробка гнучкої, надійної та доступної веб-системи, що дозволяє консолідувати облік рейсів, пасажирів та супутнього багажу в єдиному цифровому просторі, є актуальним напрямком сучасних досліджень та практичних розробок.

Мета і задачі дослідження. Метою даної кваліфікаційної роботи освітнього рівня «Бакалавр» є проєктування, програмна реалізація та тестування адаптивної веб-системи управління рейсами та багажем для транспортної компанії.

Для досягнення поставленої мети потрібно виконати ряд завдань, зокрема:

1. Проаналізувати стан досліджень у сфері автоматизації логістики, дослідити діяльність типової транспортної компанії та обґрунтувати вибір технологічного стеку.
2. Дослідити структуру даних, визначити межі функціональності системи та спроєктувати реляційну базу даних із забезпеченням цілісності

інформації. Побудувати комплекс UML-діаграм для структурного моделювання взаємодії користувачів із внутрішніми компонентами системи.

3. Програмно реалізувати серверну частину веб-системи, а також розробити адаптивний графічний інтерфейс.

4. Інтегрувати сервісні підсистеми автоматичної генерації штрих-кодів для маркування вантажів та надсилання email-нотифікацій.

5. Провести ручне тестування розробленої системи для підтвердження її працездатності, стійкості та відповідності технічним вимогам.

Об’єкт дослідження – бізнес-процеси операційного обліку, планування рейсів та логістики багажу в транспортних компаніях.

Предмет дослідження – методи, архітектурні патерни та веб-технології розробки інтегрованих інформаційних систем управління транспортними послугами.

Практичне значення одержаних результатів полягає у створенні готового до розгортання програмного продукту, який дозволяє транспортній компанії повністю відмовитися від паперового обліку, виключити ризики перевантаження автобусів завдяки автоматичному контролю пасажиромісткості, прискорити пошук і видачу посилок за допомогою штрих-кодів, а також налагодити оперативну координацію персоналу через систему автоматичних Email-сповіщень.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ РОЗРОБКИ ВЕБ-СИСТЕМИ

1.1 Аналіз діяльності транспортної компанії та існуючих підходів до управління рейсами

Основними складовими діяльності транспортних компаній, що спеціалізуються на міжнародних пасажирських перевезеннях, є не лише безпосереднє перевезення пасажирів, а й надання різних супутніх послуг, основною з яких є транспортування комерційних та приватних посилок. За рахунок такого поєднання дані транспортні компанії суттєво відрізняються від інших, що, як правило, працюють в одному конкретному напрямку — або пасажири, або посилки.

В процесі дослідження проаналізовано бізнес-процеси типової транспортної компанії, яка здійснює регулярні та нерегулярні рейси за кордон. Після проведеного аналізу можна описати типовий робочий цикл такої компанії, який складається з кількох етапів:

1. Планування рейсів. Людина, що відповідає за даний процес, як правило, формує розклад, призначає транспортний засіб, обирає водія та конкретну дату.
2. Опрацювання заявок від пасажирів. Клієнти бронюють місця за допомогою телефону або месенджерів, після чого диспетчер фіксує заявку з даними клієнта (ПІБ, контактний телефон, початкову та кінцеву точку маршруту та персональні побажання чи особливі вимоги пасажира).
3. Прийом та облік посилок. Перед відправленням або безпосередньо під час посадки водій приймає від клієнтів посилки (коробки, сумки, документи) для доставки за вказаною адресою. У ході даного етапу фіксуються дані відправника та отримувача, опис вмісту та спосіб оплати.

4. Контроль маршруту. Під час руху транспортного засобу водій повинен чітко знати де саме він повинен висадити пасажирів та передати відповідні посилки.

У більшості малих та середніх транспортних компаній такого типу автоматизація знаходиться на досить низькому рівні, оскільки облік ведеться, як правило, в «ручному режимі» за допомогою паперових журналів, блокнотів, електронних таблиць Microsoft Excel чи Google Sheets. Такий підхід обліку має ряд певних недоліків:

1. Втрата інформації. Відповідальна особа може забути внести пасажира до списку або випадково видалити рядок у таблиці, що може призвести до браку фізичних місць в автобусі під час посадки.

2. Плутанина з передачами. Оскільки посилки маркуються за допомогою позначок маркером або особливим описом посилки, водії часто витрачають багато часу на пошук потрібної посилки на проміжних зупинках, або взагалі переплутати багаж.

3. Відсутність контролю. Важко відстежити, яка сума була оплачена при відправленні посилки, а яка має бути стягнута з отримувача при її передачі, що створює певні ризики фінансових збитків для компанії.

4. Проблеми з логістикою. Відсутність чіткого списку адрес може призвести до збоїв у графіку рейсу та його затримок.

Одним із найважливіших аспектів використання сучасних технологій в управлінні є автоматизація та оптимізація процесів, а також впровадження систем моніторингу та керування . За допомогою спеціалізованих програм та систем управління можна значно прискорити та спростити процес планування та координації, більш ефективно та оперативно реагувати на проблеми та затримки, що виникають [1].

Для обґрунтування доцільності розробки власного продукту було розглянуто наявні на ринку рішення. Більшість із них можна розділити на дві основні категорії:

1. Великі агрегатори та квиткові системи (наприклад Busfor та Infobus). Ці платформи чудово справляються з продажем квитків та залученням пасажирів, проте вони є повністю закритими для внутрішнього операційного обліку компанії. Вони не дозволяють реєструвати передачі та посилки, які водії беруть безпосередньо біля транспорту, і не дають зручного інструменту для щоденного керування автопарком. Крім того, вони стягують високу комісію з кожного квитка.

2. Складні логістичні системи. Таке програмне забезпечення орієнтоване на великі логістичні центри, контейнерні перевезення та палети. Вони є занадто перевантаженими складним функціоналом, вимагають тривалого навчання персоналу, мають високу вартість ліцензій та не пристосовані для швидкого обліку звичайних пасажирів у межах одного рейсу.

Таким чином можна дійти висновку, що на ринку присутній брак простих доступних та гнучких інструментів, які були б адаптованими під специфіку невеликих транспортних компаній. Це показує актуальність розроби такої веб-системи.

1.2 Обґрунтування вибору технологічного стеку для реалізації веб-системи

Для реалізації веб-системи управління рейсами та багажем було обрано технологічний стек, який забезпечує високу швидкість розробки, надійність та простоту масштабування: Python та його фреймворк Django, СУБД PostgreSQL та класичні технології фронтенду (HTML, CSS, JavaScript). Таке поєднання дозволяє створити монолітну архітектуру, де серверна та клієнтська частини працюють як єдине ціле.

На сьогоднішній день Python впевнено займає друге місце серед найпопулярніших мов програмування, оскільки в неї широкий спектр застосування, оскільки його стандартна бібліотека містить готові інструменти для роботи з операційною системою, веб сторінками, базами даних та різними

форматами даних для побудови графічного інтерфейсу програм тощо [2], починаючи з простих програм та вебдодатків закінчуючи графічними додатками та веб-сайтами.

Серед ключових переваг мови Python варто виділити наступні:

- Висока читабельність та лаконічність коду. Синтаксис мови розроблений із фокусом на простоту та зрозумілість, завдяки чому вихідний код сприймається подібно до природної англійської мови. Це дозволяє суттєво скоротити час на розробку та подальшу підтримку програмного продукту, уникаючи громіздких та надлишкових конструкцій.

- Мультипарадигмальність. Python підтримує різні підходи до проєктування архітектури, зокрема об'єктно-орієнтоване, функціональне, структуроване, аспектно-орієнтоване та логічне програмування. Це підтверджує її універсальність та статус одного з найбільш передових інструментів розробки.

- Ефективне керування ресурсами. На відміну від компільованих мов (наприклад, C++), Python володіє вбудованою системою автоматичного керування пам'яттю (динамічний збирач сміття) та підтримує динамічну типізацію, що знижує ймовірність виникнення критичних помилок під час виконання програми.

Для побудови серверної частини веб-систем лідируючу роль відіграє високорівневий фреймворк Django. Він має багато компонентів, які зазвичай доводиться створювати з нуля, він значно прискорює процес розробки, дозволяючи програмістам зосередитися на бізнес-логіці замість рутинних завдань, таких як налаштування безпеки або робота з формами [3][4]. Також даний фреймворк має «із коробки» повну та оптимізовану сумісність із потужними реляційними базами даних, зокрема з обраною СУБД PostgreSQL.

Основними технічними перевагами фреймворку для даної веб-системи є:

- Потужний інструмент Django ORM. Оскільки веб-система має оперувати чітко пов'язаними сутностями (рейси, пасажири, передачі), використання ORM дозволить повністю відмовитися від написання «сирих»

SQL-запиті. Django ORM дозволяє транслювати код мови Python в оптимізовані SQL-запити до PostgreSQL, що спрощує вибірку даних та гарантує захист від помилок на рівні типів даних [5].

- Вбудована система безпеки та автентифікації. Проєктована веб-система передбачає роботу користувачів із різними правами доступу. Django надає підсистему автентифікації та розмежування прав доступу «із коробки». Крім того, фреймворк на базовому рівні блокує критичні вразливості, такі як SQL-ін'єкції (через екранування запитів в ORM) та CSRF-атаки (завдяки обов'язковій токенізації форм) [6].
- Ефективний механізм шаблонізації. Використання вбудованого шаблонізатора дозволяє динамічно рендерити HTML-сторінки на стороні сервера. Це означає, що водій або диспетчер отримує вже готовий сформований HTML-код із актуальними даними про рейс, що забезпечує швидке перше завантаження сторінок навіть за умов слабкого мобільного зв'язку.

Для надійного зберігання інформації та її управління обрано реляційну СУБД PostgreSQL, оскільки це надійна система баз даних, яка може обробляти дуже великі обсяги даних [7]. Специфіка транспортної веб-системи вимагає суворої консистентності даних: пасажир або передача не можуть існувати без прив'язки до конкретного рейсу, а кожен рейс обов'язково повинен містити валідний транспортний засіб та адресу.

PostgreSQL повністю задовольняє цим вимогам завдяки:

- Суворому дотриманню принципів ACID. Це гарантує, що будь-яка транзакція (наприклад, одночасне додавання кількох посилок на рейс диспетчером) буде виконана коректно або повністю скасована у разі збою, запобігаючи появі «битої» інформації.
- Підтримці механізмів обмежень цілісності: використання зовнішніх ключів та унікальних обмежень на рівні БД дозволить унеможливити видалення критично важливих адрес чи транспорту, якщо до них прив'язані активні рейси.

Клієнтський інтерфейс веб-системи розроблено на базі стандартного стеку HTML, CSS та JavaScript без використання важких SPA-фреймворків (таких як React чи Vue). Такий підхід забезпечує максимальну легкість та швидкодію інтерфейсу.

HTML та CSS складатимуть каркас та візуальне оформлення веб-системи оскільки важливим аспектом є реалізація адаптивної верстки. Основна взаємодія з системою відбуватиметься безпосередньо на маршруті через екран смартфона тому інтерфейс повинен гнучко підлаштовується під будь-яку роздільну здатність екрану. JavaScript забезпечить інтерактивність та динамічність користувацького інтерфейсу на клієнтській стороні.

Окрім того, важливим є вибір середовища розробки IDE (Integrated Development Environment), що спрощує взаємодію розробника з програмним кодом і візуалізацією його роботи [8].

PyCharm надає графічний відлагоджувач та засоби для аналізу коду, підтримує веб-розробку та працює на операційних системах Windows, Linux і Mac OS [9]. Простий інтерфейс, великі можливості, підтримка багатьох мов та доступність роблять PyCharm одним із найкращих середовищ для програмування [10].

1.3 Постановка завдання на розробку веб-системи

Метою кваліфікаційної роботи є проєктування, програмна реалізація та тестування надійної, швидкодіючої веб-систем з інтуїтивно зрозумілим і привабливим інтерфейсом, основним призначенням якої є автоматизація обліку пасажирів, рейсів, супутнього багажу (приватних та комерційних передач) та базової інфраструктури в межах діяльності транспортної компанії. [11]

Для досягнення поставленої мети необхідно вирішити такі ключові завдання:

1. Дослідити структуру даних та взаємодію між основними сутностями предметної області.

2. Спроєктувати реляційну базу даних, що унеможливлюватиме дублювання чи втрату інформації.

3. Розробити кастомний веб-інтерфейс, адаптований під специфіку роботи користувачів із різними технічними навичками та типами пристроїв.

Для забезпечення повноцінного функціонування розроблювана веб-система повинна реалізовувати наступний перелік функціональних вимог, розділених за відповідними архітектурними модулями:

1. Модуль керування транспортом та адресами

Цей модуль є довідниковою основою вебсистеми і відповідає за збереження статичних даних, які використовуються при формуванні маршрутів.

- Управління автопарком – веб-система повинна забезпечувати повноцінний CRUD-інтерфейс (створення, читання, оновлення, видалення) для облікових карток транспортних засобів. Для кожного транспортного засобу обов'язково фіксуються: марка/модель, унікальний державний реєстраційний номер, а також загальна пасажиромісткість (кількість фізичних місць), що використовуватиметься для автоматичного контролю ліміту пасажирів на рейс.
- Формування бази адрес – реалізація централізованого сховища географічних точок зупинок. Ці адреси є обов'язковими опорними пунктами під час реєстрації пасажирів та маркування точок видачі передач.

2. Модуль планування та керування рейсами

Модуль відповідає за створення розкладу компанії та консолідацію всієї інформації про конкретну поїздку.

- Створення рейсів. Відповідальна особа має мати можливість згенерувати унікальну сутність рейсу із чіткою прив'язкою додати та часу виїзду, призначенням конкретного транспорту та закріпленням відповідального водія.
- Моніторинг та фільтрація рейсів. Реалізація інтерфейсу для оперативного перегляду списку рейсів. Система повинна підтримувати архівацію та динамічне сортування за станом рейсу: заплановані, поточні та завершені рейси.

3. Модуль обліку та реєстрації пасажирів

Модуль призначений для ведення списку пасажирів, які подорожують конкретним рейсом, без перевантаження системи зайвими реєстраційними формами.

- Прив'язка до рейсу. Пасажир у системі створюється виключно як об'єкт всередині обраного рейсу. Процес додавання пасажира повинен контролюватися бізнес-логікою сервера: якщо кількість зареєстрованих осіб досягає ліміту пасажиромісткості призначеного транспортного засобу, система повинна блокувати додавання нових.
- Анкета пасажира. Збереження мінімально необхідного набору даних для ідентифікації та зв'язку: ПІБ, актуальний контактний номер телефону та адреса посадки та кінцева точка висадки з раніше сформованого довідника адрес.

4. Модуль обліку багажу та передач

Найбільш динамічний модуль системи, що автоматизує облік комерційних та приватних посилок, які перевозяться попутно з пасажирами.

- Реєстрація вантажу. Можливість швидкого внесення передачі на рейс. Картка передачі містить: короткий опис вмісту контактні дані відправника та обов'язково — номер телефону отримувача за кордоном.
- Логістичний контроль. Кожна передача суворо прив'язується до адреси видачі (пункту призначення), що дозволяє водію заздалегідь бачити, на якій саме зупинці необхідно розвантажити конкретний багаж.
- Генерація унікальних ідентифікаторів. Веб-система повинна містити алгоритм автоматичної генерації унікального номера для кожної зареєстрованої передачі. Номер має генеруватися на стороні сервера під час збереження запису в базі даних. На основі згенерованого номера система повинна динамічно створювати графічне зображення штрихкоду за до помогою якого водій або диспетчер зможе в один клік відкрити адаптивну форму для друку наліпки через портативний принтер. Це дозволить маркувати кожну коробку чи сумку, що забезпечить миттєвий пошук посилки в системі через пошуковий рядок.

5. Модуль автоматичних сервісних функцій та Email-нотифікацій

Цей модуль функціонує на рівні фонових системних подій та забезпечує миттєву комунікацію між учасниками логістичного процесу (адміністраторами, диспетчерами та водіями) через автоматичну генерацію та відправку електронних листів.

- Автоматичне сповіщення при управлінні посилками: у системі має бути реалізовано асинхронний контроль за життєвим циклом кожної посилки. Підсистема має реагувати на дві ключові події: реєстрація нових посилок та їхнє редагування.
- Сповіщення при управлінні рейсами: модуль автоматизує координацію водіїв та менеджменту компанії у межах рейсу, відстежуючи сценарії призначення нового рейсу та зміну даних поїздки.

Окрім виконання бізнес-логіки, веб-сервіс повинен мати спрощений дизайн та мінімальний час відгуку інтерфейсу для швидкої взаємодії користувача, а також підтримувати кросплатформеність для стабільної функціональності у будь-якому веб-браузері [12].

Зазначені модулі можуть бути початковою точкою для створення функціональної та ефективної веб-системи. Залежно від потреб компанії та її бізнес-моделі, можуть бути інші додаткові елементи, які слід враховувати при розробці [13].

1.4 Висновок до першого розділу

У першому розділі виконано аналіз предметної області, обґрунтовано вибір технологій та сформульовано технічне завдання на розробку веб-системи.

На основі проведеного дослідження отримано такі основні результати:

1. Визначено ключові бізнес-процеси, які вимагають одночасного оперативного обліку. Дослідження виявило критичні недоліки наявного ручного обліку, серед яких: високі ризики втрати даних, виникнення плутанини з багажем та відсутність прозорого контролю.

2. Доведено доцільність власної розробки програмного продукту. Встановлено, що існуючі аналоги або є повністю закритими, або є економічно недоцільними через високу вартість та перевантаженість надлишковим функціоналом.

3. Обґрунтовано вибір технологічного стеку для реалізації проєкту. Для серверної частини обрано мову Python та фреймворк Django у зв'язці з СУБД PostgreSQL, що забезпечує високу швидкість розробки, надійність, безпеку та сувору цілісність даних завдяки ORM. Для побудови клієнтської частини обрано стек HTML/CSS та JavaScript, який дозволяє реалізувати адаптивний, динамічний інтерфейс.

4. Сформульовано технічне завдання, яке було розділено на 5 взаємопов'язаних функціональних модулів: керування транспортом і адресами, планування рейсів, реєстрація пасажирів, облік посилок та модуль тригерних Email-нотифікацій для водіїв та суперюзерів на базі системних подій.

Отримані результати є повним теоретичним і практичним підґрунтям для переходу до другого розділу — проєктування архітектури та бази даних системи.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБ-СИСТЕМИ УПРАВЛІННЯ РЕЙСАМИ ТА БАГАЖЕМ

2.1 Проєктування архітектури веб-системи та взаємодії модулів

Сучасні інформаційні системи де кількість користувачів і обсяги даних постійно зростають, вибір відповідної архітектури стає критично важливим тому архітектурні підходи є ключовими елементами, що визначають їх продуктивність, надійність та гнучкість [14]. Для розробки веб-системи управління рейсами та багажем обрано класичну монолітну архітектуру на базі фреймворку Django, яка реалізує архітектурний патерн Model-View-Template. Організація системи у вигляді моноліту забезпечує максимальну швидкодію при взаємодії логічних модулів із базою даних, оскільки всі внутрішні виклики та транзакції виконуються в межах одного єдиного процесу додатку, що повністю виключає затримки на обмін даними, характерні для розподілених чи мікросервісних систем.

На рисунку 2.1 зображено архітектурний шаблон MVT, що покладений в основу проєктування системи, який розподіляє функціональність програми на три автономні, але жорстко інтегровані рівні, що взаємодіють між собою.

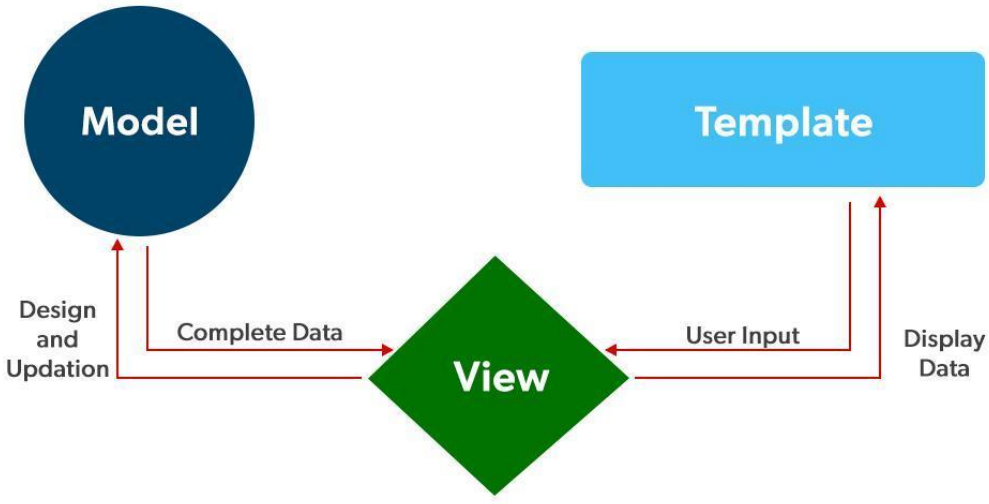


Рисунок 2.1 – Схема архітектурного шаблону MVT

Рівень моделі (Model) є першим і ключовим елементом архітектури. Він повністю відповідає за опис структури даних та бізнес-правил системи. Моделі є Python-класами, які за допомогою об'єктно-реляційного відображення (Django ORM) трансформуються у фізичні таблиці реляційної бази даних PostgreSQL. На цьому рівні зосереджена логіка валідації вхідних даних, перевизначення стандартних методів життєвого циклу об'єктів (зокрема, автоматичне формування унікальних ідентифікаторів) та контроль цілісності даних через системні обмеження СУБД. Моделі інкапсулюють у собі всі правила поведінки сутностей, захищаючи базу даних від некоректних записів.

Рівень представлення або шаблонів (Template) відповідає за безпосередню взаємодію з кінцевим користувачем , а якість інтерфейсу безпосередньо впливає на ефективність роботи користувача з системою [15]. На відміну від архітектур із розділеним фронтендом, у розробленій системі генерація користувацького інтерфейсу відбувається безпосередньо на стороні сервера. Шаблони проєктуються за допомогою вбудованої мови шаблонів Django Template Language, що дозволяє динамічно підставляти дані, отримані з бази, у структуру сторінок перед відправкою їх у браузер користувача. Такий підхід гарантує високу швидкість початкового завантаження сторінок та спрощує забезпечення безпеки, оскільки користувач отримує вже повністю сформований, валідований вміст, а логіка обробки залишається прихованою на сервері.

Рівень представлень (View) виконує роль сполучної ланки та головного координатора між рівнями моделей та шаблонів. Коли користувач виконує будь-яку дію в браузері, його запит аналізується відповідним класом або функцією у View. Представлення визначає тип запиту (перегляд сторінки чи відправка форми з даними), ініціює необхідні транзакції через ORM до бази даних, обробляє логіку веб-форм, перевіряє права доступу користувача на основі сесій і, в разі успіху, викликає рендеринг необхідного шаблону для відображення кінцевого результату.

Весь проєкт поділено на п'ять функціональних модулів, кожен з яких оформлений як окремий Django-застосунок:

1. Users – модуль, що відповідає за автентифікацію, авторизацію та збереження штатної структури підприємства. Він надає базову модель користувача для фіксації відповідальних осіб, які створюють записи.

2. Transport – модуль, що забезпечує облік фізичних ресурсів компанії автопарку та реєстраційних профілів водіїв, пов'язуючи їх із користувачами системи.

3. Passengers – модуль, що виступає універсальним інформаційним фундаментом системи, де ведеться єдиний довідник адрес та база контактних даних клієнтів.

4. Shipments – модуль, що повністю ізолює в собі бізнес-логіку обробки вантажообігу, фіксуючи параметри посилок, їхню вагу, вартість та типи доставки, а також відповідає за автоматичне кодування кожної одиниці багажу.

5. Trips – модуль, що є центральним вузлом. Він імпортує моделі з усіх інших чотирьох модулів і обєднує їх в єдиний процес, пов'язуючи водіїв, автомобілі та маршрути у конкретні рейси.

Підхід проєктування системи за принципом модульної архітектури забезпечило повну ізоляцію логіки в окремих компонентах, що мінімізує зв’язність коду і дозволяє поєднати ізольовані застосунки в цілісну систему.

2.2 Розробка інфологічної та даталогічної моделей бази даних

Проєктування інформаційного забезпечення системи є важливим етапом розробки. На етапі проєктування розробляється концепція БД, від якої залежить швидкість обробки даних та стійкість програми. База даних системи реалізована в реляційній системі управління базами даних PostgreSQL.

Структура таблиць та зв'язків між таблицями приведена до третьої нормальної форми, шляхом нормалізації моделі даних таким чином, щоб вона задовольняла функціональним різним вимогам і відповідала різним проєктним рішенням [16].

Підсистема користувачів відповідає за збереження автентифікацію персоналу та розмежування прав доступу до функцій системи. Він складається з таблиць Position (дивитись таблицю 2.1), яка є класичним довідником посад та Worker (дивитись таблицю 2.2) – модель користувача, що успадковує системні механізми безпеки Django.

Таблиця 2.1 – Структура таблиці Position

Назва поля	**Тип даних**	**Обмеження**	**Опис функціональності**
id	int	PRIMARY KEY	Унікальний ідентифікатор
name	varchar(255)	NOT NULL	Назва посади

Таблиця 2.2 – Структура таблиці Worker

Назва поля	**Тип даних**	**Обмеження**	**Опис функціональності**
id	int	PRIMARY KEY	Унікальний ідентифікатор.
username	varchar(150)	UNIQUE, NOT NULL	Логін працівника
email	varchar(255)	NOT NULL	Email користувача
first_name	varchar(255)	NOT NULL	Ім’я користувача
last_name	varchar(255)	NOT NULL	Прізвище користувача
password	varchar(128)	NOT NULL	Хеш пароля користувача
is_active	boolean	NOT NULL	Прапорець активності
is_superuser	boolean	NOT NULL	Прапорець адміністратора
position_id	bigint	FOREIGN KEY	Зв'язок із посадою.
date_joined	timestamptz	NOT NULL	Дата приєднання

Підсистема обліку автопарку та водіїв забезпечує контроль ресурсів компанії. Таблиця Driver (дивитись таблицю 2.3) виступає розширенням профілю користувача за принципом «один до одного», зберігаючи специфічні дані водія, а таблиця Vehicle (дивитись таблицю 2.4) фіксує дані про автомобілі.

Таблиця 2.3 – Структура таблиці Driver

Назва поля	Тип даних	Обмеження	Опис функціональності
id	int	PRIMARY KEY	Унікальний ідентифікатор
user_id	int	UNIQUE, FOREIGN KEY	Зв'язок користувача з профілем водія
license_number	varchar(50)	UNIQUE, NOT NULL	Посвідчення водія
is_active	boolean	NOT NULL	Статус активності водія

Таблиця 2.4 – Структура таблиці Vehicle

Назва поля	Тип даних	Обмеження	Опис функціональності
id	int	PRIMARY KEY	Унікальний ідентифікатор
brand	varchar(100)	NOT NULL	Марка автомобіля
model	varchar(100)	NOT NULL	Модель автомобіля
plate	varchar(20)	UNIQUE, NOT NULL	Державний номерний знак
year	integer	NOT NULL	Рік випуску автомобіля
capacity_weight	numeric	NOT NULL	Максимальна вантажопідйомність

Підсистема адрес та клієнтської бази є інформаційним довідником. Таблиця Address (дивитись таблицю 2.5) уніфікує точки відправлення та прибуття, а таблиця Contact (дивитись таблицю 2.6) містить базу осіб, що виключає дублювання інформації при оформленні квитків чи багажу.

Таблиця 2.5 – Структура таблиці Address

Назва поля	Тип даних	Обмеження	Опис функціональності
1	**2**	**3**	**4**
id	int	PRIMARY KEY	Унікальний ідентифікатор

Продовження таблиці 2.5

1	2	3	4
country	varchar(2)	NOT NULL	Код країни
city	varchar(100)	NOT NULL	Назва населеного пункту
address	varchar(255)	NOT NULL	Адреса
warehouse	varchar(100)	NULL	Номер відділення

Таблиця 2.6 – Структура таблиці Contact

Назва поля	Тип даних	Обмеження	Опис функціональності
id	int	PRIMARY KEY	Унікальний ідентифікатор
first_name	varchar(100)	NOT NULL	Ім'я клієнта
last_name	varchar(100)	NOT NULL	Прізвище клієнта
phone	varchar(20)	NOT NULL	Номер мобільного телефону

Підсистема вантажообігу та обліку багажу, таблиця Shipment (дивитись таблицю 2.7) описує індивідуальні відправлення, фіксує їхні характеристики. Модель має п'ять зовнішніх ключів. На всі ключі накладено правило SET_NULL, що захищає дані від видалення в разі очищення довідників клієнтів.

Таблиця 2.7 – Структура таблиці Shipment

Назва поля	Тип даних	Обмеження	Опис функціональності
1	2	3	4
id	int	PRIMARY KEY	Унікальний ідентифікатор
tracking_number	varchar(100)	UNIQUE, NOT NULL, INDEX	Унікальний трек-номер вантажу для відстеження.
sender_id	int	FOREIGN KEY	Посилання на відправника
recipient_id	int	FOREIGN KEY	Посилання на отримувача
from_address_id	int	FOREIGN KEY	Точка прийому вантажу
to_address_id	INT	FOREIGN KEY	Кінцева точка доставки

Продовження таблиці 2.7

1	2	3	4
amount	NUMERIC	NOT NULL	Вартість доставки
weight	NUMERIC	NOT NULL	Фактична вага вантажу
author_id	INT	FOREIGN KEY	Автор запису

Підсистема планування рейсів та квитків це центральний вузол системи. Таблиця Trip (дивитись таблицю 2.8) описує сам рейс, а кастомні проміжні таблиці TripShipment (дивитись таблицю 2.9) та TripPassenger (дивитись таблицю 2.10) реалізують зв'язки «багато до багатьох», виконуючи ролі відомості завантаженого багажу та забронbованих пасажирських місць з жорсткими обмеженнями унікальності.

Таблиця 2.8 – Структура таблиці Trip

Назва поля	Тип даних	Обмеження	Опис функціональності
id	int	PRIMARY KEY	Унікальний ідентифікатор
status	varchar(20)	NOT NULL	Поточний статус
driver_id	int	FOREIGN KEY	Призначений водій
vehicle_id	int	FOREIGN KEY	Автомобіль закріплений за рейсом.
from_address_id	int	FOREIGN KEY	Початкова точка рейсу
to_address_id	int	FOREIGN KEY	Кінцева точка рейсу
created_at	timestamptz	NOT NULL	Дата і час створення

Таблиця 2.9 – Структура таблиці TripShipment

Назва поля	Тип даних	Обмеження	Опис функціональності
1	2	3	4
id	int	PRIMARY KEY	Унікальний ідентифікатор
trip_id	int	FOREIGN KEY	Ідентифікатор рейсу

Продовження таблиці 2.9

1	2	3	4
shipment_id	int	FOREIGN KEY	Ідентифікатор посилки
loaded_at	timestamptz	NOT NULL	Час завантаження багажу

Таблиця 2.10 – Структура таблиці TripPassenger

Назва поля	Тип даних	Обмеження	Опис функціональності
id	int	PRIMARY KEY	Унікальний ідентифікатор
trip_id	int	FOREIGN KEY	Ідентифікатор рейсу
passenger_id	int	FOREIGN KEY	Ідентифікатор пасажира
from_address_id	int	FOREIGN KEY	Точка посадки пасажира
to_address_id	int	FOREIGN KEY	Точка висадки пасажира
seats_count	int	NOT NULL	Кількість місць.

Для кращого сприйняття спроєктованої структури даних, типів зв'язків, первинних та зовнішніх ключів, а також обмежень цілісності було розроблено інфологічну модель бази даних у вигляді ER-діаграми, яку наведено на рисунку 2.2., вона відображає інформацію про предметну галузь у вигляді незалежному від СУБД, що використовується [17].

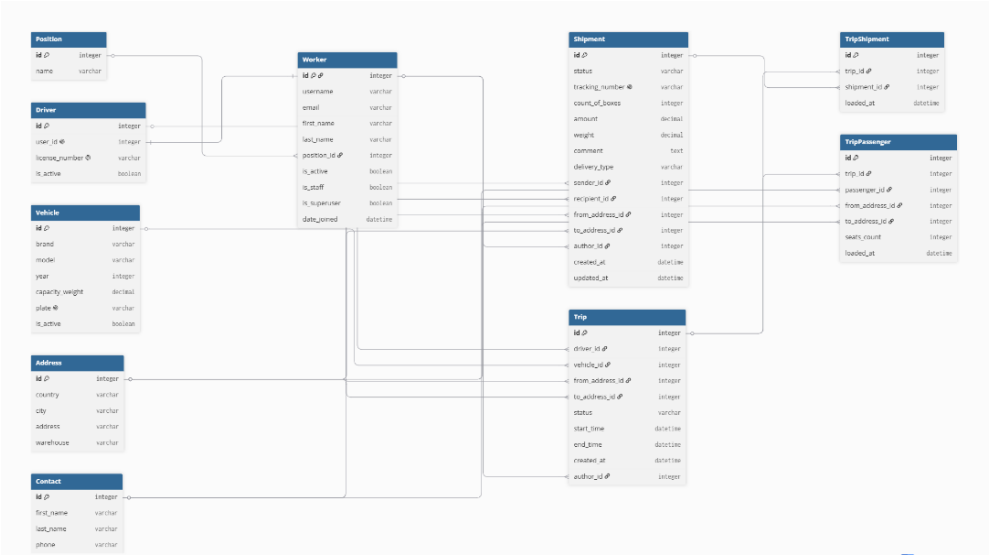


Рисунок 2.2 – ER-діаграма бази даних

Таким чином, спроєктована структура бази даних повністю охоплює всі інформаційні потоки комбінованої системи. Завдяки чіткому розподілу сутностей за п'ятьма функціональними модулями та декомпозиції зв'язків система забезпечує ізольованість даних, а впровадження каскадних правил видалення гарантує цілісність бізнес-логіки: з одного боку, система жорстко захищена від видалення активних посад чи користувачів, а з іншого — зберігає дані про виконані рейси й багаж навіть у разі очищення застарілих карток клієнтів чи адрес.

2.3 Моделювання бізнес-процесів веб-системи

Бізнес-процеси являють собою складні структури, і для їх прийнятного опису та аналізу потрібно застосувати спеціальні інформаційні засоби та методи структурного моделювання [18]. Для детального аналізу функціонування веб-системи, визначення меж її функціональності та координації взаємодії користувачів із внутрішніми компонентами системи було проведено об'єктно-динамічне та функціональне моделювання за методологією UML (Unified Modeling Language), яка є стандартною мовою моделювання для опису, проєктування та документування програмних систем [19]. Проєктування бізнес-процесів на цьому етапі дозволяє чітко трансформувати абстрактні вимоги до логістичного обліку у конкретні алгоритми виконання коду на рівні фреймворку Django та транзакцій у СУБД PostgreSQL.

Першим етапом проєктування є моделювання функціональних меж системи та розподіл прав доступу користувачів до бізнес-сутностей. Для цього було розроблено UML-діаграму прецедентів (Use Case Diagram), яку наведено на рисунку 2.3.

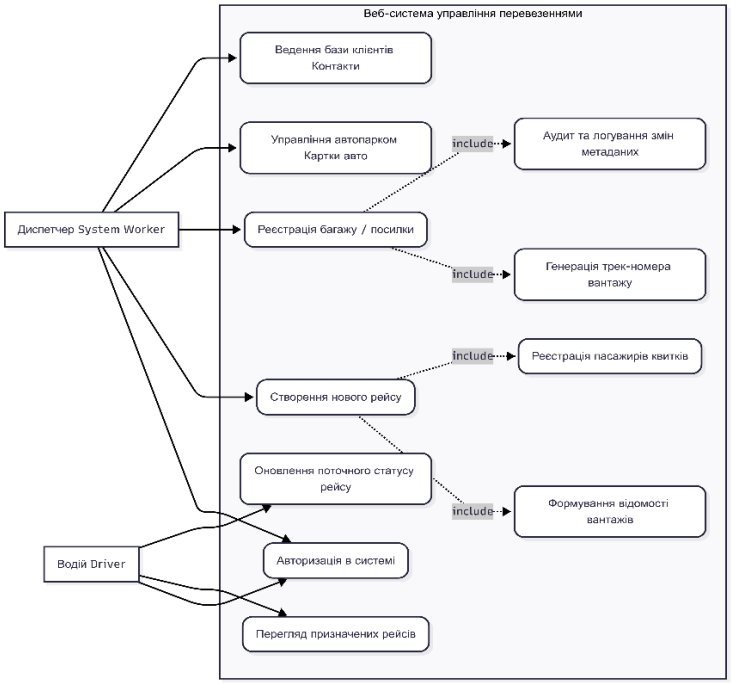


Рисунок 2.3 — UML-діаграма прецедентів веб-системи

Аналіз розробленої діаграми прецедентів дозволяє дозволяє уявити типи ролей та їх взаємодію із системою [20]. Можна виділити дві основні ролі користувачів із чітким розмежуванням їхніх повноважень, що реалізовано в коді через систему авторизації Django (групи та права доступу):

Диспетчер: виступає головним адміністративним актором системи. Він взаємодіє з програмою через вебінтерфейс та ініціює прецеденти створення й редагування маршрутів, реєстрації та маркування посилок, ведення централізованої бази клієнтів та обліку транспортних засобів компанії.

Водій: є спеціалізованим актором з обмеженими правами. Його взаємодія із сутностями системи зведена до виконання прецедентів перегляду призначених йому рейсів та оперативного оновлення статусів виконання рейсів безпосередньо під час руху за маршрутом.

У процесі роботи системи стан користувача може змінюватися залежно від його поточної активності та виконуваних дій. Зміна стану забезпечує контроль доступу до функціональних можливостей системи та відображає актуальний етап взаємодії користувача з нею [21].

Основним та найбільш навантаженим процесом у системі є процес реєстрації багажу (посилки) з наступним формуванням рейсу. Його складність полягає в необхідності автоматичної генерації унікального коду відстеження на рівні моделі, а також у забезпеченні суворого аудиту змін. У логістичних системах критично важливо фіксувати будь-які коригування параметрів вантажу (наприклад, ваги чи вартості) для запобігання зловживанням. Для цього в систему інтегровано функцію автоматичного логування історії змін метаданих полей (check_for_changes), яка відстежує стан об'єкта до і після збереження транзакції.

Часову послідовність виконання цього бізнес-процесу, взаємодію об'єктів усередині Django та алгоритм формування унікального трек-номера через метод save() моделі Shipment представлено у вигляді UML-діаграми послідовності (Sequence Diagram) на рисунку 2.4.

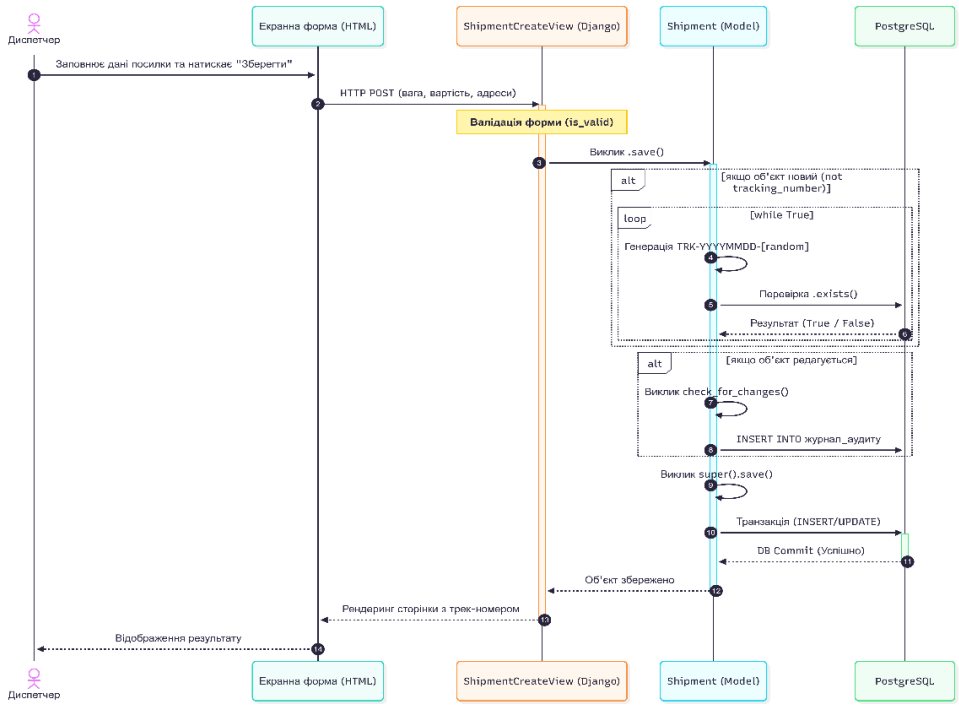


Рисунок 2.4 — UML-діаграма послідовності процесу реєстрації багажу

Детальний текстовий аналіз динаміки взаємодії об'єктів та внутрішніх компонентів системи під час виконання процесу реєстрації багажу демонструє такий покроковий алгоритм роботи:

Диспетчер через браузер заповнює екранну веб-форму створення відправлення. Він вказує вагу вантажу, фінансову вартість доставки, а також обирає з існуючих довідників адреси (точки А і Б) та контрагентів (відправника й отримувача). Після заповнення диспетчер надсилає дані, що ініціює HTTP POST-запит до сервера.

Внутрішнє представлення Django (View) приймає дані форми та запускає механізм валідації. Якщо дані коректні й відповідають типам полів у базі, ініціюється збереження екземпляра класу Shipment через виклик методу .save().

Усередині моделі Shipment керування перехоплюється перевизначеним методом save(). Спочатку система перевіряє логічну умову if not self.tracking_number. Якщо об'єкт створюється вперше, запускається цикл while True для генерації унікального коду відстеження.

Алгоритм генерації звертається до модуля timezone для отримання поточної дати у форматі YYYYMMDD та додає до неї випадкове число, згенероване функцією random.randint. Отримана строкова маска виду TRK-YYYYMMDD-[index] передається менеджеру бази даних Django ORM для швидкої перевірки.

Менеджер ORM виконує низькотранзакційний запит .exists() до унікального індексу поля tracking_number у СУБД PostgreSQL. Якщо база даних повертає відповідь True (номер уже існує), цикл продовжується і генерує новий код. Якщо повертається False, згенерований трек-номер присвоюється об'єкту, і цикл завершується за допомогою оператора break.

Перед виконанням фінального запису в базу, для об'єктів, що редагуються, запускається внутрішня функція автоматичного логування історії змін (check_for_changes). Вона порівнює поточні значення полів (наприклад, змінену ціну amount чи вагу weight) з їхніми початковими значеннями, що були завантажені з бази даних під час ініціалізації об'єкта. У разі виявлення

розбіжностей, метадані про зміни, час редагування та ідентифікатор автора фіксуються в системному журналі аудиту.

Після завершення перевірок та формування логів, метод викликає базову логіку фреймворку через super().save(), що надсилає до PostgreSQL команду INSERT (або UPDATE при редагуванні). СУБД фіксує транзакцію і повертає статус успішного виконання. Django View оновлює інтерфейс користувача, відображаючи зареєстрований вантаж із готовим унікальним трек-номером.

Для детального опису алгоритмічної логіки формування самих рейсів диспетчером та автоматичного контролю завантаження транспортних засобів було спроєктовано UML-діаграму діяльності (Activity Diagram), вона описує динамічні аспекти поведінки системи у вигляді блок-схеми, яка відображає бізнес-процеси, логіку процедур і потоки робіт — переходи від однієї діяльності до іншої [22]. Діаграму діяльності наведено на рисунку 2.5. Ця модель фокусується на розгалуженнях бізнес-логіки та перевірочних умовах, які пройде водій чи диспетчер перед тим, як рейс перейде у статус «Заплановано».

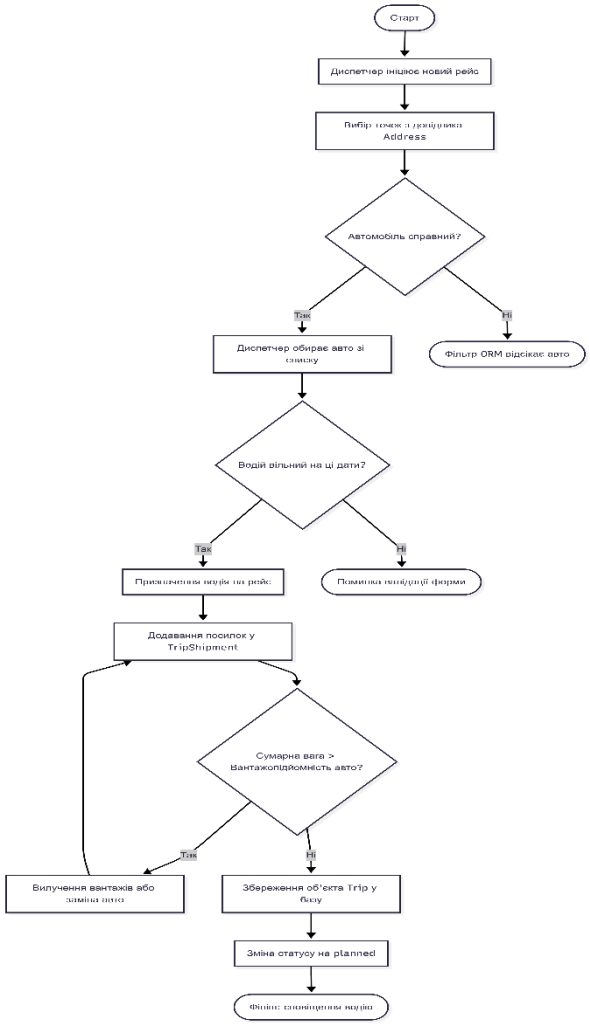


Рисунок 2.5 — UML-діаграма діяльності для процесу формування рейсів

Аналіз логіки розробленої діаграми діяльності відображає послідовність кроків та розгалужень системи під час розподілу ресурсів автопарку:

1. Диспетчер ініціює процес формування нового рейсу, обираючи початкову та кінцеву логістичні точки з довідник а Address.

2. Система пропонує обрати автомобіль. На цьому кроці вбудовані фільтри Django ORM автоматично відсікають несправні машини, залишаючи лише ті об'єкти Vehicle, які мають активний статус справності.

3. Наступним кроком диспетчер обирає водія. Система виконує логічну перевірку: через зв'язок таблиць Driver та Trip аналізується, чи не призначений обраний водій на інший рейс, часові рамки якого перетинаються з новим

маршрутом. Якщо водій зайнятий, система повертає помилку валідації форми та пропонує обрати іншого співробітника.

4. Після успішного вибору водія та авто, диспетчер починає додавати посилки до проміжної відомості TripShipment.

5. Якщо всі умови валідації виконано успішно, запис рейсу переходить у статус «Заплановано» (planned). СУБД PostgreSQL фіксує складені обмеження унікальності на квитки та багаж, а водій отримує сповіщення про призначення нового маршруту.

Таким чином спроєктований комплекс UML-діаграм дозволив трансформувати бізнес-вимоги у чітку системну архітектуру з розмежованими правами доступу користувачів, а практтична реалізація забезпечить автоматичну валідацію даних, усунення конфліктів та надійний аудит транзакцій.

2.4 Висновок до другого розділу

В другому розділі кваліфікаційної роботи було виконано повний комплекс теоретичних та конструкторських робіт із проєктування архітектури та інформаційного забезпечення веб-системи управління рейсами та багажем. На основі аналізу логістичних процесів підприємства обґрунтовано архітектурний базис програми на базі монолітної структури фреймворку Django та патерну Model-View-Template, що забезпечує максимальну швидкодію внутрішніх запитів у межах єдиного процесу додатка. Розроблено даталогічну модель бази даних у СУБД PostgreSQL, яка представлена у вигляді десяти взаємопов'язаних таблиць, приведених до третьої нормальної форми. На рівні бази даних та об'єктно-реляційного відображення впроваджено жорсткі каскадні правила видалення об'єктів та складені обмеження унікальності для проміжних таблиць рейсів. За допомогою UML-моделювання було формалізовано логіку бізнес-процесів системи, чітко розмежовано межі функціоналу. Спроєктована інформаційна структура є стійкою, оптимізованою та повністю готовою до подальшої програмної реалізації.

РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ РОБОТИ ВЕБ-СИСТЕМИ

3.1 Реалізація серверної частини та бізнес-логіки веб-системи

Програмна реалізація системи виконана на мові Python з використанням фреймворку Django та системи керування базами даних PostgreSQL. Проєкт побудовано за модульним принципом, де кожна підсистема винесена в окремий Django-застосунок. Це дозволяє ізолювати логіку різних бізнес-процесів та спрощує подальшу підтримку коду [25].

Для забезпечення безпеки та гнучкості конфігурації всі чутливі дані (секретний ключ, параметри доступу до СУБД, налаштування режимів відладки) винесені з коду в змінні оточення. Управління залежностями проєкту здійснюється через віртуальне середовище, а список необхідних бібліотек зафіксовано у файлі requirements.txt.

Взаємодія між компонентами системи організована за класичною схемою: користувач через браузер взаємодіє з клієнтською частиною, яка відправляє HTTP-запити на сервер. Маршрутизатор Django перенаправляє запит на відповідний клас представлення, який через Django ORM працює з базою даних, обробляє бізнес-логіку та повертає згенеровану сторінку клієнту [26].

Базовою архітектурною особливістю проєкту є те, що всі створені моделі даних наслідують властивості від абстрактної базової моделі HistoricalModel. Це рішення дозволяє автоматично забезпечити логування, збереження даних та відстеження історії змін для кожного об'єкта в системі без дублювання коду в окремих застосунках.

Для реалізації системи ролей (адміністратор, диспетчер, водій) стандартну модель користувача Django було розширено через успадкування класу AbstractUser. Створено кастомну модель Worker, яка містить додаткове поле зв'язку із посадою.

Контроль рівнів доступу користувачів відбувається за допомогою вбудованої системи дозволів фреймворку. Для кожного користувача призначається певний набір прав (наприклад, дозвіл на створення рейсів чи редагування карток автопарку). Прив'язка користувача до конкретної посади автоматично наділяє його відповідними правами, які перевіряються на рівні класів представлення перед виконанням операцій з даними. Програмна реалізація структури моделей користувачів та посад наведена в лістингу 3.1.

Лістинг 3.1 – Структура моделей користувачів та посад

```
class Position(HistoricalModel):
     name = models.CharField(max_length=255)

     class Meta:
          verbose_name = "Посада"
          verbose_name_plural = "Посади"
          ordering = ["name"]
     def __str__(self) -> str:
          return self.name

class Worker(AbstractUser):
     position = models.ForeignKey(Position, related_name="workers",
on_delete=models.PROTECT, null=True)

     def __str__(self) -> str:
          return self.username
     class Meta:
          verbose_name = "Працівник"
          verbose_name_plural = "Працівники"
```

Логіка обліку транспортних засобів та водіїв реалізована на основі двох взаємопов’язаних моделей: Driver та Vehicle.

Модель Driver виступає розширенням для облікового запису працівника і пов’язана з моделлю Worker гарантуючи, що один системний профіль користувача може мати лише один профіль водія. У моделі передбачено збереження унікального номера посвідчення водія та прапорця для оперативного відсторонення водія. Модель Vehicle зберігає в собі не тільки базові текстові поля марки, моделі та номерного знака автомобіля, а й критично важливі для логістичних розрахунків поля для збереження вантажопідйомності та

пасажиромісткості транспортних засобів. Програмна реалізація структури моделей транспортних засобів та водіїв наведена в лістингу 3.2.

Лістинг 3.2 – Структура моделей транспортних засобів та водіїв

```
class Driver(HistoricalModel):
    user = models.OneToOneField(User, on_delete=models.CASCADE,
verbose_name='Працівник')
    license_number = models.CharField(max_length=50, unique=True,
verbose_name="Номер ліцензії")
    is_active = models.BooleanField(default=True,
verbose_name="Активний")

    def __str__(self):
        full_name = self.user.get_full_name()
        return full_name if full_name else self.user.username

    class Meta:
        verbose_name = "Водій"
        verbose_name_plural = "Водії"

class Vehicle(HistoricalModel):
    brand = models.CharField(max_length=255, verbose_name="Бренд")
    model = models.CharField(max_length=255,
verbose_name="Модель")
    year = models.PositiveIntegerField(verbose_name="Рік випуску")
    capacity_weight = models.DecimalField(max_digits=8,
decimal_places=2, null=True, blank=True,
verbose_name="Вантажопідйомність")
    seats_count = models.IntegerField(verbose_name="Кількість
місць")
    is_active = models.BooleanField(default=True)
    plate = models.CharField(max_length=20, unique=True,
verbose_name="Номерний знак")

    def __str__(self) -> str:
        return f'{self.brand} {self.model} {self.plate}'

    class Meta:
        verbose_name = "Автомобіль"
        verbose_name_plural = "Автомобілі"
```

Інформація про контрагентів та адреси зберігається за допомогою моделей Contact та Address. Призначенням даних моделей є збереження контактної інформації та логістичних даних клієнтів для їх відображення як частину даних про посилку або рейс.

За допомогою моделі Shipment відбувається облік індивідуальних клієнтських відправлень. Модель оперує фіксованими статусами виконання («Створено», «Доставляється», «Доставлено») та типами доставки («За адресою», «Поштою»). Також в моделі реалізовано зв’язки з клієнтами та адресами. У моделі реалізовано алгоритм автоматичного формування унікального трек-номера, призначеного для кодування у штрихкод для маркування посилок та їх пошуку в системі. Логіка зашита безпосередньо у перевизначений метод збереження моделі. При першому збереженні об'єкта код генерує штамп дати у форматі «РРРРММДД» та запускає цикл, який за допомогою модуля random підбирає випадкове число [27]. Запит перевіряє базу даних на збіг, якщо номер вільний, цикл переривається, а сформований рядок фіксується за об'єктом. Програмний код методу збереження наведено в лістингу 3.3.

Лістинг 3.3 – Структура методу збереження моделі «Shipment»

```
    def save(self, *args, **kwargs):
        if not self.tracking_number:
            date_str = timezone.now().strftime("%Y%m%d")
            while True:
                random_code = random.randint(1000, 9999)
                potential_track = f"TRK-{date_str}-{random_code}"
                if not
Shipment.objects.filter(tracking_number=potential_track).exists():
                    self.tracking_number = potential_track
                    break
        super().save(*args, **kwargs)
```

Алгоритм обробки запиту на створення штрих-коду базується на отриманні унікального номера через динамічний параметр URL-адреси. Бібліотека barcode трансформує текстовий трек-номер посилки у графічне зображення формату PNG.

У результаті виконання клас представлення формує відповідь, яка ініціює автоматичне завантаження створеного файлу на пристрій користувача як вкладення з відповідним іменем. Це дозволяє миттєво отримати готовий до друку

та маркування файл етикетки вантажу. Повну програмну реалізацію цього процесу наведено в лістингу 3.4.

Лістинг 3.4 – Структура класу представлення для генерації штрих-коду

```
    class GenerateBarcodeView(LoginRequiredMixin,
PermissionRequiredMixin, generic.View):
    permission_required = "shipments.change_shipment"

    def get(self, request, barcode_number, *args, **kwargs):
        if not barcode_number:
            raise Http404("Номер для штрихкоду не вказано.")
        barcodes_dir = os.path.join(settings.MEDIA_ROOT,
'barcodes')
        if not os.path.exists(barcodes_dir):
            os.makedirs(barcodes_dir, exist_ok=True)
        file_path = os.path.join(barcodes_dir,
str(barcode_number))

        options = {
            'write_text': True,
            'module_height': 15.0,
            'module_width': 0.3,
            'font_size': 10,
            'text_distance': 4.0
        }

        CODING = barcode.get_barcode_class('code128')
        code_instance = CODING(str(barcode_number),
writer=ImageWriter())

        saved_path = code_instance.save(file_path,
options=options)

        response = FileResponse(
            open(saved_path, 'rb'),
            as_attachment=True,
            filename=f"{barcode_number}.png"
        )
        return response
```

Важливим етапом є консолідація даних з різних модулів у єдину модель, що забезпечує цілісне управління бізнес-логікою системи. Ядро всієї бізнес-логіки веб-системи консолідує в собі об'єкти з усіх інших модулів у єдині комбіновані маршрути [23][24]. Головна модель Trip містить дані про призначення водія, автомобіля, початкову і кінцеву точки маршруту, поточний

статус рейсу, часові позначки інформацію про посилки та пасажирів на рейсі. Для реалізації структури розроблено дві проміжні моделі: TripShipment яка прив'язує посилки до конкретного рейсу та TripPassenger яка відповідає за посадку пасажирів.

Для забезпечення взаємодії з базою даних та виконання операцій над рейсами розроблено систему спеціалізованих форм, що наслідуються від базового класу forms.ModelForm. Форма TripForm відповідає за базові параметри маршруту, призначення водія та транспортного засобу. Особливістю реалізації цієї форми є перевизначення методу __init__, у якому за допомогою об'єкта Q-запитів налаштовується динамічна фільтрація доступного персоналу та техніки. Програма аналізує стан прапорця активності моделей Vehicle та Driver, виводячи у випадаючі списки лише працездатні. При цьому, якщо форма відкривається у режимі редагування вже існуючого рейсу, поточний призначений автомобіль або водій залишаються у списку за допомогою логічного оператора «АБО», навіть якщо їхній статус активності було змінено в базі даних після старту перевезень. Програмну структуру форми наведено в лістингу 3.5.

Лістинг 3.5 – Структура форми рейсу

```
class TripForm(forms.ModelForm):
    class Meta:
        model = Trip
        fields = ["driver", "vehicle", "from_address",
"to_address", "status", "start_time", "end_time"]

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        if self.instance and self.instance.pk and
self.instance.vehicle:
            self.fields['vehicle'].queryset =
Vehicle.objects.filter(
                Q(is_active=True) | Q(pk=self.instance.vehicle.pk)
            )
        else:
            self.fields['vehicle'].queryset =
Vehicle.objects.filter(is_active=True)

        if self.instance and self.instance.pk and
self.instance.driver:
```

```
            self.fields['driver'].queryset =
Driver.objects.filter(
                Q(is_active=True) | Q(pk=self.instance.driver.pk)
            )
        else:
            self.fields['driver'].queryset =
Driver.objects.filter(is_active=True)
```

Візуалізація та обробка бізнес-логіки рейсів реалізована на базі класів представлення. Клас представленя TripListView наведений в лістингу 3.6 забезпечує виведення загального переліку рейсів із вбудованою пагінацією по 5 записів на сторінку та підтримкою GET-фільтрації за поточним статусом або за належністю конкретному водію.

Лістинг 3.6 – Клас представлення сторінки списку рейсів

```
    class TripListView(LoginRequiredMixin,
PermissionRequiredMixin, generic.ListView):
    model = Trip
    context_object_name = "trip_list"
    queryset = Trip.objects.all().order_by('created_at')
    template_name = "trips/trip_list.html"
    paginate_by = 5
    permission_required = "trips.view_trip"

    def get_queryset(self):
        queryset = super().get_queryset()
        status = self.request.GET.get('status')
        if status:
            queryset = queryset.filter(status=status)
        scope = self.request.GET.get('scope')
        if scope == 'my':
            if self.request.user.is_authenticated and
hasattr(self.request.user, 'driver'):
                queryset =
queryset.filter(driver=self.request.user.driver)
            else:
                return Trip.objects.none()
        return queryset
```

Центральним вузлом обробки поточних процесів є клас представлення детального перегляду TripDetailView. Окрім стандартного рендерингу інформації про маршрут, метод get_context_data динамічно насичує сторінку п'ятьма різними формами, що дозволяє диспетчеру в межах одного робочого простору редагувати параметри рейсу, додавати наявних або створювати нових

пасажирів і посилки. Метод post цього класу представлення виконує роль диспетчера запитів, аналізуючи вміст POST-даних за ключовими кнопками.

При натисканні кнопки збереження рейсу виконується оновлення основних полів моделі Trip, після чого запускається цикл валідації та актуалізації заброньованих пасажирських місць. Система зчитує динамічні поля введення update_seats_<id> для кожного об'єкта TripPassenger. Перед безпосереднім фіксуванням змін у базі даних через метод item.save(), сервер проводить перевірку місткості салону автомобіля: сума всіх оновлених значень seats_count для пасажирів цього рейсу порівнюється з значенням seats_count моделі Vehicle, призначеної на рейс [28]. Якщо ліміт вільних місць перевищено, операція блокується з виведенням повідомлення про помилку, що виключає ризик перевантаження транспортного засобу. Аналогічні алгоритми перевірки та умовного розгалуження прописані для блоків додавання та каскадного видалення (remove_passenger, remove_shipment) сутностей з проміжних таблиць зв'язку. Повну програмну реалізацію класу представлення детальної картки рейсу наведено в Додатку А.

Програмна реалізація серверної частини веб-системи забезпечує надійну роботу її структури завдяки побудові чіткої архітектури моделей, форм та класів представлень, при цьому наведені вище лістинги відображають лише ключові фрагменти розробки. Усі сутності проєкту мають реалізовані сторінки та інтерфейси для виконання всього спектра базових CRUD-операцій. Використання штатних дозволів Django для контролю доступу, перевизначення конструкторів форм для динамічної фільтрації активного транспорту, а також впровадження алгоритмів генерації штрихкодів та кастомної валідації лімітів місткості автомобілів гарантують високу безпеку, захищеність від помилок та повну відповідність бекенд-логіки реальним процесам підприємства.

3.2 Розробка інтерфейсу користувача

Для забезпечення взаємодії користувачів з серверною частиною розроблено адаптивний інтерфейс користувача, побудований на базі вбудованого двигуна шаблонів Django Templates та CSS-фреймворку Bootstrap Головним завданням проєктуванні фронтенду було створення єдиного робочого простору для диспетчера, що мінімізує кількість переходів між сторінками при роботі з веб-системою. Клієнтська логіка реалізує концепцію динамічного відображення елементів, де складні форми розбиті на візуальні блоки, а керування зв'язаними сутностями відбувається через інтеграцію модальних вікон та асинхронних елементів керування.

Уніфікація візуального стилю та структури веб-сторінок платформи досягається завдяки використанню базового шаблону base.html [29]. У ньому визначено загальну структуру, підключено зовнішні стилі та кастомні файли оформлення. Компонування сторінки реалізовано за допомогою гнучкої двокомпонентної структури, яка складається з глобальної шапки, бічного навігаційного меню та центральної робочої зони.

Важливим елементом контролю доступу на рівні інтерфейсу користувача є бічна панель навігації sidebar.html. Її реалізація базується на перевірці системних дозволів користувача через вбудований контекстний процесор perms. Замість повного відображення всіх пунктів меню для будь-якого відвідувача, шаблонізатор Django індивідуально аналізує права доступу працівника перед рендерингом кожного блоку. Наприклад, пункти «Рейси» чи «Посилки» відображаються лише за умови наявності відповідних дозволів, а адміністративні розділи примусово приховуються від лінійних водіїв або складських операторів. Загальний вигляд головного екрана веб-системи з активованою панеллю навігації наведено на рисунку 3.1.

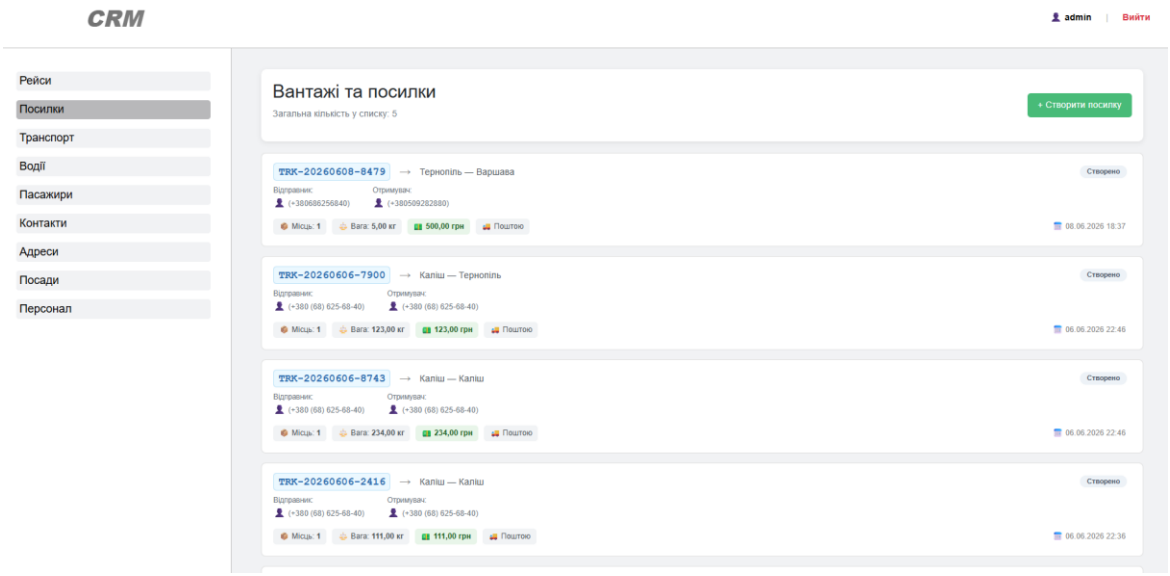


Рисунок 3.1 – Головний екран веб-системи

Центральним інтерфейсом системи є сторінка детального перегляду та редагування рейсу, реалізована в шаблоні trip_detail.html. Замість розгалуження логіки на декілька окремих сторінок, цей інтерфейс об'єднує в межах однієї HTML-форми trip-form базові параметри маршруту, призначення водія, транспортного засобу та дві деталізовані таблиці для моніторингу пасажирів і вантажів. Доступ до функцій модифікації об'єкта інтегровано на рівні інтерфейсу: кнопки переходу в режим редагування, скасування та збереження змін рендеряться лише у випадку наявності права change_trip, тоді як критична кнопка видалення посилки захищена умовою delete_trip.

Елементи введення основної інформації та маршруту за замовчуванням заблоковані атрибутом disabled. Активація полів для внесення змін, динамічна зміна видимості кнопок керування, а також виведення прихованих стовпців каскадного видалення зв'язаних сутностей реалізуються на стороні клієнта за допомогою JavaScript-сценарію edit-mode.js, який оперує кастомними data-атрибутами елементів форми. Візуальне представлення деталізованої картки рейсу в режимі перегляду зображено на рисунку 3.2.

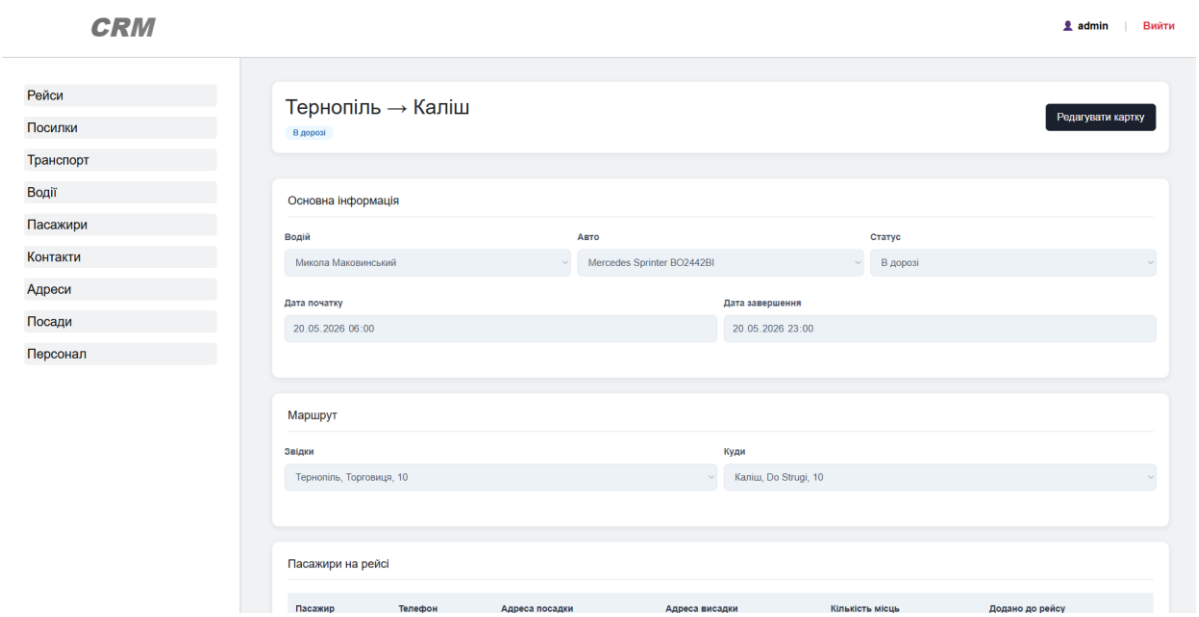


Рисунок 3.3 – Інтерфейс картки рейсу в режимі перегляду даних

Табличні блоки «Пасажири на рейсі» та «Вантажі та посилки» динамічно виводять пов'язані списки об'єктів через циклічні конструкції {% for %}. Особливістю блоку пасажирів є наявність інпуту update_seats_{{ item.id }}, який дозволяє диспетчеру коригувати кількість місць для кожного пасажира безпосередньо в таблиці під час редагування картки рейсу. Для оперативного додавання клієнтів та вантажів у нижній частині кожної секції реалізовано інтерактивні блоки пошуку на базі тегу <datalist>. Диспетчер може почати введення номера телефону пасажира або трек-коду посилки, а система запропонує варіанти автоматичного підбору з наявних в базі даних.

Якщо потрібного контрагента або вантажу немає в базі, інтерфейс підтримує логіку створення об'єктів «на льоту» за допомогою модальних вікон passenger-modal та shipment-modal. Ці вікна рендеряться в тілі сторінки умовним тегом {% if perms.trips.change_trip %} і містять форми пасажирів та посилок, що забезпечує безперервність процесу створення рейсів диспетчером. Зовнішній вигляд табличної частини та інтерфейс пошуку і додавання сутностей наведено на рисунку 3.3. Повну програмну реалізацію комбінованого шаблону картки рейсу наведено в Додатку Б.

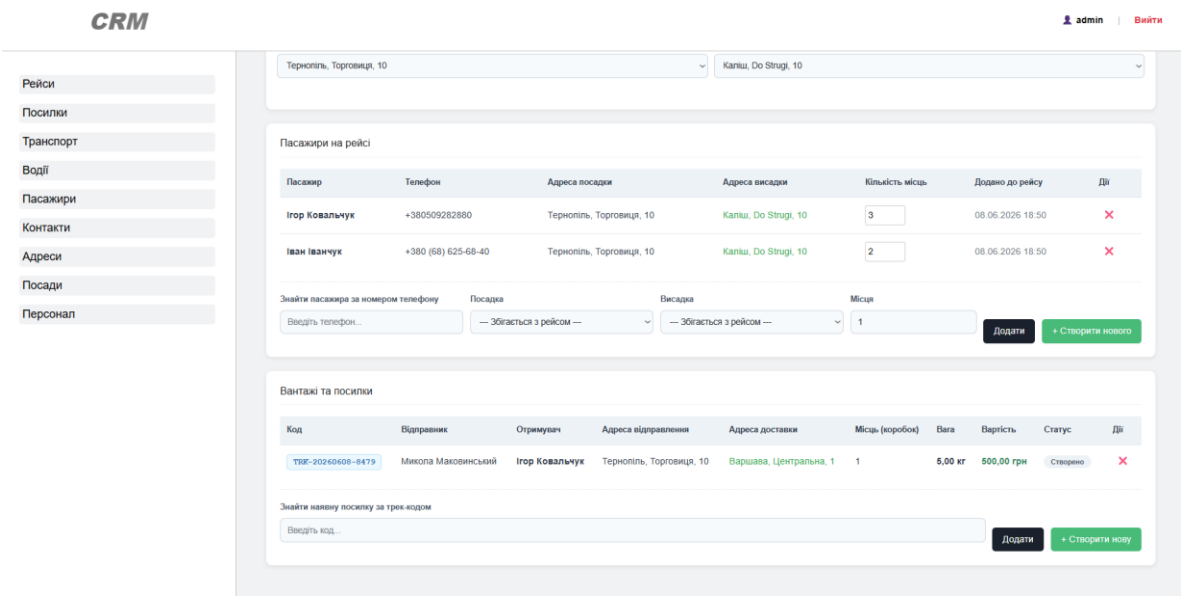


Рисунок 3.4 – Керування зв'язаними пасажирами та вантажами рейсу

Таким чином, розробка користувацького інтерфейсу на базі шаблонізатора Django Templates та компонентів Bootstrap дозволила створити цілісне та захищене середовище, при цьому наведені лістинги демонструють лише базові принципи побудови та інтеграції компонентів, тоді як інші екземпляри сторінок системи (зокрема інтерфейси карток посилок) використовують ідентичні механізми взаємодії. Впровадження перевірок дозволів безпосередньо у процесі рендерингу бічної панелі та екранних форм виключає появу невалідних елементів керування, а інтеграція JavaScript-сценарію для динамічного перемикання атрибутів блокування полів дозволила поєднати режими перегляду та редагування в межах однієї сторінки. Це суттєво підвищує ергономічність системи, знижує навантаження на мережу та забезпечує оперативність роботи диспетчерської служби підприємства.

3.3 Тестування функціональності та аналіз результатів розробки

Для підтвердження працездатності веб-системи, контролю коректності взаємодії клієнтської та серверної частин, а також перевірки відповідності реалізованого функціоналу поставленим технічним вимогам було проведено

ручне тестування. Оскільки веб-система оперує критично важливими бізнес-процесами підприємства, метою цього етапу є повна симуляція дій диспетчерів, складських операторів та водіїв для виявлення можливих логічних помилок, перевірки стійкості інтерфейсу та контролю виконання бізнес-правил безпосередньо через екранні форми.

Процес мануального тестування розпочався з перевірки сторінок списків на прикладі інтерфейсу перегляду рейсів. Під час тестування екрана TripListView було детально перевірено логіку динамічної GET-фільтрації даних та роботу механізму пагінації. Виконано ручний вибір фільтра за поточним логістичним статусом рейсу. Перевірка підтвердила, що клас представлення коректно модифікує SQL-запит до бази даних, роблячи вибірку лише відповідних запитів. Графічний інтерфейс платформи під час виконання фільтрації та відображення списку рейсів списків рейсів наведено на рисунку 3.5.

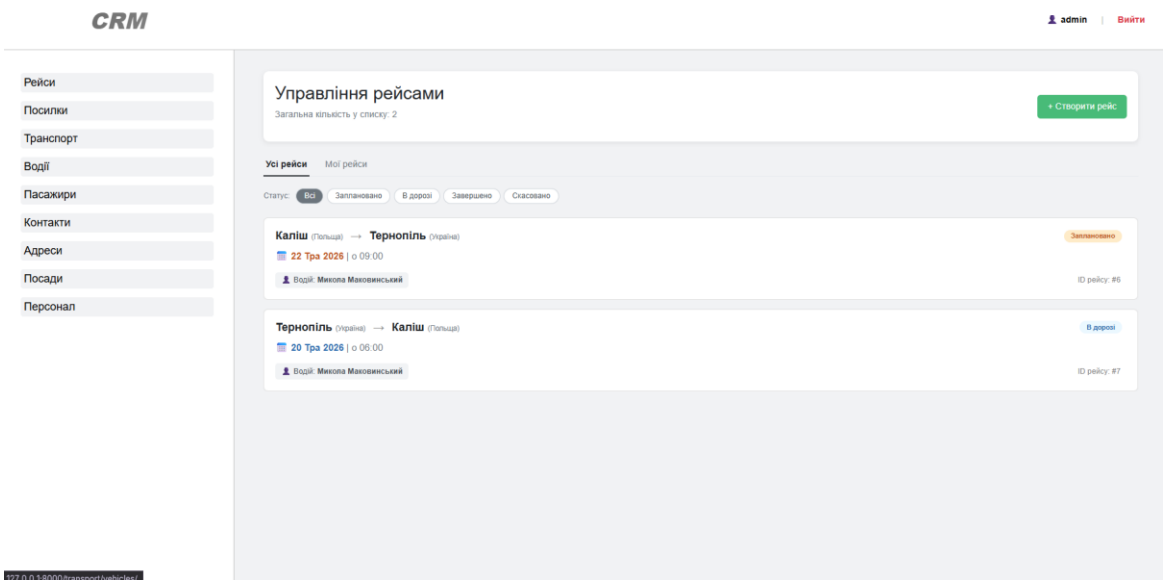


Рисунок 3.5 — Інтерфейс сторінки списків рейсів з активним фільтром

Наступний етап тестування — перевірка режимів створення сутностей та відкладеного зв'язування даних «на льоту». Під час заповнення основної форми рейсу перевірено роботу конструктора форми, який автоматично виключив із

випадаючих списків неактивних водіїв. Зовнішній вигляд інтерфейсу в момент створення наведено на рисунку 3.6.

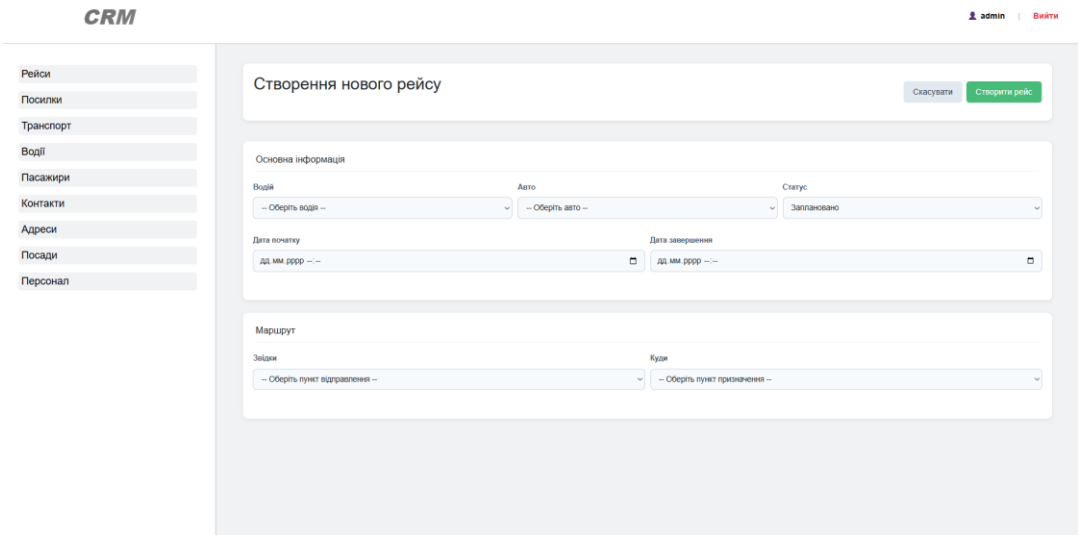


Рисунок 3.6 — Створення та додавання об'єктів через модальні вікна

Додатково перевірено працездатність підсистеми автоматичних поштових сповіщень. За сценарієм, у момент фіксації нового рейсу або при внесенні критичних змін у його параметри бекенд-архітектура ініціює тригер генерації та відправки email-повідомлень на поштові скриньки відповідного персоналу. Візуальне відображення згенерованих та успішно відправлених системою email-листів наведено на рисунку 3.7.

Наступний етап перевірки став контроль виконання логіки динамічного редагування картки посилки та підсистеми генерації штрихкодів. Обравше вже створену посилку та перевівши її в режим редагування за допомогою кнопки «Редагувати картку», внісено зміни у фінансові поля та успішно збережено їх за допомогою кнопки зберегти, яка відправляє відповідний POST-запит. Візуальне відображення інтерфейсу системи в момент переходу до режиму модифікації

Після цього виконано перевірку генерації штрих-коду шляхом натиску на кнопку інтерактивну кнопку «Штрих-код». Клас представлення успішно перехопив трек-номер посилки та згенерував графічний файл зберігши його у медіа-директорію сервера та миттєво видав у браузер грфічну відповідь.

Перевірка підтвердила, що клієнтське середовище автоматично та без помилок ініціювало завантаження готового до друку PNG-зображення. Візуальне відображення згенерованого штрих-коду наведено на рисунку 3.8.

Рисунок 3.8 — Вигляд згенерованого штрих-коду.

Варто підкреслити, що тут наведено опис тестування лише найбільш відповідальних візуальних та логічних вузлів платформи. Усі інші компоненти веб-інтерфейсу, включаючи форми додавання адрес, роботу пошукових інпутів та видалення застарілих карток, пройшли аналогічну перевірку. У результаті тестування не було виявлено збоїв у логіці обробки запитів чи дефектів рендерингу, що підтверджує повну готовність веб-системи. Знімки екрану веб-системи наведені в Додатку В.

3.4 Висновок до третього розділу

В третьому розділі кваліфікаційної роботи виконано повний цикл практичної реалізації та тестування веб-системи управління рейсами та багажем для транспортної компанії. На основі застосування сучасного технологічного стеку, вдалося побудувати надійну модульну архітектуру бекенду з високим рівнем безпеки та гнучким керуванням доступом на базі кастомної моделі користувача. Інтеграція логування історії змін забезпечила надійний аудит даних, а розроблені інженерні алгоритми динамічної фільтрації ресурсів через

об'єкти Q-запитів та автоматичного формування унікальних трек-номерів із генерацією графічних штрих-кодів дозволили повністю автоматизувати рутинні операції диспетчерської служби.

Створення адаптивного користувацького інтерфейсу дозволило об'єднати режими перегляду й редагування даних в межах єдиного робочого простору, що суттєво підвищило ергономічність системи та швидкість обробки інформації. Проведене комплексне тестування всіх розроблених модулів, включаючи перевірку підсистем GET-фільтрації, пагінації, каскадного оновлення зв'язаних сутностей та серверної валідації лімітів місткості автомобілів, підтвердило повну працездатність платформи. Відсутність логічних помилок і дефектів рендерингу інтерфейсу свідчить про стійкість веб-системи і повну готовність до успішного впровадження.

РОЗДІЛ 4. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Порядок дій при виникненні пожежі

Виходячи з сучасних уявлень, безпека життєдіяльності є багатогранним об’єктом розуміння і сприйняття дійсності, який потребує інтеграції різних стратегій, сфер, аспектів, форм і рівнів пізнання [30]. Практично щодня людина знаходиться під загрозою великої кількості небезпек, пов’язаних із вирішенням різних завдань [31]. Оскільки апаратні засоби створеної логістичної CRM-системи (персональні комп'ютери, монітори, комутаційні пристрої) живляться від електричної мережі напругою 220 В, існує потенційна небезпека виникнення пожежі. Основними причинами займання на робочому місці користувача можуть бути короткі замикання в блоках живлення, перевантаження електромережі через підключення занадто великої кількості периферійних пристроїв в один подовжувач, або термічне перегрівання елементів системного блока через накопичення пилу.

Пожежа в офісному або житловому приміщенні з комп'ютерною технікою несе пряму загрозу життю працівника (критерій БЖД) через швидке поширення токсичних продуктів горіння пластику та ізоляції (чадного газу) і обмеження видимості [32]. Для мінімізації ризиків та захисту персоналу пропонується чіткий інженерно-організаційний алгоритм дій у разі виявлення ознак займання.

Покроковий алгоритм дій при виникненні пожежі:

1. Екстрене реагування та оповіщення

У разі виявлення перших ознак пожежі (запах горілої ізоляції, дим, іскріння провідників або відкрите полум'я) користувач повинен діяти за такою схемою:

- Оцінка ситуації: негайно припинити роботу з CRM-системою (за можливості швидко зберегти критичні дані).
- Виклик рятувальної служби: терміново зателефонувати до Служби порятунку за номером 101. При повідомленні необхідно чітко вказати: точну

адресу об'єкта, поверх, номер кімнати, місце виникнення займання (наприклад, загорівся системний блок комп'ютера), а також назвати своє прізвище.

- Оповіщення оточуючих: гучно повідомити про небезпеку всіх колег у приміщенні або задіяти ручний сповіщувач пожежної сигналізації (якщо такий є на виході).

2. Локалізація та гасіння пожежі на початковій стадії

Гасіння займання силами самого користувача дозволяється лише за умови, що пожежа є незначною (локальною), а її ліквідація не загрожує життю та здоров'ю працівника.

- Обов'язкове знеструмлення: перед початком гасіння категорично заборонено лити воду на обладнання. Необхідно негайно висмикнути шнур живлення комп'ютера з розетки або повністю знеструмити кімнату за допомогою загального автоматичного вимикача (пакетника) у щитку.
- Вибір засобу пожежогасіння: для ліквідації займання комп'ютерної техніки та електропроводки під напругою рекомендується використовувати лише вуглекислотні (ВВ) або порошкові (ВП) вогнегасники. Вуглекислотний вогнегасник є пріоритетним, оскільки він не пошкоджує плати комп'ютера та ефективно збиває полум'я за рахунок охолодження зони горіння.
- Техніка безпеки при гасінні: при використанні вуглекислотного вогнегасника необхідно пам'ятати, що не можна триматися голими руками за розтруб, оскільки його температура падає до -70°С, що може викликати обмороження шкіри.

3. Безпечна евакуація з небезпечної зони

Якщо з вогнем не вдалося впоратися протягом перших 30–40 секунд, або якщо кімната почала стрімко наповнюватися димом, гасіння слід негайно припинити та розпочати евакуацію:

- Маршрут руху: залишити кімнату, щільно закривши за собою двері (але не замикаючи їх на замок), щоб обмежити доступ кисню до вогнища і стримати поширення диму в коридор. Рухатися відповідно до затвердженого плану евакуації будівлі до основних або запасних виходів.

- Захист органів дихання: оскільки основна небезпека при пожежі в комп'ютерному класі чи офісі — це токсичні гази від пластику, дихати рекомендується через вологу тканину (серветку, хустинку, одяг) [33].
- Правила переміщення: оскільки чисте повітря та краща видимість зберігаються ближче до підлоги, при сильному задимленні коридорів необхідно пересуватися навприсядки або повзком, тримаючись руками за стіни.
- Категорична заборона: під час пожежі в багатоповерхових офісних будівлях заборонено користуватися ліфтами, оскільки при аварійному вимкненні електрики є ризик опинитися в пастці та задихнутися. Евакуація здійснюється виключно сходовими клітками.

Після виходу з будівлі користувач повинен перебувати у безпечному місці, пройти перевірку за списками співробітників та зустріти пожежну бригаду, щоб чітко вказати їм схему приміщень і місце виникнення пожежі.

4.2 Естетичне оформлення та ергономічне дослідження робочого місця оператора

Робота користувача (диспетчера або розробника) з веб-системою пов'язана з тривалим перебуванням у сидячому положенні та постійною увагою на екран монітора. Це створює навантаження на зір, хребет та м'язи рук [34]. Для запобігання втомі та збереження здоров'я працівника, організацію робочого місця виконано відповідно до вимог НПАОП 0.00-7.15-18 «Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями».

Для забезпечення комфортного та правильного положення тіла за комп'ютером пропонується встановити такі нормативні параметри елементів робочого місця:

Робочий стіл:

- Висота робочої поверхні столу повинна становити 725 мм.

- Стільниця повинна мати ширину не менше 1200 мм та глибину від 800 мм, що дозволить вільно розмістити монітор, клавіатуру, мишу та робочі записи.
- Простір для ніг під столом необхідно забезпечити у таких межах: висота — 650 мм, ширина — 500 мм, глибина — 450 мм.
- Поверхня столу має бути матовою (сірого або деревоподібного відтінку), щоб виключити появу бліків від ламп.

Робоче крісло:

- Рекомендується використання підйомно-поворотного крісла на коліщатках.
- Висота сидіння має регулюватися під зріст працівника в межах 400–530 мм. Передній край сидіння повинен мати заокруглену форму, щоб не перетискати судини на ногах.
- Спинка крісла повинна мати анатомічну форму з підтримкою попереку. Кут нахилу спинки рекомендується встановити на рівні 105°для зниження навантаження на хребет.
- Наявність підлокітників має забезпечувати положення ліктів під кутом близько 90°, що знімає напругу з рук і запобігає появі болю в зап'ястях.

Розміщення монітора:

- Монітор необхідно розміщувати прямо перед користувачем на відстані 650 мм від очей.
- Верхня кромка екрана повинна знаходитися на рівні очей користувача. Це забезпечує оптимальний кут огляду (15–20° донизу), завдяки чому не напружуються м'язи шиї та менше втомлюються очі.

Кольорове та естетичне оформлення робочої кімнати має бути спрямоване на створення сприятливого психологічного клімату:

Кольорова архітектура та виробниче середовище приміщення, де експлуатується веб-система, безпосередньо формують емоційне тло, знижують монотонність логістичних операцій та підтримують тонус нервової системи [35]:

- Стіни кімнати рекомендується фарбувати матовими акриловими фарбами спокійних пастельних тонів із високим коефіцієнтом розсіяння (світло-бежевий, кремовий або блідо-зелений кольори). Такі відтінки нормалізують кров'яний тиск і зменшують психологічне напруження.
- Стеля повинна бути виключно білою для забезпечення рівномірного перевідбиття та розсіювання світлового потоку від систем верхнього освітлення.
- Вимоги до освітленості: робочі місця користувачів комп'ютерної техніки повинні бути забезпечені штучним загальним або комбінованим освітленням з рівнем освітленості не менше 300 лк на поверхні столу відповідно до чинних будівельних норм. Джерела світла повинні розташовуватися так, щоб на екрані монітора не виникало прямих або відбитих світлових бліків.
- Естетичні чинники та озеленення: корпуси комп'ютерної техніки та периферії повинні мати нейтральні матові кольори (темно-сірий, чорний), які візуально зливаються з робочим простором і не відволікають від сприйняття інтерфейсу CRM. В інтер'єрі приміщення доцільно організувати зони вертикального або настільного фітодизайну з використанням живих широколистих рослин (наприклад, хлорофітум, спатифіллум). Вони не лише покращують естетичне сприйняття робочої зони, але й абсорбують дрібнодисперсний пил і сприяють підтримці оптимальної відносної вологості повітря у межах нормованих 40–60%

Таким чином, запропонований комплекс просторово-ергономічних та естетичних рішень для організації робочого місця оператора веб-системи має чітко виражений превентивний характер. Дотримання наведених геометричних параметрів меблів, правильне налаштування зорової зони монітора, а також оптимізація світлотехнічного та кольорового клімату кімнати дозволяють мінімізувати шкідливий вплив тривалої статичної пози і зорової напруги. Впровадження цих рекомендацій у комплексі забезпечує збереження здоров'я працівника, знижує ризик розвитку професійних захворювань опорно-рухового апарату та підтримує стабільно високу продуктивність праці протягом усієї робочої зміни.

ВИСНОВКИ

У кваліфікаційній роботі вирішено завдання з проєктування, програмної реалізації та тестування веб-системи управління рейсами та багажем для транспортної компанії. Розроблене програмне забезпечення дозволяє повністю відмовитися від неефективного паперового обліку, консолідувати дані про пасажирів та вантажі в єдиній реляційній базі даних, автоматично контролювати ліміти місткості транспорту, маркувати посилки штрих-кодами та оперативно сповіщати персонал. Практична цінність результатів полягає у створенні готового до розгортання веб-сервісу, що підвищує швидкість обробки замовлень та мінімізує ризики логістичних помилок.

В першому розділі кваліфікаційної роботи освітнього рівня «Бакалавр»:

– Подано опис бізнес-процесів операційного обліку, планування рейсів та логістики супутнього багажу в сучасних компаніях міжнародних і міжміських пасажирських перевезень.

– Розглянуто та обґрунтовано вибір технологічного стеку для розробки платформи, який включає високорівневий фреймворк Django (мова Python), реляційну СУБД PostgreSQL та інструменти побудови адаптивного frontend-інтерфейсу (Bootstrap, JavaScript).

– Висвітлено критичні недоліки та ризики ручного ведення обліку, що призводять до втрати даних, перевантаження транспорту та логістичних збоїв.

– Проаналізовано існуючі комерційні програмні аналоги на ринку, доведено їхню негнучкість щодо специфіки роботи приватних перевізників та сформовано детальне технічне завдання на розробку власної веб-системи.

В другому розділі кваліфікаційної роботи:

– Досліджено структуру даних предметної області та розроблено концептуальну інфологічну ER-модель бази даних, розділену на логічні блоки: транспорт, адреси, рейси, пасажири та посилки.

– Обґрунтовано архітектурну побудову веб-системи на основі патерну MVT, що забезпечує чіткий розподіл бізнес-логіки, представлення та взаємодії з даними.

– Сформовано комплекс UML-діаграм (прецедентів, послідовності та діяльності) для детального моделювання сценаріїв взаємодії користувачів із внутрішніми компонентами системи та автоматизації перевірки стану автопарку.

В третьому розділі кваліфікаційної роботи:

– Розроблено серверну частину веб-додатку на базі Django ORM із реалізацією механізмів автентифікації користувачів, вбудованого захисту даних від вразливостей та каскадного контролю цілісності зв’язків у БД.

– Запропоновано та інтегровано підсистему автоматичної графічної генерації унікальних штрих-кодів для маркування прийнятих посилок та асинхронний модуль надсилання email-нотифікацій персоналу про зміни.

– Спроєктовано адаптивний графічний інтерфейс користувача з використанням шаблонізатора Django Templates та CSS-компонентів Bootstrap, де за допомогою JavaScript-сценаріїв реалізовано динамічний режим перегляду й модифікації даних у межах однієї веб-сторінки рейсу.

– Протестовано роботу бізнес-логіки веб-додатку за допомогою ручного мануального тестування, у ході якого підтверджено працездатність алгоритмів блокування реєстрації понад ліміт місткості автобуса та коректність динамічної GET-фільтрації списків рейсів.

У розділі «Безпека життєдіяльності, основи охорони праці» проаналізовано умови праці диспетчерів та адміністраторів веб-системи за комп'ютеризованими робочими місцями. Висвітлено потенційні небезпечні та шкідливі виробничі фактори та розроблено комплекс інженерно-технічних заходів із забезпечення оптимальних параметрів мікроклімату, освітленості, захисного заземлення й режимів праці та відпочинку персоналу.

ПЕРЕЛІК ДЖЕРЕЛ

1 Holovina, O. (2023). Сучасні технології в управлінні транспортною логістикою. International Science Journal of Management, Economics & Finance, 2(3), 35-42.

2 Мізюк, О. (2024). Путівник мовою програмування Python.[Електронний ресурс]. *Режим доступу: https://pythonguide. rozh2sch. org. ua*.

3 Документація Python [електронний ресурс] / — режим доступу: https://docs.python.org/3.14/– Назва з екрану.

4 Фреймворк Python Django для створення [електронний ресурс] / — режим доступу: https://wezom.com.ua/ua/blog/razrabotka-sajtov-na-python-django – Назва з екрану.

5 Django ORM - Inserting, Updating & Deleting Data [електронний ресурс] / — режим доступу: https://www.geeksforgeeks.org/python/django-orm-inserting-updating-deleting-data/ – Назва з екрану.

6 Томка, Ю. Я., Талах, М. В., & Ушенко, Ю. О. (2022). Python та Django Full Stack веб-розробка.

7 Документація PostgreSQL [електронний ресурс] / — режим доступу: https://www.postgresql.org/docs/current/ – назва з екрану.

8 Палагін, В. В., Палагіна, О. А., & Зорін, О. С. (2024). Основи Python та програмування електронних систем: навчальний посібник. Черкаси: ЧДТУ.

9 PyCharm [електронний ресурс] / — режим доступу: https://uk.wikipedia.org/wiki/PyCharm.– Назва з екрану.

10 Kendhe, S., Nishad, A., Labhade, D., & Magar, V. (2023). Comparative Analysis of Different Python Editors. Vidhyayana-An International Multidisciplinary Peer-Reviewed E-Journal-ISSN 2454-8596, 8(si7), 562-574.

11 Липак, Г., Кунанець, Н., Дуда, О., & Липак, Т. (2025). Побудова інтерфейсів користувача вебсайту бібліотеки на засадах UX-дизайну. Цифрова платформа: інформаційні технології в соціокультурній сфері, 8(1), 172-192.

12 Шакуров, Є. О., & Пономарьова, В. К. (2021). Сучасні тенденції побудови веб-сайтів.

13 Середа, В. П. АНАЛІЗ ПРОГРАМНИХ ЗАСОБІВ ДЛЯ РОЗРОБКИ ДИЗАЙНУ ВЕБ-САЙТУ. Математичні методи, моделі та інформаційні технології в управлінні, 173.

14 Скляренко, О., Савченко, Я., Литвиненко, Л., & Сушинський, О. (2024). Архітектурні підходи до розробки масштабованих веб-застосунків. *Електронне фахове наукове видання «Кібербезпека: освіта, наука, техніка»*, *4*(24), 341-350.

15 Липак Т. А., Липак Г. І. Оцінка користувацького досвіду (UX Evaluation) при розробці інтерфейсів / Scientific Collection «InterConf»,(235): with the Proceedings of the 5th International Scientific and Practical Conference «Society and Science: Interconnection» (February16-18, 2025; Porto, Portugal)/ comp. by LLC SPC «InterConf».Porto: Kramer, 2025. P.236-239.

16 Остапченко, К. Б. (2022). Бази даних. Комп'ютерний практикум.

17 Доценко, С. І. (2023). Організація та системи керування базами даних.

18 Мержинський, Є. К., & Комазов, П. В.. Побудова інформаційної системи структурного моделювання бізнес-процесів. Науковий вісник Ужгородського національного університету. Серія: Міжнародні економічні відносини та світове господарство, (25 (1)), 162-166.

19 Основи інформаційного моделювання. Методичні вказівки з дисципліни “Інформаційне моделювання” / укладачі: Горошко Ю.В., Цибко Г.Ю., Чернігів: НУЧК, 2024, 85 с

20 Як будувати UML-діаграми [електронний ресурс] / — режим доступу: https://dou.ua/forums/topic/40575/ – Назва з екрану.

21 Кліщ, М., Липак, Г., Кунанець, Н., Пасічник, С., & Липак, Т. Структура інформаційної системи передбачення та інтерпретації зміни стану користувача сервісу. Вісник Національного Університету “Львівська Політехніка”. Інформаційні системи та мережі, 17 (2025). С. 226 - 238.

22 UML для бізнес-моделювання: для чого потрібні діаграми процесів [електронний ресурс] / — режим доступу: https://evergreens.com.ua/ua/articles/uml-diagrams.html – Назва з екрану.

23 Duda, O., Pasichnyk, V., Lypak, H., ...Matsiuk, O., Mudrokha, V. Formation of integrated repositories of social and communication data by consolidating the resources of museums, libraries and archives in smart cities projects. CEUR Workshop Proceedings, 2021, 2870, pp. 1420–1430.

24 Липак Г., Липак Т., Кунанець Н. Проєктування інформаційної системи на основі машинного навчання для збереження та класифікації артефактів документальної спадщини. Вісник Хмельницького національного університету: технічні науки. Т. 334 № 4 (2024). С. 176-182.

25 Парненко, В. С. "Мова програмування Python.", 2023.

26 Документація Django [електронний ресурс] / — режим доступу: https://docs.djangoproject.com – Назва з екрану.

27 Васильєв, О. М. (2022). Програмування мовою Python. Bohdan Books.

28 Івановський, О. А., & Парненко, В. С. (2023). Інформатика. Програмування на PYTHON.

29 Vincent, W. S. (2022). Django for Professionals. Still River Press.

30 Желібо Є.П. Безпека життєдіяльності : підручник / В. В. Зацарний. Київ : Каравела, 2023. 344 с.

31 Безпека життєдіяльності та охорона праці : підруч. / В.В. Сокуренко, О.М. Бандурка та ін. – Харків : ХНУВС, 2021. 308 с

32 Пістун І.П., Кочубей В.І. Практикум з безпеки життєдіяльності. Підручник. Вид-во Університетська книга, 2023. 560 с.

33 ДСТУ ISO/TS 16976-8:2021 Засоби індивідуального захисту органів дихання. Людські чинники. Частина 8. Ергономічні чинники (ISO/TS 16976-8:2013, IDT).

34 Мелех Л.В. Безпека життєдіяльності та охорона праці: навч. посіб. / Мелех Л.В. – Львів: ЛДУ внутрішніх справ. 2022. 219 с.

35 Андрейчук Н.І. Охорона праці : навч. посіб. / Н.І. Андрейчук, Ю.В. Кіт, С.В. Шибанов, О.В. Шерстньова. Львів : Видавництво Львівська політехніка, 2021. 276 с. (Підручник з охорони праці)

ДОДАТКИ

Додаток А

Клас представлення TripDetailView А

```
class TripDetailView(LoginRequiredMixin, PermissionRequiredMixin,
generic.DetailView):
    model = Trip
    context_object_name = "trip_data"
    template_name = "trips/trip_detail.html"
    permission_required = "trips.view_trip"

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context["form"] = TripForm(instance=self.object)
        context["passenger_form"] = AddPassengerForm()
        context["shipment_form"] = AddShipmentForm()
        context["new_passenger_form"] = ContactForm()
        context["new_shipment_form"] = ShipmentForm()
        return context

    def post(self, request, *args, **kwargs):
        self.object = self.get_object()
        if "save_trip" in request.POST:
            form = TripForm(request.POST, instance=self.object)
            if form.is_valid():
                form.save()
            for item in self.object.trip_passengers.all():
                input_name = f"update_seats_{item.id}"
                if input_name in request.POST:
                    try:
                        new_seats =
int(request.POST.get(input_name))
                        if new_seats > 0 and new_seats !=
item.seats_count:
                            item.seats_count = new_seats
                            item.save()
                    except (ValueError, TypeError):
                        pass

        elif "add_passenger" in request.POST:
            passenger_id = request.POST.get("passenger")
            from_address_id =
request.POST.get("passenger_from_address") or
self.object.from_address_id
            to_address_id =
request.POST.get("passenger_to_address") or
self.object.to_address_id
            try:
                seats_count = int(request.POST.get("seats_count",
1))
            except ValueError:
                seats_count = 1
```

```
            if passenger_id:
                obj, created =
TripPassenger.objects.get_or_create(
                    trip=self.object,
                    passenger_id=passenger_id,
                    defaults={
                        'seats_count': seats_count,
                        'from_address_id': from_address_id,
                        'to_address_id': to_address_id
                    }
                )
                if not created:
                    obj.seats_count += seats_count
                    obj.save()

        elif "create_new_passenger" in request.POST:
            p_form = ContactForm(request.POST)
            if p_form.is_valid():
                new_passenger = p_form.save()
                from_address_id =
request.POST.get("passenger_from_address") or
self.object.from_address_id
                to_address_id =
request.POST.get("passenger_to_address") or
self.object.to_address_id

                TripPassenger.objects.create(
                    trip=self.object,
                    passenger=new_passenger,
                    seats_count=request.POST.get("seats_count",
1),
                    from_address_id=from_address_id,
                    to_address_id=to_address_id
                )

        elif "add_shipment" in request.POST:
            shipment_id = request.POST.get("shipment")
            if shipment_id:
                TripShipment.objects.get_or_create(
                    trip=self.object,
                    shipment_id=shipment_id
                )

        elif "create_new_shipment" in request.POST:
            s_form = ShipmentForm(request.POST)
            if s_form.is_valid():
                new_shipment = s_form.save()
                TripShipment.objects.create(
                    trip=self.object,
                    shipment=new_shipment
```

```
                )

        elif "remove_passenger" in request.POST:
            record_id = request.POST.get("remove_passenger")
            TripPassenger.objects.filter(id=record_id,
trip=self.object).delete()

        elif "remove_shipment" in request.POST:
            record_id = request.POST.get("remove_shipment")
            TripShipment.objects.filter(id=record_id,
trip=self.object).delete()

        return redirect("trips:trip-detail", pk=self.object.pk)
```

Додаток Б

Фрагменти програмної реалізації шаблону картки рейсу

```
{% extends "base.html" %}
{% load static %}

{% block content %}
<link rel="stylesheet" href="{% static 'css/styles.css' %}">
<div class="page-header full-width-container">
    <div>
        <h2>
            {{ trip_data.from_address.city }} → {{
trip_data.to_address.city }}
        </h2>
        <span class="badge status-{{ trip_data.status }}">
            {{ trip_data.get_status_display }}
        </span>
    </div>
    <div class="page-actions" style="display: flex; gap: 8px;">
        {% if perms.trips.change_trip %}
            <button type="button" class="btn btn-dark" data-edit-
btn>Редагувати картку</button>
            <button type="button" class="btn btn-light" data-
cancel-btn style="display:none;">Скасувати</button>
            <button type="submit" form="trip-form" class="btn btn-
success" data-save-btn style="display:none;">Зберегти
зміни</button>
        {% endif %}

        {% if perms.trips.delete_trip %}
            <a href="{% url 'trips:trip-delete' trip_data.id %}"
class="btn" style="background-color: #dc3545; color: white;
display: none; text-decoration: none; line-height: 24px;" data-
delete-btn>
                Видалити рейс
            </a>
        {% endif %}
    </div>
</div>
<form method="post" id="trip-form" class="trip-form full-width-
container" data-edit-form>
    {% csrf_token %}
    <input type="hidden" name="passenger" id="passenger-hidden-
id">
    <input type="hidden" name="shipment" id="shipment-hidden-id">
    <div class="form-section">
        <h3>Основна інформація</h3>
        <div class="form-row">
            <div class="form-group">
            <label>Водій</label>
            <select name="driver" disabled class="form-control">
```

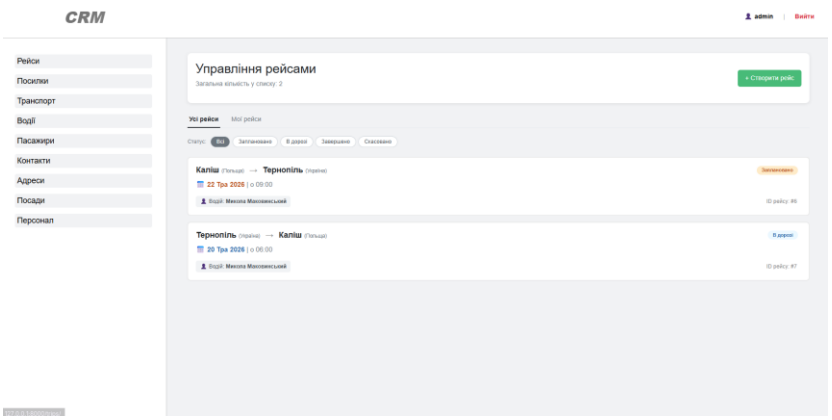
```
                    {% for d in form.fields.driver.queryset %}
                        <option value="{{ d.id }}" {% if d.id ==
trip_data.driver.id %}selected{% endif %}>
                            {{ d }}
                        </option>
                    {% endfor %}
                </select>
            </div>
            <div class="form-group">
                <label>Авто</label>
                <select name="vehicle" disabled class="form-
control">
                    {% for v in form.fields.vehicle.queryset %}
                        <option value="{{ v.id }}" {% if v.id ==
trip_data.vehicle.id %}selected{% endif %}>
                            {{ v }}
                        </option>
                    {% endfor %}
                </select>
            </div>
            <div class="form-group">
                <label>Статус</label>
                <select name="status" disabled class="form-
control">
                    {% for key, val in trip_data.STATUS_CHOICES %}
                        <option value="{{ key }}" {% if
trip_data.status == key %}selected{% endif %}>
                            {{ val }}
                        </option>
                    {% endfor %}
                </select>
            </div>
        </div>
        <div class="form-row" style="margin-top: 15px;">
            <div class="form-group">
                <label>Дата початку</label>
                <input type="datetime-local" name="start_time"
value="{{ trip_data.start_time|date:'Y-m-d\TH:i' }}" disabled
class="form-control">
            </div>
            <div class="form-group">
                <label>Дата завершення</label>
                <input type="datetime-local" name="end_time"
value="{{ trip_data.end_time|date:'Y-m-d\TH:i' }}" disabled
class="form-control">
            </div>
        </div>
    </div>
    <div class="form-section">
        <h3>Маршрут</h3>
```

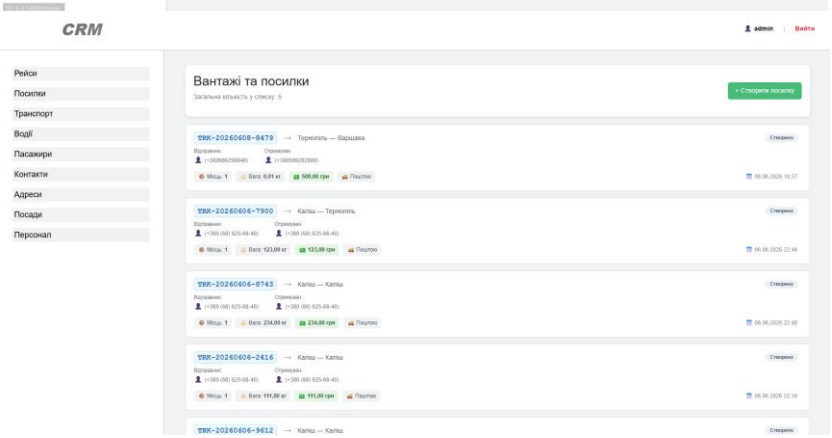
```
        <div class="form-row">
            <div class="form-group">
                <label>Звідки</label>
                <select name="from_address" disabled class="form-
control">
                    {% for a in form.fields.from_address.queryset
%}
                        <option value="{{ a.id }}" {% if a.id ==
trip_data.from_address.id %}selected{% endif %}>
                            {{ a }}
                        </option>
                    {% endfor %}
                </select>
            </div>
            <div class="form-group">
                <label>Куди</label>
                <select name="to_address" disabled class="form-
control">
                    {% for a in form.fields.to_address.queryset %}
                        <option value="{{ a.id }}" {% if a.id ==
trip_data.to_address.id %}selected{% endif %}>
                            {{ a }}
                        </option>
                    {% endfor %}
                </select>
            </div>
        </div>
    </div>
    <div class="form-section">
        <div class="section-header">
            <h3>Пасажири на рейсі</h3>
        </div>
        <div class="table-wrapper">
            <table class="custom-table">
                <thead>
                    <tr>
                        <th>Пасажир</th>
                        <th>Телефон</th>
                        <th>Адреса посадки</th>
                        <th>Адреса висадки</th>
                        <th>Кількість місць</th> <th>Додано до
рейсу</th>
                        {% if perms.trips.change_trip %}
                            <th class="action-column"
style="display: none;">Дії</th>
                        {% endif %}
                    </tr>
                </thead>
                <tbody>
```

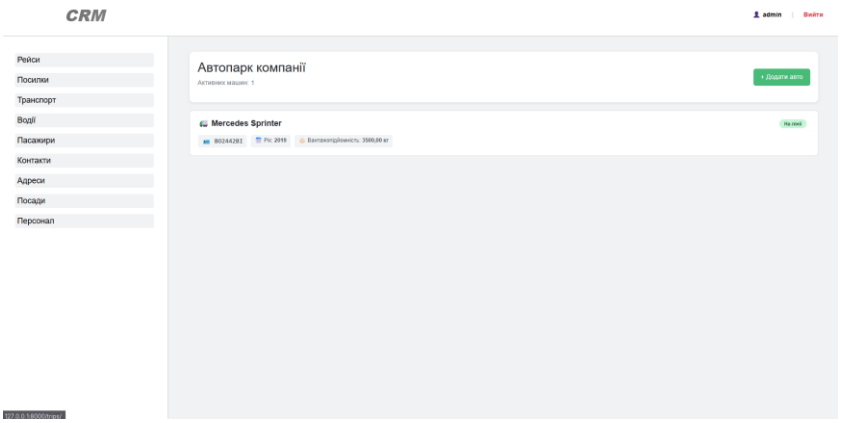
```
                    {% for item in trip_data.trip_passengers.all
%}
                        <tr>
                            <td>
                                <a href="{% url 'trips:passenger-
edit' item.id %}" class="table-link">
                                    <strong>{{ item.passenger
}}</strong>
                                </a>
                            </td>
                            <td>{{ item.passenger.phone }}</td>
                            <td><span class="address-text">{{
item.from_address|default:"Не вказано" }}</span></td>
                            <td><span class="address-text"
style="color: #28a745;">{{ item.to_address|default:"Не вказано"
}}</span></td>
                            <td>
                                <input type="number"
                                       name="update_seats_{{
item.id }}"
                                       value="{{ item.seats_count
}}"
                                       min="1"
                                       style="width: 70px;
padding: 4px; border-radius: 4px; border: 1px solid #ccc;"
                                       disabled
                                       data-seats-input>
                            </td>
                            <td><span class="text-muted">{{
item.loaded_at|date:"d.m.Y H:i" }}</span></td>
                            {% if perms.trips.change_trip %}
                                <td class="action-column"
style="display: none;">
                                    <button type="submit"
name="remove_passenger" value="{{ item.id }}" class="btn-delete-
icon" title="Видалити з рейсу">
                                        ×
                                    </button>
                                </td>
                            {% endif %}
                        </tr>
                    {% empty %}
                        <tr>
                            <td colspan="7" style="text-align:
center; color: #888; padding: 20px;">Пасажирів ще не додано до
цього рейсу</td>
                        </tr>{% endfor %}
</tbody></table></div>
```

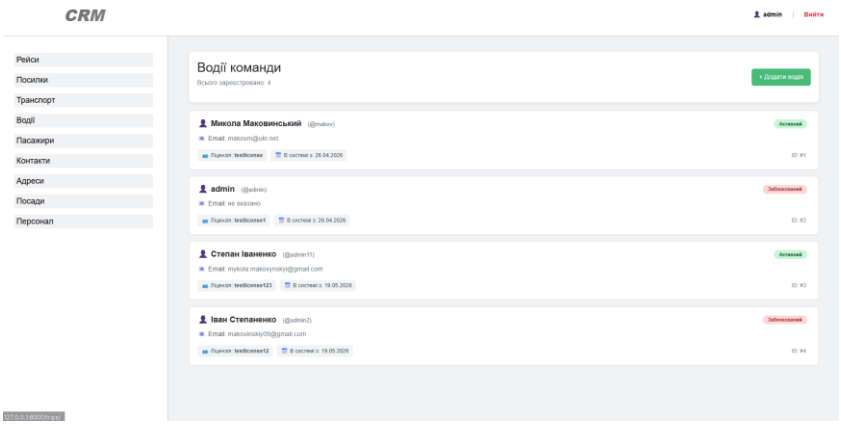
Додаток В

Знімки інтерфейсу реалізованого веб-системи

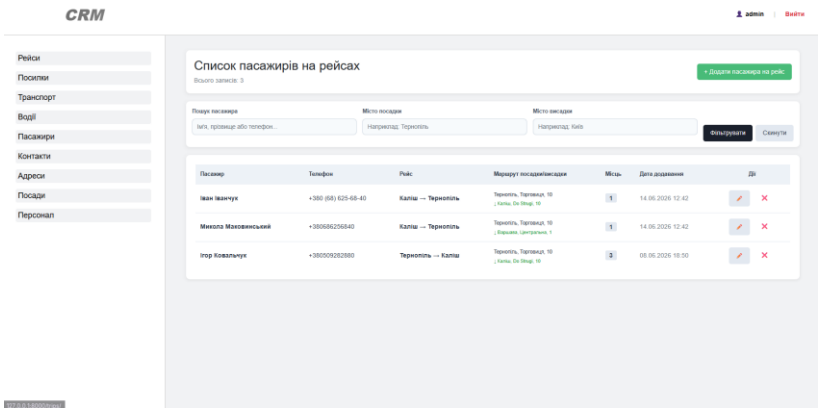


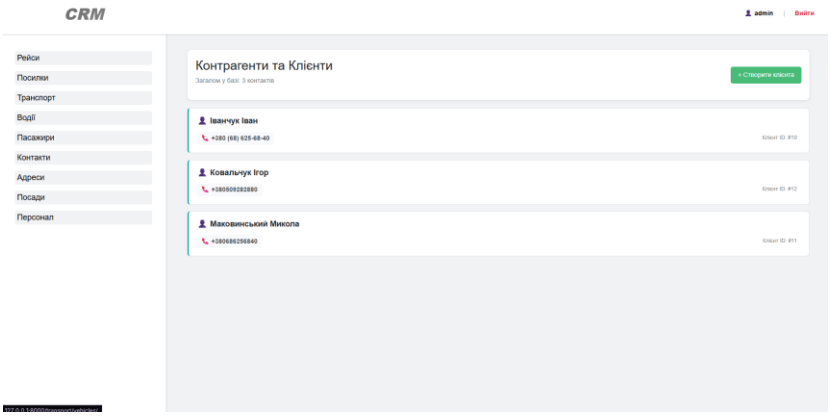


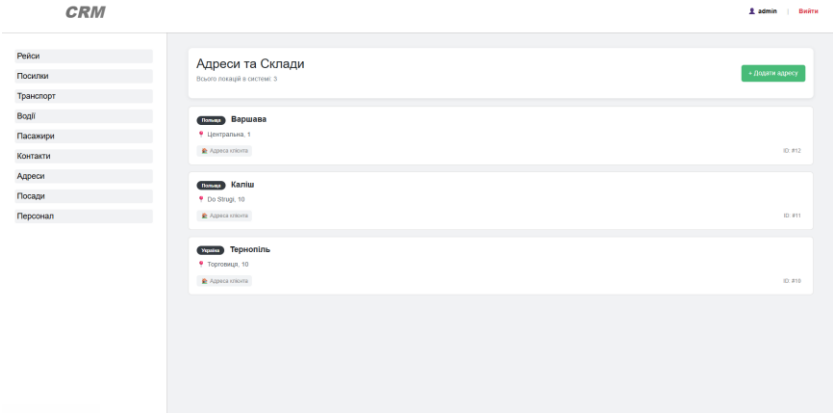


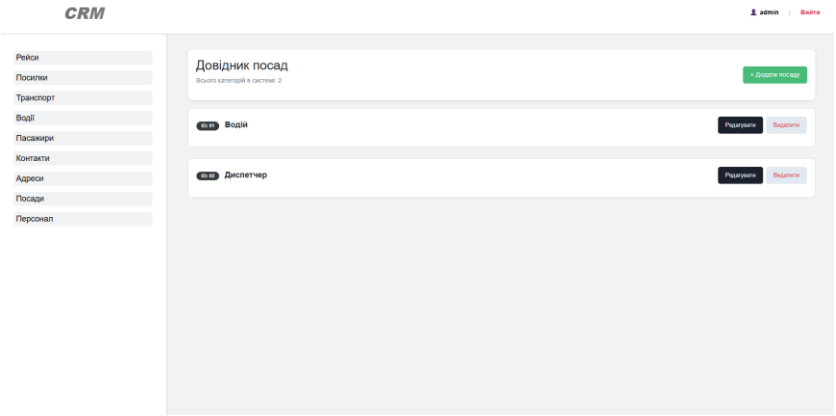


Продовження додатку В









Продовження додатку В